UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

PROJETO II – Sistemas de Informação

Padrões de Segurança Aplicados à Linguagem XML

Florianópolis, 2005

Luiz Henrique Wiggers Kato
Marcos Hideki Watanebe de Moraes

PADRÕES DE SEGURANÇA APLICADOS À LINGUAGEM XML

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como um dos pré-requisitos para obtenção do grau de bacharel em Sistemas de Informação.

Florianópolis, 2005

Luiz Henrique Wiggers Kato
Marcos Hideki Watanabe de Moraes

“PADRÕES DE SEGURANÇA APLICADOS À LINGUAGEM XML”

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. João Bosco Mangueira Sobral

Banca examinadora:

_____
Profa. Kathia Regina Lemos Jucá

_____
Profa. Mirela Sechi Moretti Anonni Notare

KATO, Luiz Henrique Wiggers; MORAES, Marcos Hideki Watanabe de. **PADRÕES DE SEGURANÇA APLICADOS À LINGUAGEM XML**. 2005. 118 f. Trabalho de Conclusão de Curso, Curso de Bacharelado em Sistemas de Informação, Universidade Federal de Santa Catarina, Florianópolis (SC), 2005.

Sumário

Lista de Figuras

Lista de Siglas

AC	Autoridade Certificadora
CRL	*Certification Revocation List*
DNS	*Domain Name System*
HTTP	*Hyper Text Transfer Protocol*
LRA	*Local Registration Authority*
MRA	*Master Registration Authority*
PAP	*Policy Administration Point*
PDP	*Policy Decision Point*
PEP	*Policy Enforcement Point*
PGP	*Pretty Good Privacy*
PIP	*Policy Information Point*
PKI	*Public Key Infrastructure*
POP	*Post Office Protocol*
S/MIME	*Secure / Multipurpose Internet Mail Extensions*
SAML	*Security Assertion Markup Language*
SOAP	*Simple Object Access Protocol*
URI	*Uniform Resource Indicator*
URN	*Uniform Resource Name*
W3C	*The World Wide Web Consortium*
XACML	*Extensible Access Control Markup Language*
XKISS	*XKMS Key Information Service Specification*
XKMS	*XML Key Management*
XKRSS	*XKMS Key Registration Service Specification*
XML	*Extensible Markup Language*
XSLT	*Extensible Stylesheet Language Transformations*

Resumo

Existem muitos sistemas operando sobre a Rede Mundial de Computadores (Internet), cada qual com seu próprio formato de documentos para troca de informações, controle de acesso, confiabilidade e integridade dos dados. Tantos formatos não são compatíveis entre si, sendo uma regra de acesso ou um documento assinado entendido apenas pelo sistema para o qual foi criado. Além da falta de integração, um problema encontrado é a dificuldade no desenvolvimento de aplicações com suporte as funcionalidades do *PKI*, devido a complexidade das operações. Descrevemos quatro padrões de segurança aplicados a linguagem *XML*, que tornam os documentos intercambiáveis e abstraem a camada *PKI*, diminuindo o tempo de desenvolvimento das aplicações e permitindo a reutilização de regras de controle de acesso para os mesmos recursos. A fim prático, demonstramos a utilização da biblioteca '*xmlsec'*, que implementa dois destes quatro padrões (Criptografia *XML* e Assinatura *XML*). Com o uso destes padrões é possível a aplicação de segurança aos sistemas de forma geral, padronizada e mais simplificada, diminuindo o uso de tempo e recursos em novas implementações.

Palavras-chave: segurança, xml, criptografia, assinatura digital

Abstract

There are many systems operating on the Internet, each one with his own documents format for information exchange, access control, reliability and data integrity. So many formats are not compatible amongst themselves, so an access rule or a signed document is just compatible with the system for which was created. Besides the integration lack, a problem found is the difficulty in the development of applications with support the PKI functionalities, due to complexity of the operations. We described four security patterns applied to the XML language, that turn the documents interchangeable and abstract the PKI layer, reducing the development time of applications and allowing the reuse of access control rules for the same resources. In pratical, we demonstrated the use of the 'xmlsec' library, that implements two of these four patterns (XML Encrypt and XML Signature). With these patterns it is possible to apply security to systems in a general way, standardized and more simplified, reducing the use of time and resources in new implementations.

Keywords: security, xml, cryptography, digital signature

Capítulo 1 Introdução

1.1 Justificativa

As cifras envolvendo transações que utilizam a *Internet* elevam a segurança a um nível de preocupação muito grande, e este estudo apresenta uma maneira de incrementar a confiabilidade e integridade das informações que trafegam pelas redes de forma padronizada e intercambiável utilizando a linguagem *XML*.

As tecnologias de segurança *XML* se encaixam nos *Web Services*, e como estes estão se tornando o padrão para sistemas distribuídos, estas tecnologias tendem a ser amplamente utilizadas.

1.2 Objetivos

São apresentadas aqui as tarefas desenvolvidas e a motivação para realização do trabalho.

1.2.1 Objetivo Geral

O objetivo geral é apresentar meios de incrementar a segurança de sistemas de forma geral, padronizada entre eles e que facilite a implementação destes sistemas. O estudo descreve o modo de funcionamento e estrutura dos padrões de segurança na linguagem *XML* propostos, assim como a utilização de alguns destes padrões de forma prática, utilizando uma biblioteca que implementa estas especificações.

O principal objetivo é a aplicação destes padrões em sistemas distribuídos, com foco nos Web Services, que estão se tornando a base para sistemas distribuídos na Internet.

Estes padrões permitem tornar documentos intercambiáveis, abstraem a camada *PKI*, diminuindo o tempo de desenvolvimento das aplicações e permitem a reutilização de regras de controle de acesso para os mesmos recursos, diminuindo o uso de tempo e recursos na reescrita de regras para cada diferente sistema.

1.2.2 Objetivo Específico

Com base no estudo realizado, é demonstrado na prática o desenvolvimento de programas simples, que devem garantir a integridade e confiabilidade das mensagens e das partes envolvidas. Os padrões estudados são:

- *XML Signature*
- *XML Encryption*
- *XML Key Management (XKMS)*
- *Extensible Access Control Markup Language (XACML)*

As atividades realizadas são:

- Entender os termos e funcionamento da criptografia;
- Descrever a estrutura e principais componentes dos padrões *XML* propostos;
- Descrever a seqüência de passos a serem executados para a correta aplicação destes padrões;

- Demostrar a utilização de uma biblioteca que implementa a especificação dos padrões '*XML Signature*' e '*XML Encryption*';
- Exibir os documentos *XML* de entrada, e os documentos após serem assinados ou criptografados;

1.3 Contexto e Tecnologia

São apresentadas as linguagem utilizadas e o contexto da aplicação dos padrões em questão.

1.3.1 Contexto do trabalho

O estudo dos padrões de segurança utilizando a linguagem *XML* e suas aplicações reais podem ser aplicadas em qualquer área computacional que necessite de uma ou mais características que a segurança da informação pode fornecer, como por exemplo:

- autoria
- integridade
- sigilo
- autorização

Assim como meio para alcançar um ambiente intercambiável de regras e dados por aplicações que suportem os padrões *XML* utilizados.

1.3.2 Tecnologias

Utilizamos como linguagem de programação o *C*, por seu amplo suporte as tecnologias *XML* e criptografia. A linguaguem *XML* é utilizada já que é a base dos padrões estudados, e será melhor descrita a seguir.

A biblioteca '*xmlsec*' é escrita na linguagem *C, e* implementa os padrões de Criptografia e Assinatura *XML*. O padrão para Gerenciamento de Chaves Pública está em desenvolvimento nesta biblioteca, mas ainda não é funcional.

Capítulo 2 Conceitos de Criptografia

2.1 Introdução

O objetivo aqui não é, de maneira alguma, se aprofundar em criptografia, estudar técnicas de criptoanálise, muito menos destrinchar algoritmos de criptografia. Mas, por se tratar de um trabalho realizado no campo da Segurança da Informação, devem ser apresentados aqui alguns conceitos básicos sobre Criptografia, que é a base de qualquer sistema seguro.

Neste capítulo serão apresentados fundamentos em Criptografia, como criptografia de chaves simétricas, criptografia de chaves assimétricas, funções resumo ou funções hash e o funcionamento das assinaturas digitais.

2.1.1 Fluxo de informações

O fluxo normal de informação de uma fonte A para um destino B, sem solução de continuidade, e sem qualquer interferência, é a situação ideal que se pretende alcançar em uma comunicação eletrônica segura.

Segundo Stallings [STA 99] existem várias formas de ataques em que um agente malicioso inimigo I pode impedir, interferir ou detectar as informações transmitidas de A para B.

A seguir serão apresentadas as quatros formas de deturpação no fluxo de informações: interrupção, modificação, interceptação e fabricação.

Interrupção

A informação parte da fonte A, porém não chega ao chega ao destino B, pois o fluxo é interrompido no meio do caminho, por algum motivo como, por exemplo, queda de energia ou problemas no meio físico de transmissão.

Modificação

O fluxo parte da fonte A, mas é interceptado por um inimigo I, que modifica a informação e envia para o destino B. Este tipo de ataque visa afetar a *integridade* dos dados.

Fabricação

O inimigo entra no circuito de comunicação e envia uma mensagem para o destino B, se passando pela fonte A, ou seja, a informação é fabricada por I, enviada para B, e este a recebe como sendo de autoria de A.

Interceptação

A informação gerada na fonte A chega normalmente até o destino B. O fluxo de informação aparentemente é normal, exceto pelo fato de haver um inimigo I monitorando o fluxo e que também conseguiu ter acesso à informação, sem que A ou B percebam. Este tipo de ataque afeta a confidencialidade dos dados.

2.1.2 Terminologia

Para que um sistema ofereça um fluxo seguro da informação a partir de A para B, ele deve atender principalmente os seguintes requisitos:

Confidencialidade – ou sigilo, é a garantia de que somente as partes envolvidas na comunicação possam ler e utilizar as informações transmitidas eletronicamente pela rede. Isto é, caso um inimigo consiga interceptar de alguma forma o fluxo de informação, a mensagem transmitida deve ser indecifrável;

Integridade – garantia de que o conteúdo de uma mensagem não será alterado durante seu tráfego, sem sofrer alterações, isto é, os dados que foram gerados em A são os dados que foram recebidos por B;

Autenticação – garantia da identificação das partes envolvidas na comunicação, tendo certeza de que a comunicação esteja realmente sendo feita entre A e B, ou seja, que não há nenhum agente malicioso se fazendo passar por A para gerar uma informação falsa, ou se passando por B para simular a recepção da mensagem enviada por A;

Não repúdio – garantia de que o emissor de uma mensagem não poderá, posteriormente, negar sua autoria.

2.2 Criptografia

[BUR02] "A Criptografia converte dados legíveis em algo sem sentido, ilegível. Estes dados poderão ser recuperados posteriormente a partir desses dados sem sentido".

Criptografia é uma palavra originada no grego, *kriptos* = escondido, oculto e *grifo* = grafia ou escrita, e é a arte ou ciência de escrever em cifras ou códigos.

A criptografia é uma das ferramentas mais importantes para proteção dos dados, e a idéia principal é transformar um conjunto de dados legível em algo ilegível, garantindo que o segredo ali codificado permanecerá confidencial.

Apenas a pessoa que possuir a chave para decifrar poderá reverter essa operação, ou seja, tornar esses dados ilegíveis em algo que faça sentido novamente.

Desde a Antigüidade, tão logo o homem aprendeu a escrever, ele começou a pensar em alguma forma de esconder o que ele havia escrito. Foi assim que começou a se desenvolver a Criptografia, base tecnológica para problemas de segurança em comunicações e em computação. Hoje em dia, esta ciência utiliza conceitos e fundamentos matemáticos para a construção de seus algoritmos. Um estudo sobre aritmética modular e teoria dos números primos é indispensável para quem deseja conhecer mais profundamente o assunto.

Porém, é fato que não existe alguém que diga que não precisa de segurança ou que não tenha algum segredo para ocultar. Todos nós temos informações que queremos manter em segredo, seja pelo simples desejo de privacidade ou até por autoproteção. Então, nenhuma pessoa ou empresa, quer ver seus dados sigilosos caindo em mãos erradas e maliciosas.

A Criptografia, de modo algum, é a única tecnologia necessária para assegurar os dados, nem resolverá todos os problemas relacionados à segurança. É apenas um instrumento, dentre outros existentes. Infelizmente, além disso, a Criptografia não é à prova de falhas, o que significa que por mais ininteligível que uma cifra seja, ainda assim existe a possibilidade dela ser quebrada.

2.2.1 Criptografia Simétrica

É a forma mais convencional de criptografia que existe. Por isso, também é conhecida por Criptografia Clássica.

Os algoritmos de criptografia de chaves simétricas são assim denominados por apresentarem como principal característica o fato de utilizarem apenas uma chave. Isto significa que a mesma chave que é utilizada para cifrar é também usada para decifrar. Os algoritmos para cifrar e decifrar também são os mesmos. O que muda é apenas a forma como são utilizadas as chaves. É tido como pré-requisito que os usuários envolvidos compartilhem esta chave entre si. A troca de chaves é geralmente feita através do estabelecimento de um canal de comunicação seguro, através de tunelamento, entre as duas partes envolvidas.

Existem vários algoritmos de criptografia simétrica, os mais conhecidos e utilizados atualmente são:

AES - *Advanced Encryption Standard*, um algoritmo concebido em outubro de 2000 para substituir o famoso, e cada vez mais vulnerável, algoritmo *Data Encryption Standard* (*DES*) e *3DES*.

Blowfish - cifrador de bloco simétrico desenvolvido por Bruce Schneier, em 1993, e permanece inquebrável até hoje. Possui as características de ser rápido, compacto, simples e possuir o tamanho de sua chave variando entre 32 e 448 *bits*. Opera com blocos de 64 *bits*.

O principal problema encontrado ao utilizar a criptografia simétrica é a questão do gerenciamento das chaves. Para cada par de usuários que desejam se comunicar é necessário uma chave, o que torna o gerenciamento muito complexo. Em uma rede com n usuários, serão necessárias n(n-1)/2 chaves para que todos os usuários possam trocar informações de maneira sigilosa entre si.

2.2.2 Criptografia Assimétrica

Este tipo de criptografia utiliza duas chaves diferentes: uma usada para cifrar o texto na origem e a outra usada para decifrar o texto quando este chegar no destino. Este sistema gera um par de chaves que são, na verdade, números primos muito grandes relacionados entre si, através de propriedades da aritmética modular.

Uma das chaves será chamada de chave privada e deverá ser mantida em sigilo. A outra chave será chamada de chave pública e deverá ser compartilhada com outras pessoas.

Se uma pessoa A cifra uma informação com sua chave privada, outra pessoa B somente poderá decifrar a mensagem com a chave pública de A, tendo a certeza de que esta mensagem realmente teve sua autoria em A, pois somente A possui a chave privada. Por outro lado, se A cifrar uma mensagem com a chave pública de B, somente B poderá decifrar a mensagem com sua chave privada, garantindo desta forma o sigilo ou confidencialidade da mensagem.

A grande vantagem da criptografia assimétrica é a eficiência no sigilo. Porém, como os algoritmos de criptografia assimétrica são um pouco lento quando, em sua entrada, apresentam volume muito grande de dados, para cifragem de mensagens grandes ela é ineficiente se for levado em conta o parâmetro tempo que tal processo leva. O que muito se faz é empregar a criptografia assimétrica para realizar a troca de

chave de criptografia simétrica, de modo que esta última é cifrada com chaves assimétricas, podendo ser transmitida eletronicamente pela rede de maneira de segura.

2.3 Funções de Resumo

Como os algoritmos de criptografia assimétrica são lentos, não é uma boa idéia cifrar um grande volume de dados, como um texto inteiro.

Na prática, o que geralmente se faz é gerar um resumo dos dados e depois cifrar este resumo utilizado a chave privada do autor dos dados. Estes resumos são gerados através das chamadas funções de *hash*, e constituem uma representação da mensagem. Isto significa que, se o resumo possui essencialmente o mesmo valor de identidade da mensagem, é muito mais fácil trabalhar com um resumo criptográfico cujo tamanho é de alguns *bytes* (geralmente 16 ou 24 *bytes*), do que manipular mensagens demasiadamente grandes.

Tais funções são um algoritmo que recebe qualquer comprimento de entrada e mescla a entrada para produzir uma saída pseudo-aleatória de tamanho fixo.

A palavra "*hash*" pode significar desordem ou confusão, o que descreve eficientemente o resultado de uma mensagem resumida.

Uma propriedade verificável nestas funções é que elas são irreversíveis, isto é, não é possível reconstruir o conjunto original de dados a partir do resumo. Outra particularidade que é importante saber é que até hoje não foram verificadas colisões – termo técnico empregado para descrever uma situação em que duas mensagens produzem um mesmo resumo. Elas existem, mas ninguém consegue encontrar uma colisão por demanda.

Os dois algoritmos mais importantes e conhecidos de resumo, dentre os vários existentes, são:

MD5 - Message Digest 5

Criado por Ron Rivest, do *Massachussets Institute of Technology – MIT*, é o sucessor mais bem-sucedido do *MD2*. Produz um resumo de 128 *bits*, porém é mais robusto e rápido que seu antecessor. Sabe-se de potenciais vulnerabilidades que podem ser exploradas e atualmente está o seu uso tem sido desencorajado por profissionais da área de Criptografia, pois já foram encontradas colisões. Mesmo assim, por ter sido muito utilizado e ainda estar muito presente nos sistemas, vale a pena ser lembrado.

SHA-1 - Secure Hash Algorithm

Parecido com o *MD5*, porém mais forte. Sua saída produz um resumo de 160 *bits*, o que significa em uma probabilidade menor de ocorrência de colisões, pois apresenta um universo de 2160 resumos possíveis.

É altamente recomendado pela comunidade de criptografia, e é o resumo mais utilizado hoje em dia em certificados digitais e bibliotecas criptográficas.

A grande vantagem de utilizar resumos criptográficos é que mensagens muito grandes podem ser reduzidas a poucos *bytes*.

Com resumos é possível garantir a integridade dos dados, isto é, é possível saber se um conjunto de dados foi alterado a partir da comparação de resumos criptográficos.

Se você está preocupado com o fato de que as informações podem ser alteradas, envie seus dados juntamente com um resumo. Se os dados forem alterados, o resumo de verificação também será diferente, e você saberá que alguma coisa aconteceu. Obviamente é preciso se assegurar de que o valor do resumo não pode ser alterado para corresponder com quaisquer alterações na mensagem. É neste ponto, portanto, que entra a assinatura digital, que consiste basicamente no procedimento de cifrar o resumo com a chave privada do assinante.

2.4 Assinatura Digital

Verificar a integridade dos dados muitas vezes não basta. É preciso também poder garantir a autoria destes.

Baseadas na aplicação de técnicas de criptografia assimétrica, as assinaturas digitais foram feitas para que uma entidade possa, digitalmente, "assinar" um documento eletrônico como, por exemplo, um arquivo texto ou um e- mail, comprovando de forma única e exclusiva a autoria dos dados. Espera-se que uma assinatura digital possua as mesmas características de uma assinatura qualquer no mundo real. Estas características desejáveis são:

1. A assinatura digital deve ser fácil de produzir por quem assina
2. Deve ser fácil de ser verificada por qualquer pessoa
3. Deve ser muito difícil de ser falsificada
4. Quem assine não possa negar que assinou (não-repúdio)

Além das quatro características citadas acima, é requisito fundamental que a assinatura digital dependa do conteúdo assinado. Isto é, a assinatura deve sempre ser diferente para diferentes informações assinadas. É nesta propriedade que se encontra o grande poder das assinaturas digitais. Senão, é possível imaginar como seria fácil de forjar a assinatura: bastaria o inimigo pegar a assinatura em um outro documento assinado e utilizá- la no documento o qual ele deseja falsificar.

É importante também saber que todo conjunto de dados cifrados com uma chave privada é uma assinatura digital, pois só pode ter sido gerado pela pessoa que possui a chave privada.

Forjar uma assinatura digital significa que alguém é capaz de criar uma massa de dados, por um outro meio, que fosse idêntica à assinatura. Isso significaria que o forjador quebrou o algoritmo *RSA*, o que é altamente improvável. É nisso que são baseadas as propriedades do não-repúdio e não-falsificação.

É até possível alegar que a chave privada foi roubada, o que tornaria possível uma outra pessoa gerar uma assinatura digital autêntica. Porém, existem meios de revogar um certificado digital (o que inclui invalidar a chave privada a ele associado) e também é possível proteger a chave privada.

As assinaturas digitais têm sido implementadas basicamente utilizando conceito de chaves públicas com o padrão *RSA*, para cifrar resumos gerados a partir de funções de hash como *MD5* e *SHA-1*. Existem também o *DSS - Digital Signature Standard* – que é um algoritmo muito utilizado para criação de assinaturas digitais e existe um esforço para torná- lo um padrão para tal finalidade.

A chave privada é utilizada pelo assinante para assinar, enquanto a chave pública é utilizada para verificar a autenticidade da assinatura.

É muito importante esclarecer que uma assinatura digital não é feita para proteger uma mensagem contra um suposto inimigo. Ela tem como principal finalidade garantir que uma determinada pessoa, a autora da assinatura, realmente tenha assinado tal mensagem.

Todos que quiserem verificar a assinatura terão que ter acesso à assinatura e à mensagem. O inimigo, então, poderá verificar a assinatura. A idéia é evitar apenas que se falsifique a assinatura.

Para fins de confidencialidade e se gurança das informações devem ser utilizadas técnicas de criptografia simétrica.

Uma maneira de criar assinaturas verificáveis é cifrar o resumo criptográfico com a chave privada *RSA* do assinante.

2.5 Certificados Digitais

Pergunta: Como alguém pode verdadeiramente saber se uma chave pública pertence a uma determinada pessoa em questão?

A maneira mais comum de verificar isso é por meio de um certificado digital, que associa um nome à uma chave pública. Uma analogia para compreender isso melhor é o passaporte, que associa uma foto, a um número e a um nome.

Um certificado digital é produzido de tal maneira que torna perceptível se um impostor pegou um certificado existente e substituiu a chave pública ou o nome ali contido.

Qualquer pessoa ao examinar esse certificado fraudado saberá que algo está errado e, portanto, não confiará na combinação desse par de chaves e nome.

2.5.1 Funcionamento

Pegue um nome e uma chave pública, concatene esses dois dados e assine-os.

Isto é um certificado digital. Geralmente a assinatura é feita por uma Autoridade Certificadora (AC), que é responsável por gerenciar as chaves públicas e os certificados digitais de terceiros.

As ACs servem como terceiros confiáveis para associar uma identidade de uma pessoa a sua chave pública. A AC tem como principal tarefa a autenticação de seus usuários finais. Para isso, é preciso que a AC forneça sua própria chave pública para todos os usuários finais certificados, bem como para todas as partes verificadoras que podem utilizar as informações certificadas, pois para poder validar um certificado digital é necessário ter a chave pública da AC.

Os certificados digitais constituem um meio seguro de distribuir a chave pública para quem quiser verificar sua assinatura digital. O formato de certificado mais amplamente aceito é o X.509 v3 da *International Telecommunications Unions* (ITU-T), e apresenta alguns campos como: versão, número serial, identificador do algoritmo de assinatura, nome do emissor (AC), validade, nome do sujeito, chave pública, identificação do algoritmo de chave pública, etc.

2.5.2 Revogando um certificado digital

Dentre os dados que são armazenados dentro de um certificado digital está a data de validade. Uma assinatura digital gerada com uma chave privada expirada não será considerada válida no momento de sua verificação.

Os certificados são criados acreditando-se que estes serão válidos e usáveis durante todo o tempo de vida estimado no campo de *Validade*. Entretanto, em alguns casos, um certificado ainda em vigor poderá não ser mais utilizado.

Por exemplo, a chave privada pode ter sido comprometida de alguma forma.

Caso o certificado digital de uma pessoa for extraviado, isto é, sua chave privada for roubada, ou ainda se for o caso de alguém perder o seu certificado digital, é possível entrar com um pedido de revogação do mesmo.

O método mais comum de se invalidar certificados é a *Certification Revocation List – CRL* – que é uma lista dos certificados revogados em cada Autoridade Certificadora.

Desta forma é possível constatar junto com a AC se o certificado é válido ou não, antes de gerar ou validar uma assinatura digital e verificar se o certificado digital se encontra ou não na *CRL*.

Similarmente, também é possível suspender por um tempo determinado a validade de um certificado digital.

Capítulo 3 *XML*

3.1 Introdução

XML é uma linguagem de marcação para documentos que contêm informação estruturada.

Uma linguagem de marcação é um mecanismo para identificar estruturas em um documento. A especificação *XML* define um modo padrão para acrescentar marcação em documentos.

3.2 Documento

O número de aplicações desenvolvidas atualmente que são baseadas ou fazem uso de documentos *XML* é muito grande. Para nossos propósitos, a palavra "documento" indica não só documentos tradicionais, como este, mas também para outros formatos de dados. Estes incluem gráficos de vetor, transações de comércio eletrônico, equações matemáticas, meta-dados, *APIs*, e mil outros tipos de informação estruturada.

3.3 *XML e HTML*

No *HTML*, a semântica de *tag* e o conjunto de *tags* são fixas. Um `<h1>` sempre é o primeiro nível de cabeçalho e a *tag* <ati.product.code> não faz sentido. O *W3C*, junto com os desenvolvedores de *browsers* e a comunidade *WWW*, estão constantemente trabalhando para estender a definição de *HTML* para permitir que novas *tags* acompanhem as mudanças da tecnologia e trazer variações de apresentação (*stylesheets*) para o *Web*. Porém, estas mudanças sempre são limitadas pelo que os desenvolvedores de *browser* programaram e pelo fato que compatibilidade é limitada.

O *XML* não especifica nem semântica nem um conjunto de *tags*. Na realidade o *XML* é uma meta-linguagem para descrever linguagens de marcação. Em outras palavras, *XML* provê uma facilidade para definir *tags* e as relações estruturais entre eles. Já que não há nenhum conjunto de *tags*, não pode haver nenhuma semântica preconcebida. Tudo na semântica de um documento *XML* ou será definido pelas aplicações que o processa ou através de *stylesheets*.

3.4 *XML e SGML*

O *XML* é definido como um perfil de aplicação de *SGML*. *SGML* (*Standard Generalized Markup Language*) é um padrão generalizado de linguagem de marcação definido pela ISO 8879. *SGML* foi a maneira padrão, independente de fabricante, para manter repositórios de documentação estruturada por mais que uma década, mas não é o mais indicado para servir documentos na *WEB*. Definindo *XML* como um perfil de aplicação de *SGML*, significa que qualquer sistema completamente compatível com *SGML* será capaz de ler documentos *XML*. Porém, para usar e entender documentos de *XML* não requer um sistema capaz de compreender por completo o *SGML*. *XML* é então uma forma restringida de *SGML*.

3.5 Objetivos do desenvolvimento do *XML*

Para entender o *XML*, é importante entender por que foi criado. O *XML* foi criado para que documentos estruturados pudessem ser usados na *WEB*. As únicas alternativas viáveis, *HTML* e *SGML*, não são práticas para este propósito.

O *HTML* vem acompanhado de um conjunto semântico e não prove a criação de outros tipos de estruturas.

O *SGML* provê a criação de estruturas, mas é muito difícil sua implementação, apenas para ser usado por um navegador *WEB*. Sistemas *SGML* completos resolvem problemas grandes, complexos que justificam a despesa. Para utilizar documentos estruturados enviados pela *WEB* raramente justifica esse tipo de implementação.

Não se pode dizer que o *XML* substitui o *SGML* completamente. Como o *XML* é designado para transferir conteúdo estruturado pela *WEB*, algumas características inexistem, fazendo o *SGML* uma solução mais satisfatória para a criação e armazenamento a longo prazo de documentos complexos.

A especificação *XML*(*W3C*) cita os seguintes objetivos para *XML*:

1. Deve ser fácil usar *XML* na *Internet*. Usuários devem poder ver documentos *XML* assim como vêem documentos *HTML*.
2. *XML* deve suportar uma grande variedade de aplicações.
3. *XML* deve ser compatível com *SGML*. Muitas das pessoas envolvidas com o *XML* são de organizações que tem uma grande quantidade de material em *SGML*. Deve ser compatível com padrões existentes, enquanto resolve problemas relativamente novos.
4. Deve ser fácil escrever programas para processar documentos *XML*.
5. O número de características opcionais deve ser o mínimo possível, por motivos de compatibilidade.
6. Documentos *XML* devem ser legíveis e claros. Deve ser possível observar o documento utilizando um editor de textos comum e descobrir o significado daquele conteúdo.
7. O desenvolvimento do *XML* deve ser rápido. Padrões são notoriamente lentos, mas o *XML* é necessário imediatamente, e foi desenvolvido o mais rápido possível.
8. O *design* do *XML* deve ser formal e conciso.
9. Documentos *XML* devem ser fáceis de escrever, por exemplo através de um editor de texto.
10. Concisão nas *tags XML* tem importância mínima.

3.6 Validade

Alguns documentos são válidos e alguns não são. Há duas categorias de documentos *XML*: bem-formado e válido.

3.6.1 Documentos bem-formados

Um documento só pode ser bem-formado se obedece a sintaxe *XML*. Um documento que inclui seqüências de caracteres de marcação que não podem ser analisados gramaticalmente ou são inválidos não pode ser bem-formado.

Além disso, o documento tem que estar conforme as seguintes condições (entender algumas destas condições podem requerer experiência com *SGML*):

1. A instancia do documento deve estar em conforme com a gramática do documento *XML*.
2. Nenhum atributo deve aparecer mais de uma vez na mesma tag inicial
3. Atributo *string* não pode conter referencia a entidades externas.
4. Tags não vazias devem ser propriamente aninhadas.
5. Entidades parâmetro devem ser declaradas antes de serem utilizadas.
6. Todas entidades, exceto as seguintes devem ser declaradas: amp, lt, gt, apos e quot
7. Uma entidade binária não pode ser referenciada no fluxo do conteúdo, só pode ser utilizada num atributo declarado como *ENTITY* ou *ENTITIES*.
8. Nenhum texto ou entidade parâmetro são permitidas serem recursivas, diretamente ou indiretamente.

Por definição, se um documento não é bem-formado, não é *XML*. Isto significa que não existe um documento *XML* que não é bem-formado, e não se exige de processadores *XML* fazer qualquer coisa com tal documento.

3.6.2 Documentos Válidos

Um documento bem-formado só é válido se contiver uma declaração de tipo de documento e se o documento obedece as restrições daquela declaração. A especificação de *XML* identifica todos os critérios em detalhe.

3.7 Documento *XML*

Se você estiver familiarizado com *HTML* ou *SGML*, documentos *XML* parecerão familiares. Um documento *XML* simples é apresentado:

Exemplo. UM Documento *XML* Simples

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

<oldjoke>

<burns>Say <quote>goodnight</quote>,
Gracie.</burns>

<allen><quote>Goodnight,
Gracie.</quote></allen>

<applause/>

</oldjoke>
```

Alguns esclarecimentos:

- O documento começa com uma instrução de processamento, apesar de não ser obrigatório, identifica o documento como sendo *XML*, e indica a versão utilizada.

- Não há uma declaração de tipo de documento (*DTD*). Diferente do *SGML*, o *XML* não necessita de um *DTD*. Entretanto, o *DTD* pode ser fornecido, e alguns documentos vão precisar de um para serem entendidos sem ambigüidade.

- Elementos vazios possuem uma sintaxe modificada (`<applause/>`). Também é permitido utilizar `<applause></applause>`.

Documentos *XML* são compostos de marcações e conteúdo. Há seis tipos de marcações que podem ocorrer em um documento *XML*: elementos, referências de entidade, comentários, processando instruções, seções marcadas, e declarações de tipo de documento. As seções seguintes introduzem cada um destes conceitos de marcação.

3.7.1 Elementos

Elementos são a forma mais comum de marcação. Delimitado por '< >', a maioria dos elementos identifica a natureza do conteúdo que eles cercam. Alguns elementos podem estar vazios. Se um elemento não estiver vazio, começa com uma *tag* de inicio, `<elemento>`, e acaba com uma tag final, `</ elemento>`.

3.7.2 Atributos

Atributos são pares nome-valor que acontecem dentro de uma *tag* inicial depois do nome do elemento. Por exemplo, `<div classificam = "prefacie">` é um elemento de *div* com a classe de atributo que tem o valor *prefacie*. Em *XML*, todos os valores de atributo devem estar entre aspas.

3.7.3 Referências de entidade

Para introduzir marcação em um documento, alguns caracteres foram reservados para identificar o começo de marcação. O '<', por exemplo, identifica o começo de um elemento. Para inserir estes caracteres em seu documento como conteúdo, deve haver um modo alternativo para os representar. Em *XML*, são usadas entidades para representar estes caracteres especiais. Entidades também são usadas para textos que variam ou são repetidos com freqüência, e para incluir o conteúdo de arquivos externos.

3.7.4 Comentários

Comentários começam com <!-- e terminam com -->. Comentários podem conter qualquer dado fora a *string* --. Você pode colocar comentários em qualquer lugar entre marcação em seu documento.

Comentários não fazem parte do conteúdo textual de um documento *XML*. Um processador *XML* não precisa os passar junto para uma aplicação.

3.7.5 Instruções de Processamento

Instruções de processamento (PI) é uma forma para prover informação a uma aplicação. Como comentários, não fazem parte textual do documento *XML*, mas o processador *XML* os passa a uma aplicação.

Instruções processando têm a forma: <?*name data*?>. O nome, chamado de alvo PI, identifica o PI à aplicação. Aplicações deveriam processar só os alvos conhecidos e ignorar todos os outros PI. Qualquer dado que segue o alvo PI é opcional, é para a aplicação que reconhece o alvo. Os nomes usados em PI devem ser declarados como anotações para identificá-los formalmente.

Nomes PI que começam com *xml* são reservados para padronização *XML*.

3.7.6 Seções *CDATA*

Em um documento, uma seção de *CDATA* instrui o parser para ignorar a maioria dos caracteres de marcação.

```
<![CDATA[

*p = &q;

b = (i <= 3);

]]>
```

Entre o começo da seção, '<! [CDATA [' e o fim da seção, ']]>', todo o dados de caracteres são passados diretamente à aplicação, sem interpretação. A única *string* que não pode aparecer em uma seção CDATA é ']]>'.

3.8 Declarações de Tipo de documento (*Document Type Declarations*)

Uma porcentagem grande da especificação *XML* trata de vários tipos de declarações que são permitidas em *XML*. Se você tiver experiência com *SGML*, você reconhecerá estas declarações de *SGML DTDs* (Definições de Tipo de Documento). Se você nunca os viu antes, o significado pode não ser imediatamente óbvio.

Um das maiores forças de *XML* é que lhe permite criar seus próprios nomes de *tags*. Mas para qualquer determinada aplicação, não tem significado *tags* que aparecem em uma ordem completamente arbitrária. Considere o exemplo. Isto seria significante?

```
<gracie><quote><oldjoke>Goodnight,
<applause/>Gracie</oldjoke></quote>

<burns><gracie>Say <quote>goodnight</quote>,
</gracie>Gracie.</burns></gracie>
```

Não há significado algum no exemplo acima, porém, de um ponto de vista estritamente sintático, não há nada errado com este documento *XML*. Assim, se o documento é para ter significado, e certamente se você estiver escrevendo um *stylesheet* ou aplicação para processar isto, deve haver alguma restrição na seqüência e aninhando das *tags*. Nas declarações(*DTD*) estas restrições podem ser expressadas.

As declarações permitem um documento comunicar meta-informação ao parser sobre seu conteúdo. Meta-informação inclui a seqüência permitida e aninhando de *tags*, valores de atributo, os tipos e padrões, os nomes de arquivos externos que podem ser referenciados e se eles contêm ou não *XML*, os formatos de algum dado externo (*non-XML*) que pode ser referenciado, e as entidades que podem ser encontradas.

Há quatro tipos de declarações em *XML*: declarações de tipo de elemento, declarações de lista de atributo, declarações de entidade, e declarações de anotação.

3.9 Utilidades

Como vimos, o conteúdo *XML* pode ser processado sem uma declaração de tipo de documento. Porém, há alguns exemplos onde a declaração é necessária:

Ambientes de Autorização

A maioria dos ambientes de autorização precisa ler e processar declarações de tipo de documento para entender e obrigar os modelos de conteúdo do documento.

Valores Padrão de Atributo

Se um documento *XML* possui valores padrão de atributo, pelo menos parte da declaração deve ser processada para obter os valores padrão correto.

Manipulando Espaços em Branco

A semântica associada com espaço em branco no conteúdo de um elemento difere da semântica associada com espaço em branco num conteúdo misturado. Sem um *DTD*, não há nenhum modo para o processador distinguir entre estes casos, e todos os elementos são conteúdo misturado.

Em aplicações onde uma pessoa cria ou edita os dados (ao invés de dados que são gerados diretamente de um banco de dados, por exemplo), um *DTD* vai ser necessário para garantia da estrutura.

Capítulo 4 Seguranca *XML*

4.1 Assinatura *XML*

Assinatura *XML* é um método de associar uma chave com dados referenciados (octetos); não possui norma que especifica como as chaves são associadas a pessoas ou instituições, nem o significado dos dados que são referenciados e assinados. Logo, esta especificação é um componente importante de aplicações *XML* seguras, mas não é suficiente para resolver todos os problemas da segurança/confiança que uma aplicação necessita, particularmente com respeito a usar assinatura *XML* como base para comunicação pessoa para pessoa e acordos. Tal aplicação tem que especificar uma chave adicional, algoritmo, processo e exigências de retribuição.

O *namespace XML* [XML-ns] *URI* que deve ser usada para implementações desta data:

xmlns=http://www.w3.org/2000/09/xmldsig#

4.1.1 Visão Geral e Exemplos

Assinaturas *XML* são aplicadas em conteúdos digitais (objetos de dados). Os objetos de dados são resumidos, o valor resultante é colocado em um elemento (com outra informação) que é resumido e criptograficamente assinado. Assinaturas digitais *XML* são representadas pelo elemento *Signature* que tem a estrutura seguinte (onde "? " denota zero ou uma ocorrência; "+" denota uma ou mais ocorrências; e "* " denota zero ou mais ocorrências):

```
<Signature ID?>
    <SignedInfo>
      <CanonicalizationMethod/>
      <SignatureMethod/>
      (<Reference URI? >
        (<Transforms>)?
        <DigestMethod>
        <DigestValue>
      </Reference>)+
    </SignedInfo>
    <SignatureValue>
   (<KeyInfo>)?
   (<Object ID?>)*
  </Signature>
```

4.1.1.1 Exemplo Simples (*Signature*, *SignedInfo*, *Methods*, e *Reference*)

O exemplo seguinte é uma assinatura avulsa de um conteúdo qualquer:

```
  [s01] <Signature Id="MyFirstSignature"
xmlns="http://www.w3.org/2000/09/xmldsig#">
  [s02]   <SignedInfo>
  [s03]   <CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
  [s04]   <SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
  [s05]   <Reference
URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
```

```
   [s06]     <Transforms>
   [s07]       <Transform Algorithm="http://www.w3.org/TR/2001/REC-
xml-c14n-20010315"/>
   [s08]     </Transforms>
   [s09]     <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
   [s10]     <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
   [s11]   </Reference>
   [s12] </SignedInfo>
   [s13]   <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
   [s14]   <KeyInfo>
   [s15a]    <KeyValue>
   [s15b]      <DSAKeyValue>
   [s15c]        <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
   [s15d]      </DSAKeyValue>
   [s15e]    </KeyValue>
   [s16]   </KeyInfo>
   [s17] </Signature>
```

[s02-12] O elemento *SignedInfo* é a informação que é de fato assinada. A validação de *SignedInfo* consiste em dois processos obrigatórios: validação da assinatura sobre *SignedInfo* e validação de cada resumo de *Reference* dentro de *SignedInfo*. Note que os algoritmos usados no calculo do elemento *SignatureValue* também são incluídos na informação assinada enquanto o elemento *SignatureValue* está fora de SignedInfo.

[s03] *CanonicalizationMethod* é o algoritmo que é usado para codificar a estrutura lógica do elemento SignedInfo antes de ser resumido como parte da operação de assinatura. Note que este exemplo, e todos os exemplos nesta especificação, não estão na forma canônica.

[s04] *SignatureMethod* é o algoritmo que é usado para converter o *SignedInfo* codificado em *SignatureValue*. É uma combinação de um algoritmo de resumo e um algoritmo baseado em chave e possivelmente outros algoritmos, por exemplo *RSA-SHA1*. Os nomes de algoritmo são assinados para resistir a ataques baseados em substituir um algoritmo mais fraco. Para promover interoperabilidade entre aplicação foi especificado um conjunto de algoritmos de assinatura que devem ser implementados, entretanto o uso deles/delas é opção do criador da assinatura. O *design* também permite o uso de algoritmos arbitrários criados pelo usuário.

[s05-11] Cada elemento *Reference* inclui o método de resumo e valor de resumo resultante calculado sobre o objeto de dados identificado. Também pode incluir transformações que produziram a entrada para a operação de resumo. Um objeto de dados é assinado computando seu valor de resumo e aplicando uma assinatura em cima daquele valor. A assinatura é conferida depois por referência e validação de assinatura.

[s14-16] *KeyInfo* indica a chave a ser usada para validar a assinatura. Possíveis formas para identificação incluem certificados, nomes de chaves, algoritmos de acordo de chave e informações. *KeyInfo* é opcional por duas razões. Primeiro, o signatário pode não desejar revelar informação da chave a todos. Segundo, as informações podem ser conhecidas dentro do contexto da aplicação e não haja necessidade de ser representada explicitamente. Considerando que *KeyInfo* está fora

de *SignedInfo*, se o signatário desejar ligar a informação da chave à assinatura, um *Reference* pode facilmente identificar e incluir o *KeyInfo* como parte da assinatura.

4.1.2 Regras de Processamento

As seções abaixo descrevem as operações a serem executadas como parte da geração e validação da assinatura.

Os passos exigidos incluem a geração de elementos *Reference* e o *SignatureValue* sobre *SignedInfo*.

4.1.2.1 Geração de Referência

Para cada objeto a ser assinado:

1. Aplique o *Transforms*, como determinado pela aplicação, para o objeto de dados.

2. Calcule o valor de resumo sobre o objeto de dados resultante.

3. Crie um elemento *Reference*, inclusive o (opcional) identificador do objeto de dados, qualquer (opcional) elemento transform, o algoritmo de resumo e o *DigestValue*. (Note, é a forma canônica destas referências que são assinadas e validadas)

4.1.2.2 Geração da Assinatura

1. Crie o elemento *SignedInfo* com *SignatureMethod*, *CanonicalizationMethod* e *Reference*(s).

2. Codifique e então calcule o *SignatureValue* em cima de *SignedInfo* baseado nos algoritmos especificados em *SignedInfo*.

3. Construa o elemento *Signature* que inclui *SignedInfo*, *Object*(s) (se desejado), *KeyInfo* (se preciso), e *SignatureValue*.

4.1.3 Validação

Os passos exigidos de validação incluem a validação de *Reference*, a verificação do resumo contido em cada *Reference* em *SignedInfo*, e a validação criptográfica da assinatura calculada sobre SignedInfo.

Implementações diferentes podem produzir diferentes resumos e assinaturas quando processando o mesmo recurso por causa de uma variação na sua codificação, como por exemplo, espaços acidentais. Mas se é sempre utilizada a comparação numérica ou por octeto estes problemas são eliminados.

4.1.3.1 Validação de *Reference*

• Codifique o elemento *SignedInfo* baseado no *CanonicalizationMethod* em *SignedInfo*. Para cada *Reference* em *SignedInfo:*

• Obtenha os dados para serem resumidos.

• Resuma o objeto de dado resultante utilizando o *DigestMethod* especificado na sua especificação em *Reference.*

• Compare o valor de resumo gerado com o valor em *DigestValue* em *SignedInfo Reference*; se há qualquer mudança, a validação falha.

4.1.3.2 Validação da Assinatura

• Obtenha a informação da chave em *KeyInfo* ou de uma fonte externa.

• A Obtenha a forma canônica de *SignatureMethod* usando o *CanonicalizationMethod* e use o resultado para confirmar o *SignatureValue* sobre o elemento *SignedInfo.*

4.1.4 Elementos

Esta parte descreve a funcionalidade dos principais elementos da especificação da Assinatura *XML.*

4.1.4.1 O Elemento *Signature*

O elemento *Signature* é o elemento raiz de uma Assinatura *XML.* A implementação tem que gerar um esquema válido [XML-schema] do elemento *Signature* como especificado pelo esquema seguinte:

Schema Definition:

```
<element name="Signature" type="ds:SignatureType"/>
  <complexType name="SignatureType">
    <sequence>
      <element ref="ds:SignedInfo"/>
      <element ref="ds:SignatureValue"/>
      <element ref="ds:KeyInfo" minOccurs="0"/>
      <element ref="ds:Object" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="Id" type="ID" use="optional"/>
  </complexType>
  DTD:

  <!ELEMENT Signature (SignedInfo, SignatureValue, KeyInfo?, Object*)
>
  <!ATTLIST Signature
   xmlns   CDATA   #FIXED 'http://www.w3.org/2000/09/xmldsig#'
   Id      ID  #IMPLIED >
```

4.1.4.2 O Elemento *SignatureValue*

O elemento *SignatureValue* contém o valor da assinatura digital; que sempre usa a codificação base64 [MIME].

Schema Definition:

```
<element name="SignatureValue" type="ds:SignatureValueType"/>
 <complexType name="SignatureValueType">
   <simpleContent>
     <extension base="base64Binary">
       <attribute name="Id" type="ID" use="optional"/>
     </extension>
   </simpleContent>
 </complexType>
 DTD:

 <!ELEMENT SignatureValue (#PCDATA) >
 <!ATTLIST SignatureValue  Id  ID      #IMPLIED>
```

4.1.4.3 O Elemento *SignedInfo*

A estrutura de *SignedInfo* inclui o algoritmo de codificação, um algoritmo de assinatura, e uma ou mais referências. O elemento *SignedInfo* pode conter um atributo ID opcional que permitirá ser referenciado por outras assinaturas e objetos.

SignedInfo não inclui propriedades de resumo ou assinatura (como tempo de cálculo, número de série de dispositivo criptográfico, etc.). Se uma aplicação precisar associar propriedades com a assinatura ou resumo, pode incluir tal informação em um elemento *SignatureProperties* dentro de um elemento *Object*.

Schema Definition:

```
<element name="SignedInfo" type="ds:SignedInfoType"/>
  <complexType name="SignedInfoType">
    <sequence>
      <element ref="ds:CanonicalizationMethod"/>
      <element ref="ds:SignatureMethod"/>
      <element ref="ds:Reference" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="Id" type="ID" use="optional"/>
  </complexType>
  DTD:
  <!ELEMENT SignedInfo (CanonicalizationMethod,
   SignatureMethod,  Reference+)  >
  <!ATTLIST SignedInfo
   Id   ID      #IMPLIED
```

4.1.4.4 O Elemento *CanonicalizationMethod*

CanonicalizationMethod é um elemento que especifica o algoritmo de codificação aplicado ao elemento *SignedInfo* antes de executar cálculos de assinatura.

Schema Definition:

```
 <element name="CanonicalizationMethod"
type="ds:CanonicalizationMethodType"/>
   <complexType name="CanonicalizationMethodType" mixed="true">
     <sequence>
       <any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
       <!-- (0,unbounded) elements from (1,1) namespace -->
     </sequence>
     <attribute name="Algorithm" type="anyURI" use="required"/>
   </complexType>
   DTD:
```

```
   <!ELEMENT CanonicalizationMethod (#PCDATA %Method.ANY;)* >
   <!ATTLIST CanonicalizationMethod
    Algorithm CDATA #REQUIRED >
```

4.1.4.5 O Elemento *SignatureMethod*

SignatureMethod é um elemento que especifica o algoritmo usado para geração e validação da assinatura. Este algoritmo identifica todas as funções criptográficas envolvidas na operação de assinatura (por exemplo resumo, algoritmos de chave pública, *MACs*, etc.).

Schema Definition:

```
 <element name="SignatureMethod" type="ds:SignatureMethodType"/>
   <complexType name="SignatureMethodType" mixed="true">
     <sequence>
       <element name="HMACOutputLength" minOccurs="0"
type="ds:HMACOutputLengthType"/>
       <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
       <!-- (0,unbounded) elements from (1,1) external namespace -->
      </sequence>
    <attribute name="Algorithm" type="anyURI" use="required"/>
   </complexType>
   DTD:

   <!ELEMENT SignatureMethod (#PCDATA|HMACOutputLength %Method.ANY;)*
>
   <!ATTLIST SignatureMethod
    Algorithm CDATA #REQUIRED >
```

4.1.4.6 O Elemento *Reference*

Reference é um elemento que pode aparecer uma ou mais vezes. Ele especifica um algoritmo e valor de resumo, e opcionalmente um identificador do objeto que está sendo assinado, o tipo do objeto, e/ou uma lista de transformações a serem aplicadas antes do resumo.

Schema Definition:

```
   <element name="Reference" type="ds:ReferenceType"/>
   <complexType name="ReferenceType">
     <sequence>
       <element ref="ds:Transforms" minOccurs="0"/>
       <element ref="ds:DigestMethod"/>
       <element ref="ds:DigestValue"/>
     </sequence>
     <attribute name="Id" type="ID" use="optional"/>
     <attribute name="URI" type="anyURI" use="optional"/>
     <attribute name="Type" type="anyURI" use="optional"/>
   </complexType>
   DTD:

   <!ELEMENT Reference (Transforms?, DigestMethod, DigestValue)  >
   <!ATTLIST Reference
    Id  ID  #IMPLIED
    URI CDATA   #IMPLIED
```

```
    Type    CDATA   #IMPLIED>
```

4.1.4.7 O Elemento *Transforms*

O elemento opcional *Transforms* contém uma lista ordenada de elementos *Transform*; estes descrevem como o signatário obteve o objeto de dados que foi resumido.

Schema Definition:

```
<element name="Transforms" type="ds:TransformsType"/>
  <complexType name="TransformsType">
    <sequence>
      <element ref="ds:Transform" maxOccurs="unbounded"/>
    </sequence>
  </complexType>

  <element name="Transform" type="ds:TransformType"/>
  <complexType name="TransformType" mixed="true">
    <choice minOccurs="0" maxOccurs="unbounded">
      <any namespace="##other" processContents="lax"/>
      <!-- (1,1) elements from (0,unbounded) namespaces -->
      <element name="XPath" type="string"/>
    </choice>
    <attribute name="Algorithm" type="anyURI" use="required"/>
  </complexType>
  DTD:

  <!ELEMENT Transforms (Transform+)>

  <!ELEMENT Transform (#PCDATA|XPath %Transform.ANY;)* >
  <!ATTLIST Transform
   Algorithm    CDATA    #REQUIRED >

  <!ELEMENT XPath (#PCDATA) >
```

4.1.4.8 O Elemento *DigestMethod*

DigestMethod é um elemento que identifica o algoritmo de resumo a ser aplicado ao objeto.

Schema Definition:

```
<element name="DigestMethod" type="ds:DigestMethodType"/>
  <complexType name="DigestMethodType" mixed="true">
    <sequence>
      <any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>
    <attribute name="Algorithm" type="anyURI" use="required"/>
  </complexType>
  DTD:

  <!ELEMENT DigestMethod (#PCDATA %Method.ANY;)* >
  <!ATTLIST DigestMethod
   Algorithm       CDATA   #REQUIRED >
```

4.1.4.9 O Elemento *DigestValue*

DigestValue é um elemento que contém o valor de resumo. O resumo sempre usa a codificação base64 [MIME].

Schema Definition:

```
<element name="DigestValue" type="ds:DigestValueType"/>
<simpleType name="DigestValueType">
  <restriction base="base64Binary"/>
</simpleType>
DTD:

<!ELEMENT DigestValue  (#PCDATA)  >
<!-- base64 encoded digest value -->
```

4.1.4.10 O Elemento *KeyInfo*

KeyInfo é um elemento opcional que permite que os destinatários obtenham a chave necessária para validar a assinatura. *KeyInfo* pode conter chaves, nomes, certificados e outras informações do gerenciamento de chave pública.

Schema Definition:

```
<element name="KeyInfo" type="ds:KeyInfoType"/>
<complexType name="KeyInfoType" mixed="true">
  <choice maxOccurs="unbounded">
    <element ref="ds:KeyName"/>
    <element ref="ds:KeyValue"/>
    <element ref="ds:RetrievalMethod"/>
    <element ref="ds:X509Data"/>
    <element ref="ds:PGPData"/>
    <element ref="ds:SPKIData"/>
    <element ref="ds:MgmtData"/>
    <any processContents="lax" namespace="##other"/>
    <!-- (1,1) elements from (0,unbounded) namespaces -->
  </choice>
  <attribute name="Id" type="ID" use="optional"/>
</complexType>
DTD:

<!ELEMENT KeyInfo (#PCDATA|KeyName|KeyValue|RetrievalMethod|
            X509Data|PGPData|SPKIData|MgmtData %KeyInfo.ANY;)* >
<!ATTLIST KeyInfo
 Id  ID   #IMPLIED >
```

4.1.4.11 O Elemento *KeyName*

O elemento *KeyName* contém uma string que pode ser usada pelo signatário para comunicar um identificador da chave ao destinatário. Tipicamente, *KeyName* contém um identificador relacionado ao par de chaves usada para assinar a mensagem, mas pode conter outras informações relacionadas ao protocolo que indiretamente identificam um par de chaves.

Schema Definition:

```
<element name="KeyName" type="string"/>
DTD:

<!ELEMENT KeyName (#PCDATA) >
```

4.1.4.12 O Elemento *KeyValue*

O elemento *KeyValue* contém uma única chave pública que pode ser útil para validar a assinatura.

Schema Definition:

```
<element name="KeyValue" type="ds:KeyValueType"/>
  <complexType name="KeyValueType" mixed="true">
   <choice>
     <element ref="ds:DSAKeyValue"/>
     <element ref="ds:RSAKeyValue"/>
     <any namespace="##other" processContents="lax"/>
   </choice>
  </complexType>
  DTD:

  <!ELEMENT KeyValue (#PCDATA|DSAKeyValue|RSAKeyValue
%KeyValue.ANY;)* >
```

4.2 XML Encryption

Esta sessão especifica o processo para criptografar dados e representar o resultado no formato *XML*. Os dados podem ser dados arbitrários (inclusive um documento de *XML*), um elemento *XML*, ou o conteúdo de um elemento *XML*. O resultado da criptografia dos dados é o elemento *EncryptedData* que contém os dados cifrados.

Quando se criptografa um elemento *XML* ou conteúdo de um elemento, o elemento *EncryptedData* substitui o elemento ou o conteúdo na versão codificada do documento *XML*.

Ao codificar dados arbitrários (inclusive documentos *XML* inteiros), o elemento *EncryptedData* pode se tornar a raiz do novo documento *XML* ou pode se tornar um elemento filho de um documento *XML* escolhido pela aplicação.

O *XML namespace* [XML-NS] *URI* para esta implementação desta especificação é:

xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'

Esta especificação faz uso do *namespace* e definições de esquemas da Assinatura *XML* [XML-DSIG]:

xmlns:ds='http://www.w3.org/2000/09/xmldsig#'

4.2.1 Visão Geral e Exemplos

Esta parte provê uma visão geral e exemplos da sintaxe da Criptografia *XML*.

O elemento *EncryptedData* tem a estrutura seguinte (onde "? " denota zero ou uma ocorrência; "+" denota uma ou mais ocorrências; "* " denota zero ou mais ocorrências;

```
<EncryptedData Id? Type? MimeType? Encoding?>
  <EncryptionMethod/>?
  <ds:KeyInfo>
    <EncryptedKey>?
    <AgreementMethod>?
    <ds:KeyName>?
    <ds:RetrievalMethod>?
    <ds:*>?
  </ds:KeyInfo>?
  <CipherData>
    <CipherValue>?
    <CipherReference URI?>?
  </CipherData>
  <EncryptionProperties>?
</EncryptedData>
```

O elemento *CipherData* contêm ou referencia os dados criptografados. Se contidos, os dados criptografados são o conteúdo do elemento *CipherValue*. Se referenciados, o atributo *URI* do elemento *CipherReference* aponta o local onde estão os dados criptografados.

4.2.2 Granularidade da Criptografia

Considere as seguintes informações fictícias de pagamento que incluem informação de identificação e informação destinadas a um método de pagamento (por exemplo, cartão de crédito, transferência de dinheiro, ou cheque eletrônico):

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Esta estrutura representa que John Smith está usando um cartão de crédito com um limite de $5,000USD.

4.2.3 Criptografando um Elemento *XML*

O número de cartão de crédito de Smith é uma informação sigilosa! Se a aplicação desejar manter aquela informação confidencial, pode criptografar o elemento *CreditCard*:

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
   xmlns='http://www.w3.org/2001/04/xmlenc#'>
```

```
      <CipherData>
        <CipherValue>A23B45C56</CipherValue>
      </CipherData>
    </EncryptedData>
  </PaymentInfo>
```

Criptografando o elemento *CreditCard* inteiro da sua *tag* de inicio até a tag final, a identidade do próprio elemento é escondida. (Um intrometido não sabe se ele usou um cartão de crédito ou transferência de dinheiro.) O elemento *CipherData* contém a serialização criptografada do elemento *CreditCard*.

4.2.3.1 Criptografando um Elemento *XML* (Elemento)

Um cenário alternativo pode ser útil para agentes intermediários saberem que John usou um cartão de crédito com um limite particular, mas não o número do cartão, emissor, e data de expiração. Neste caso, o conteúdo do elemento *CreditCard* é codificado:

```
<?xml version='1.0'?>
  <PaymentInfo xmlns='http://example.org/paymentv2'>
    <Name>John Smith</Name>
    <CreditCard Limit='5,000' Currency='USD'>
      <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
       Type='http://www.w3.org/2001/04/xmlenc#Content'>
        <CipherData>
          <CipherValue>A23B45C56</CipherValue>
        </CipherData>
      </EncryptedData>
    </CreditCard>
  </PaymentInfo>
```

4.2.3.2 Criptografando de um Elemento *XML* (Dados)

Considere o cenário no qual todas as informações, excluindo o número de cartão de crédito, podem estar a vista, inclusive o fato que o elemento *Number* existe:

```
<?xml version='1.0'?>
  <PaymentInfo xmlns='http://example.org/paymentv2'>
    <Name>John Smith</Name>
    <CreditCard Limit='5,000' Currency='USD'>
      <Number>
        <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
         Type='http://www.w3.org/2001/04/xmlenc#Content'>
          <CipherData>
            <CipherValue>A23B45C56</CipherValue>
          </CipherData>
        </EncryptedData>
      </Number>
      <Issuer>Example Bank</Issuer>
      <Expiration>04/02</Expiration>
    </CreditCard>
  </PaymentInfo>
```

CreditCard e *Number* estão a vista, mas o conteúdo de *Number* é codificado.

4.2.3.3 Criptografando Dados Arbitrários e Documentos *XML*

Se o cenário da aplicação requerer que toda informação seja codificada, o documento inteiro é codificado como uma seqüência de octetos. Isto se aplica a dados arbitrários inclusive documentos *XML*.

```
<?xml version='1.0'?>
<EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
 MimeType='text/xml'>
  <CipherData>
    <CipherValue>A23B45C56</CipherValue>
  </CipherData>
</EncryptedData>
```

4.2.4 Sintaxe da Criptografia

Esta seção provê uma descrição detalhada da sintaxe e características da Criptografia *XML*.

Schema Definition:

```
<?xml version="1.0" encoding="utf-8"?>
 <!DOCTYPE schema  PUBLIC "-//W3C//DTD XMLSchema 200102//EN"
  "http://www.w3.org/2001/XMLSchema.dtd"
  [
    <!ATTLIST schema
      xmlns:xenc CDATA #FIXED 'http://www.w3.org/2001/04/xmlenc#'
      xmlns:ds CDATA #FIXED 'http://www.w3.org/2000/09/xmldsig#'>
    <!ENTITY xenc 'http://www.w3.org/2001/04/xmlenc#'>
    <!ENTITY % p ''>
    <!ENTITY % s ''>
   ]>

 <schema xmlns='http://www.w3.org/2001/XMLSchema' version='1.0'
         xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
         xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'
         targetNamespace='http://www.w3.org/2001/04/xmlenc#'
         elementFormDefault='qualified'>

   <import namespace='http://www.w3.org/2000/09/xmldsig#'
           schemaLocation='http://www.w3.org/TR/2002/REC-xmldsig-
core-20020212/xmldsig-core-schema.xsd'/>
```

4.2.4.1 O Elemento *EncryptedType*

Schema Definition:

```
<complexType name='EncryptedType' abstract='true'>
   <sequence>
     <element name='EncryptionMethod'
type='xenc:EncryptionMethodType'
              minOccurs='0'/>
     <element ref='ds:KeyInfo' minOccurs='0'/>
     <element ref='xenc:CipherData'/>
     <element ref='xenc:EncryptionProperties' minOccurs='0'/>
```

```
  </sequence>
  <attribute name='Id' type='ID' use='optional'/>
  <attribute name='Type' type='anyURI' use='optional'/>
  <attribute name='MimeType' type='string' use='optional'/>
  <attribute name='Encoding' type='anyURI' use='optional'/>
 </complexType>
```

EncryptionMethod é um elemento opcional que descreve o algoritmo de criptografia aplicado aos dados cifrados. Se o elemento estiver ausente, o algoritmo de criptografia deve ser conhecido pelo destinatário ou a decriptografia falhará.

ds:KeyInfo é um elemento opcional e recebe a informação sobre a chave usada para criptografar os dados. Seções subseqüentes desta especificação definem elementos novos que podem aparecer como filhos de *ds:KeyInfo*.

CipherData é um elemento obrigatório que contém o *CipherValue* ou *CipherReference* com os dados criptografados.

EncryptionProperties pode conter informação adicional relativo à geração do *EncryptedType* (por exemplo, date/hora).

Id é um atributo opcional que provê um método de nomear um id ao elemento dentro do contexto do documento.

Type é um atributo opcional que identifica o tipo de informação do conteúdo criptografado. Se o elemento *EncryptedData* contém dados do tipo 'element' ou elemento 'content', e substitui os dados em um contexto no documento *XML*, é recomendado que o atributo *Type* seja fornecido. Sem esta informação, não será possível restabelecer automaticamente o documento *XML* para sua forma original.

MimeType é um atributo opcional que descreve o tipo dos dados que foram criptografados. O valor deste atributo é uma seqüência de caracteres com valores definidos por [MIMIQUE]. Por exemplo, se o dado criptografado é um *PNG* codificado em base64, o '*Encondig*' de transferência pode ser especificada como 'http://www.w3.org/2000/09/xmldsig#base64' e o *MimeType* como 'image/png'.

4.2.4.2 O Elemento *EncryptionMethod*

EncryptionMethod é um elemento opcional que descreve o algoritmo de criptografia aplicado aos dados cifrados. Se o elemento estiver ausente, o algoritmo de criptografia deve ser conhecido pelo recipiente ou a decriptografia falhará.

Schema Definition:

```
<complexType name='EncryptionMethodType' mixed='true'>
  <sequence>
    <element name='KeySize' minOccurs='0' type='xenc:KeySizeType'/>
    <element name='OAEPparams' minOccurs='0' type='base64Binary'/>
    <any namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
  </sequence>
  <attribute name='Algorithm' type='anyURI' use='required'/>
</complexType>
```

4.2.4.3 O Elemento *CipherData*

CipherData é um elemento obrigatório que provê os dados criptografados. Ou contêm uma seqüência de octetos criptografados como texto codificado em base64 do elemento *CipherValue*, ou provê uma referência a um local externo que contém a sequência de octetos criptografados via o elemento *CipherReference*.

Schema Definition:

```
<element name='CipherData' type='xenc:CipherDataType'/>
<complexType name='CipherDataType'>
   <choice>
     <element name='CipherValue' type='base64Binary'/>
     <element ref='xenc:CipherReference'/>
   </choice>
 </complexType>
```

4.2.4.4 O Elemento *EncryptedData*

O elemento *EncryptedData* é o elemento chave na sintaxe. Não só seu filho, *CipherData*, contêm os dados criptografados, mas também é o elemento que substitui o elemento criptografado, ou serve como a raiz do novo documento.

Schema Definition:

```
<element name='EncryptedData' type='xenc:EncryptedDataType'/>
<complexType name='EncryptedDataType'>
  <complexContent>
   <extension base='xenc:EncryptedType'>
   </extension>
  </complexContent>
</complexType>
```

4.2.4.5 Extensões para o Elemento *ds:KeyInfo*

Há três modos para que o material da chave necessário para decriptografar *CipherData* possa ser fornecido:

1. O elemento *EncryptedData* ou *EnacryptedKey* especificam a chave associado via um filho de *ds:KeyInfo*.

2. Um elemento destacado (não dentro de *ds:KeyInfo*) de *EncryptedKey* pode especificar o *EncryptedData* ou *EncryptedKey*, onde a chave para decriptografia será aplicada via *DataReference* ou *KeyReference*.

3. A chave pode ser determinada pelo recipiente através de contexto de aplicação e assim não precisa ser mencionado explicitamente no *XML* transmitido.

4.2.4.6 O Elemento EncryptedKey

Identificador: Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"

O elemento *EncryptedKey* é usado para transportar chaves criptográficas do remetente para um destinatário. Pode ser usado como documento *XML*, por ser colocado dentro de um documento de aplicação, ou aparecer dentro de um elemento *EncryptedData* como filho um elemento de *ds:KeyInfo*.

O valor da chave é sempre codificado para o destinatário. Quando *EncryptedKey* é decifrado, os octetos resultantes são disponibilizados ao algoritmo de *EncryptionMethod* sem qualquer processo adicional.

Schema Definition:

```
<element name='EncryptedKey' type='xenc:EncryptedKeyType'/>
  <complexType name='EncryptedKeyType'>
    <complexContent>
      <extension base='xenc:EncryptedType'>
        <sequence>
          <element ref='xenc:ReferenceList' minOccurs='0'/>
          <element name='CarriedKeyName' type='string'
minOccurs='0'/>
        </sequence>
        <attribute name='Recipient' type='string' use='optional'/>
      </extension>
    </complexContent>
  </complexType>
```

4.2.5 Regras de Processamento

Esta seção descreve as operações a serem executadas como parte do processo de criptografia e decriptografia implementadas nesta especificação.

4.2.5.1 Criptografia

Para cada item de dados a ser criptografado como *EncryptedData* ou *EncryptedKey* (elementos derivados de *EncryptedType*), o codificador deve:

1. Selecionar o algoritmo (e parâmetros) a serem usados para criptografar os dados.

2. Obter e (opcionalmente) representar a chave
    2.1. Se a chave deve ser identificada, construa o *ds:KeyInfo*
    2.2. Se a própria chave será criptografada, construa um elemento *EncryptedKey* recursivamente aplicando este processo de criptografia. O resultado pode ser então um filho de *ds:KeyInfo*, ou pode existir em outro lugar e pode ser identificado no próximo passo.

3. Criptografando os dados
    3.1. Se o dado é um 'elemento' [*XML*] ou conteúdo de um elemento [*XML*], obtenha os octetos, serializando os dados em UTF-8. A serialização deve ser feita pelo crifrador. Se o cifrador não serializar, então a aplicação tem que executar a serialização.
    3.2. Se o dado for de qualquer outro tipo que não octetos, a aplicação tem que serializar.

3.3. Criptografe os octetos utilizando o algoritmo e chave dos passos 1 e 2.

3.4. A menos que o decifrador saiba o tipo dos dados criptografados implicitamente, o cifrador deve prover o tipo para representação.

4. Construa o estrutura *EncryptedType* (*EncryptedData* ou *EncryptedKey*):
Uma estrutura *EncryptedType* representa toda informação previamente discutida inclusive o tipo dos dados codificados, algoritmo de criptografia, parâmetros, chave, tipo dos dados codificados, etc.

4.1. Se a seqüência de octetos criptografada obtida no passo 3 será armazenado no elemento *CipherData* dentro do *EncryptedType*, então a seqüência de octetos será codificada em base64 e inserida como o conteúdo de um elemento *CipherValue*.

4.2. Se a seqüência de octetos será armazenada externamente à estrutura *EncryptedType*, então guarde ou devolva a seqüência de octetos criptografada, e represente a *URI* e transformações exigidas para o decifrador recuperar a seqüência de octetos criptografada dentro de um elemento *CipherReference*.

5. Processando *EncryptedData*

5.1. Se o tipo dos dados criptografados é um 'elemento' ou conteúdo de um elemento, então o cifrador deve poder devolver o elemento *EncryptedData* à aplicação. A aplicação pode usar isto como o elemento topo em um documento *XML* novo ou pode inserir em outro documento *XML* que pode requerer uma re-codificação. O cifrador também deve poder substituir o 'elemento' ou conteúdo do elemento não criptografado com o elemento *EncryptedData*.
O cifrador remove o elemento ou conteúdo identificado e insere o elemento *EncryptedData* no seu lugar.

5.2. Se o tipo dos dados criptografado não é um 'elemento' ou o conteúdo de um elemento, então o cifrador sempre têm que devolver o elemento *EncryptedData* à aplicação. A aplicação pode usar isto como o elemento topo em um documento *XML* novo ou pode inserir isto em outro documento *XML* que pode requerer uma re-codificação.

4.2.5.2 Decriptografia

Para cada *EncryptedType*, (*EncryptedData* ou *EncryptedKey*), para serem decifrados, o decifrador deve:

1. Processar o elemento para determinar o algoritmo, parâmetros e elemento *ds:KeyInfo* a ser usado. Se alguma informação é omitida, a aplicação tem que provê-la.

2. Localizar a chave de codificação dos dados de acordo com o elemento *ds:KeyInfo* que pode conter um ou mais elementos filho. Estes filhos não têm nenhuma ordem de processo incluída. Se a chave de codificação dos dados esta codificada, localize a chave correspondente para decifrá-la.

3. Decriptografar os dados contidos no elemento *CipherData*.

3.1. Se um elemento filho de *CipherValue* estiver presente, então o valor do texto associado é decodificado em base64 para se obter a seqüência de octetos criptografada.

3.2. Se um filho do elemento *CipherReference* está presente, a *URI* e transform são usados para obter a seqüência de octetos criptografada.

3.3. A seqüência de octetos é decriptografada usando o algoritmo, parâmetros e chave determinados nos passos 1 e 2.

4. Processo decriptografa dados do tipo 'elemento' ou o conteúdo de um elemento.

4.1. A seqüência de octetos texto obtida no passo 3 é interpretada como caracteres codificados em UTF-8.

4.2. O decifrador deve poder devolver o valor de '*Type*' e os caracteres de dados *XML* codificados em UTF-8. O decifrador não precisa executar a validação do *XML* serializado.

4.3. O decifrador deve poder substituir o elemento *EncryptedData* pelo elemento ou pelo conteúdo do elemento decriptografado, representados em caracteres codificados em UTF-8.

5. Processo de decriptografar dados se o '*Type*' não é especifidado ou se não é um 'elemento' ou o conteúdo de um elemento.

5.1. A seqüência de octetos obtida no Passo 3 deve ser devolvida à aplicação para processo futuro junto com os valores dos atributos '*Type*', *MimeType*, e '*Encoding*' quando especificado.

4.3 *XML Key Management (XKMS)*

O *XKMS* (*XML Key Management*) substitui muitos dos protocolos e formatos de dados do *PKI*. Através dele, o cliente e o servidor compartilham um serviço *XKMS* para se validarem e processarem requisições entre si. Ele também pode ser implementado cliente-para-cliente, servidor-para-cliente e servidor-para-servidor.

Diferentemente do serviço *XKMS*, o *PKI* requer que cada usuário e aplicação verifique a identidade das entidades com quem estão se comunicando e se certifiquem que a identidade dessas entidades são válidas (não foram revogadas) e apropriadas para a transação. Infelizmente a infra-estrutura necessária para suportar tais requisitos torna o desenvolvimento de aplicações seguras muito demorado e complicado.

Com o *PKI*, todas as decisões são feitas na aplicação. Isso requer complicadas bibliotecas de programação e configurações.

Já com o *XKMS*, as decisões são feitas em um servidor comum. A única informação necessária à um cliente *XKMS* é a *URL* do servidor e o certificado que o servidor estará usando para assinar as suas respostas.

4.3.1 Trocas do Protocolo

As trocas do protocolo *XKMS* consistem de uma seqüência de um ou dois pares de resposta.

As mensagens compartilham um formato comum e podem ser utilizadas em uma variedade de protocolos. É recomendado que os implementadores do *XKMS* suportem *SOAP* sobre *HTTP* para propósitos de interoperabilidade.

No *XKMS* nenhuma operação é idempotente, ou seja, todas as requisições podem causar uma mudança de estado.

O protocolo *XKMS* consiste de pares de requisições e respostas. Cada mensagem de resposta contem um código *MajorResult* que determina se a resposta é final ou se requer um processamento adicional. O protocolo é assim especificado:

Final = { *Success*, *VersionMismatch*, *Sender*, *Receiver* }

Request -> Result.Final
|
Request -> Result.Pending->PendingNotification->Request->Result.Final
|
Request -> Result.Represent->Request->Result.Final

As sessões a seguir descrevem o protocolo de mensagem e as etapas de processamento efetuadas por ambas as partes em cada mensagem.

4.3.2.1 Todas as Mensagens

As seguintes etapas são processadas independente de a mensagem ser de requisição ou de resposta:

Geração

ID é setado para um valor único gerado aleatoriamente.

Service é setado para o valor da *URI* para onde a requisição *XKMS* é direcionada.

Assinatura de autenticação é gerada (se necessária).

Processamento

O valor de *Service* é verificado.

A assinatura de autenticação é verificada (se necessária).

Exemplo

```
<?xml version="1.0" encoding="utf-8"?>
<MessageAbstractType Id="1noOYHt5Lx7xUuizWZLOMw=="
      Service="http://test.xmltrustcenter.org/XKMS"
       xmlns="http://www.w3.org/2002/03/xkms#" />
```

4.3.2.2 Tipos de Requisição

A especificação do *XKMS* define três tipos de requisição:

Requisição *X-KISS*

Uma requisição de localização ou validação.

Requisição *X-KRSS*

Uma requisição de registro, alteração, revogação ou de recuperação.

Requisição Composta

Uma requisição composta consiste de um conjunto de uma ou mais requisições *X-KISS* ou *X-KRSS*.

O protocolo *XKMS* suporta diversas opções como processamento assíncrono, requisição de duas fases e requisição composta. O cliente especifica que opções do protocolo suporta em relação a uma requisição específica através do elemento *ResponseMechanism* na própria requisição.

4.3.2.3 Respostas

Todas as respostas contêm um código de resultado compreendido de um componente *major* e *minor*. Se um serviço aplica uma opção de processamento de protocolo o cliente é informado através do valor do elemento *MajorResult* da resposta.

4.3.2.4 Processamento Síncrono e Assíncrono

O protocolo *XKMS* suporta dois modos de processo: síncrono e assíncrono.

- No processamento síncrono o serviço responde uma requisição de modo que ele não tenha de enviar mais nenhuma resposta para aquela requisição.
- No processamento assíncrono o serviço não irá completar a requisição imediatamente enviando uma notificação de que outras mensagens de reposta serão encaminhadas.

O cliente pode avisar o serviço que irá aceitar o processamento assíncrono de uma requisição especificando o elemento *ResponseMechanism* com o valor *Pending*. Um serviço *XKMS* que recebe uma requisição com o valor *Pending* para o elemento *ResponseMechanism p*ode responder tanto de modo síncrono como assíncrono. Se o serviço decidir responder de modo assíncrono, ele avisa o cliente especificando o elemento *MajorResult* com o valor *Pending*.

Um serviço *XKMS* não deve retornar o elemento *MajorResult* com o valor *Pending* a não ser que o elemento *ResponseMechanism* tenha sido especificado com o valor *Pending* na requisição correspondente. Se o serviço receber uma requisição que não possa ser processada de modo síncrono e que o valor *Pending* não tenha sido especificado para a variável *ResponseMechanism* é retornado o valor *Receiver* no elemento *MajorResult* e *NotSynchronous* para o elemento *MinorResult*.

O processamento assíncrono pode ser utilizado para permitir que o administrador possa intervir durante o processamento da requisição. Por exemplo, pode ser necessário que o administrador verifique e aprove todas as requisições *XKRSS* de registro antes que sejam processadas.

4.3.2.4.1 Requisição / Resposta Síncrona

O processamento de uma requisição e resposta síncrona é feita da seguinte maneira:

Geração da mensagem de requisição (requisitor)

Nonce e *OriginalRequestId* não estão presentes.

ResponseMechanism pode ser especificado com o valor *Represent* e/ou *Pending*.

Processamento da mensagem de requisição (serviço)

Verifica se a requisição está de acordo com a política de autorização do serviço.

Processa a requisição até ser completa.

Geração da mensagem de resposta (serviço)

RequesId é setado com o valor *Id* da mensagem de requisição.

Nonce não está presente.

MajorResult é setado para um valor de resultado *Final*.

Processamento da mensagem de resposta (requisitor)

O valor de *RequestId* é verificado.

Exemplo

Requisição

```
<?xml version="1.0" encoding="utf-8"?>
<LocateRequest Id="I6d995b8d05a9a2ce0573d29e32ab9441"
      Service="http://test.xmltrustcenter.org/XKMS"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <QueryKeyBinding />
</LocateRequest>
```

Resposta

```
<?xml version="1.0" encoding="utf-8"?>
<LocateResult Id="I089b18dc1a520b26e2e6689dd3a5a820"
      Service="http://test.xmltrustcenter.org/XKMS" ResultMajor="Success"
      RequestId="I6d995b8d05a9a2ce0573d29e32ab9441"
      xmlns="http://www.w3.org/2002/03/xkms#" />
```

4.3.2.5 Processamento Assíncrono

O processamento assíncrono consiste na seqüência de dois pares de requisição/resposta, uma requisição inicial que especifica os valores requeridos e uma requisição de pendência que obtém o resultado da operação.

4.3.2.5.1 Requisição Inicial

A requisição inicial é processada da seguinte maneira:

Geração da mensagem de requisição inicial (requisitor)

Nonce e *OriginalRequestId* não estão presentes.

ResponseMechanism deve ser especificado com o valor *Pending*.

Processamento da mensagem de requisição inicial (serviço)

Agenda o processamento da requisição assíncrona.

Geração da mensagem de resposta inicial (serivço)

RequestId é setado para o valor *Id* da mensagem de requisição inicial.

Nonce não está presente.

MajorResult é setado para *Pending*.

Processamento da mensagem de resposta inicial (requisitor)

Registra a requisição como em pendência e aguarda notificação.

4.3.2.5.2 Requisição de Status

O cliente pode verificar o status da operação assíncrona:

Geração da mensagem de requisição de status (requisitor)

O elemento de requisição é *StatusRequest*.

OriginalRequestId é setado para o valor de *Id* da mensagem de requisição inicial.

ResponseId é setado para o valor de *Id* da mensagem inicial de resposta.

Processamento da mensagem de requisição de status (serviço)

Identifica a requisição pendente usando *OriginalRequestId* e *ResponseId*.

Geração da mensagem de resposta de status (serviço)

Request Id é setado para o valor de *Id* na mensagem de requisição de status.
Nonce não está presente.

Processamento da mensagem de resposta de status (requisitor)

Para mensagens simples, o atibuto *ResultMajor* indica o status da operação.
Para mensagens compostas os atributos *Success*, *Failure* e *Pending* indicam o número de requisições internas que possuem os respectivos status.

4.3.2.5.3 Requisição de pendência

Quando notificado, o cliente emite uma mensagem *PendingRequest* requisitando o retorno do resultado do processamento da requisição inicial.

Geração da mensagem de requisição de pendência (requisitor)

O elemento de requisição é *PendingRequest*.
OriginalRequestId é setado para o valor *Id* da mensagem de requisição inicial.
ResponseId é setado para o valor *Id* da mensagem de resposta inicial.

Processamento da mensagem de requisição de pendência (serviço)

Associa à requisição de pendência a resposta pendente.

Geração da mensagem de resposta de pendência (serviço)

RequestId é setado para o valor *Id* da mensagem de requisição de pendência.
Nonce não está presente.
ResponseId é setado para um valor único gerado aleatoriamente.

Processamento da mensagem de resposta de pendência (requisitor)

Se *MajorResult* está setado com um valor não final, considera como falha.

O cliente pode requisitar o retorno do resultado antes do processamento ser completo. Nesse caso o serviço responde a mensagem de requisição de pendência com o valor *Pending* de *MajorResult*.

Exemplo

Requisição

```
<?xml version="1.0" encoding="utf-8"?>
<LocateRequest Id="I6227979ae4073f2b3b145db7a488ce16"
      Service="http://test.xmltrustcenter.org/XKMS"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <ResponseMechanism>Pending</ResponseMechanism>
  <QueryKeyBinding />
</LocateRequest>
```

Resposta

```
<?xml version="1.0" encoding="utf-8"?>
<LocateResult Id="I98366e407a2a78dff79687dbdb4d974c"
      Service="http://test.xmltrustcenter.org/XKMS" ResultMajor="Pending"
      RequestId="I6227979ae4073f2b3b145db7a488ce16"
      xmlns="http://www.w3.org/2002/03/xkms#" />
```

Notificação

```
<?xml version="1.0" encoding="utf-8"?>
<Result xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      xmlns="http://www.w3.org/2002/03/xkms#"/>
```

Requisição de Pendência

```
<?xml version="1.0" encoding="utf-8"?>
<PendingRequest Id="I6045ff8b2eb204edb538be1fa22e340a"
      Service="http://test.xmltrustcenter.org/XKMS"
      OriginalRequestId="I6227979ae4073f2b3b145db7a488ce16"
      ResponseId="I98366e407a2a78dff79687dbdb4d974c"
      xmlns="http://www.w3.org/2002/03/xkms#" />
```

Resposta

```
<?xml version="1.0" encoding="utf-8"?>
<LocateResult Id="I4da52fc78e0391a11257d64926cd184c"
      Service="http://test.xmltrustcenter.org/XKMS" ResultMajor="Success"
      RequestId="I6045ff8b2eb204edb538be1fa22e340a"
      xmlns="http://www.w3.org/2002/03/xkms#" />
```

4.3.2.6 Protocolo de Requisição de Duas Fases

As requisições *XKMS* podem utilizar o protocolo de requisição de duas fases para se proteger de um ataque de negação de serviço. O protocolo de requisição de duas fases permite que o serviço execute uma autenticação leve da fonte da requisição *XKMS*, determinando se o cliente consegue ler as mensagens enviadas para o suposto endereço da fonte. Apesar de esse mecanismo fornecer uma forma fraca de autenticação, ele previne um ataque de negação de serviço ao forçar o serviço a executar uma autenticação de recurso como a verificação de uma assinatura digital.

O protocolo de duas fases consiste de duas etapas:

Na primeira etapa o requisitor apresenta a requisição e o serviço responde com o valor *Represent* para o elemento *MajorResult* e apresenta um *Nonce*.

Na segunda etapa o requisitor apresenta a requisição inicial juntamente com o *Nonce*.

O cliente pode avisar o serviço que suporta o protocolo de requisição de duas fases especificando o elemento *ResponseMechanism* com o valor *Represent*. O serviço *XKMS* avisa o cliente de que é necessário o uso do protocolo de requisição de duas fases especificando com o código *Represent* o elemento *MajorResult*.

Um serviço *XKMS* não deve retornar o código *Represent* para o elemento *MajorResult* a não ser que o elemento *ResponseMechanism* tenha sido especificado com o valor *Represent* na requisição correspondente. Se o serviço *XKMS* requer o uso do protocolo de duas fases e o elemento *ResponseMechanism* não foi especificado com o valor *Represent* na mensagem de requisição correspondente, é retornado o valor *Sender* para *MajorResult* e *RepresentRequired* para o elemento *MinorResult*.

O protocolo de requisição de duas fases se assemelha ao processamento de requisição assíncrono. Ambos os mecanismos introduzem uma rodada extra de troca de mensagens, porem para propósitos diferentes. O propósito do processamento assíncrono é o de permitir que seja introduzido um atraso entre a requisição inicial e o retorno do resultado. Já no caso do protocolo de duas fases não há atraso entre a primeira requisição e a primeira resposta ou entre a primeira resposta e a segunda requisição. O propósito do protocolo de duas fases é o de permitir que o serviço se proteja contra um ataque de negação de serviço ao executar uma leve autenticação sobre a fonte da requisição.

4.3.2.6.1 Passos do processamento

Primeira Etapa:

Geração da mensagem de requisação da primeira etapa (requisitor)

ResponseMechanism deve ser especificado com o valor *Represent*.

Processamento da Mensagem de requisição da primeira etapa (serviço)

Serviço opta por utilizar o processamento de duas fases.

Requisição não é processada.

Geração da mensagem de resposta da primeira etapa (serviço)

RequestId é setado para o valor *Id* da mensagem de requisição da primeira etapa.

O valor de *Nonce* é setado de acordo com os requisitos de proteção do serviço.

MajorResult é setado para o valor *Represent*.

Processamento da mensagem de resposta da primeira etapa (requisitor)

Segue para a segunda etapa.

Segunda Etapa:

Geração da mensagem de requisição da segunda etapa (requisitor)

OriginalRequestId é setado para o valor de *id* da mensagem de requisição da primeira etapa.

O valor de *Nonce* é setado para o valor de *Nonce* da mensagem de resposta da primeira etapa.

Processamento da mensagem de requisição da segunda etapa (serviço)

Valor de *Nonce* é verificado.

Verifica se a requisição está de acordo com a politica de autorização.

Processa a requisição.

Geração da mensagem de resposta da segunda etapa (serviço)

RequestId é setado para o valor de *Id* da mensagem de requisição da segunda etapa.

Nonce não está presente.

MajorResult é setado para o valor do resultado final.

Processamento da mensagem de resposta da segunda etapa (requisitor)

Se *MajorResult* está setado para um valor que não seja final, considera-se como falha.

Exemplo

Requisição 1

```
<?xml version="1.0" encoding="utf-8"?>
<LocateRequest Id="Ia1d6ca7a067fdd545f1a1396d2f26779"
      Service="http://test.xmltrustcenter.org/XKMS"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <ResponseMechanism>Represent</ResponseMechanism>
  <QueryKeyBinding />
</LocateRequest>
```

Resposta 1

```
<?xml version="1.0" encoding="utf-8"?>
<LocateResult Id="Idbc77142059a3a51c9eccd2425d77757"
      Service="http://test.xmltrustcenter.org/XKMS"
```

```
      Nonce="Rj2BoUZM7PisPX2ytSAAWA==" ResultMajor="Represent"
      RequestId="Ia1d6ca7a067fdd545f1a1396d2f26779"
      xmlns="http://www.w3.org/2002/03/xkms#" />
```

Requisição 2

```
<?xml version="1.0" encoding="utf-8"?>
<LocateRequest Id="I47804adaec32e34afeecdb51f3e0f765"
      Service="http://test.xmltrustcenter.org/XKMS"
      Nonce="Rj2BoUZM7PisPX2ytSAAWA=="
      OriginalRequestId="Ia1d6ca7a067fdd545f1a1396d2f26779"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <QueryKeyBinding />
</LocateRequest>
```

Resposta 2

```
<?xml version="1.0" encoding="utf-8"?>
<LocateResult Id="I3b0111d2232507a56444c1bc85409a94"
      Service="http://test.xmltrustcenter.org/XKMS" ResultMajor="Success"
      RequestId="I47804adaec32e34afeecdb51f3e0f765"
       xmlns="http://www.w3.org/2002/03/xkms#" />
```

4.3.2.7 Protocolo de duas fases com processamento assíncrono

O protocolo de duas fases pode ser combinado com o processamento assíncrono. Nesse caso a operação irá consistir de 3 rodadas:

- Requisição Inicial / Etapa 1
- Requisição Inicial / Etapa 2
- Resposta Pendente

O processamento das mensagens é executada como descrito acima com as seguintes exceções:

- *OriginalRequestId* é setado para o valor de *Id* da Requisição Inicial / Etapa 1 em ambas as requisições subseqüentes.
- O valor de *Nonce* é setado para o valor de *Nonce* da Etapa 1 em ambas as requisições subseqüentes.

4.3.2.8 Requisições e Respostas Compostas

Um serviço *XKMS* pode suportar o processamento de requisições compostas. Uma requisição composta permite que se faça múltiplas requisições *XKMS* ao mesmo tempo. Ela consiste de uma requisição externa e uma ou mais requisições internas. Não há nenhuma ordenação implícita nas requisições internas. A semântica de se fazer um conjunto de requisições como uma requisição composta é exatamente igual de como seria se cada requisição individual do conjunto tivesse sido feita separadamente e simultaneamente.

A resposta para uma requisição composta é uma resposta composta. Uma resposta composta consiste de uma resposta externa e zero ou mais respostas internas. Se o valor de *ResultMajor* da resposta externa é igual a *Success*, a resposta composta deve conter uma resposta interna correspondente a cada requisição interna da requisição composta. Se o valor de *ResultMajor* da resposta externa não for igual a *Success*, a resposta não deve conter nenhuma resposta interna. Se uma resposta composta possui o valor *Success* para *ResultMajor* da resposta externa e não contem uma resposta correspondente a uma requisição interna, o valor de *ResultMajor* é assumido como *failure* para essa requisição.

Um serviço *XMKS* pode suportar o uso do protocolo de duas fases na requisição externa de uma resposta composta. O protocolo de duas fases não deve ser usado em uma resposta interna. Se uma requisição interna especificar o valor *Represent* para *ResponseMechanism*, o valor deve ser ignorado.

Um service *XKMS* pode suportar o uso de processamento assíncrono em conjunto com uma requisição composta. O processamento assíncrono pode ser executado em uma requisição composta como um todo, em uma requisição interna ou em ambas.

Se um processamento assíncrono for executado em uma requisição composta como um todo, a requisição externa deve conter o valor *Pending* para *ResponseMechanism*. Se o serviço decidir retornar uma resposta assíncrona, uma resposta composta é retornada com o valor *Pending* para *ResultMajor*. Após receber a notificação, o cliente envia uma mensagem do tipo *PendingRequest* como requisição externa na qual o serviço retorna uma resposta composta com respostas internas correspondentes as requisições internas originais ou um relatório de erro.

Se um processamento assíncrono é executado nas requisições internas individuais, para cada requisição interna na qual deverá ser aceita uma resposta assíncrona, deve ser especificado o valor *Pending* para *ResponseMechanism*. Se o serviço decidir retornar uma resposta assíncrona para uma requisição interna, é retornada uma resposta composta com o código *Success* para o *ResultMajor* externo e o código *Pending* para o *ResultMajor* interno para as requisições nas quais uma resposta assíncrona deve ser enviada. O serviço pode retornar respostas síncronas e assíncronas em uma única resposta composta.

Já que a semântica de uma requisição composta é exatamente a mesma, como se cada requisição interna fosse feita separadamente, o cliente pode enviar requisições de pendência separadamente para obter os resultados de requisições internas de uma requisição composta anterior. Alternativamente, o cliente pode enviar uma requisição composta contendo múltiplas requisições internas de pendência correspondendo a requisições que foram feitas independentemente originalmente.

Exemplo

Requisição 1

```
<?xml version="1.0" encoding="utf-8"?>
<CompoundRequest Id="I264f5da49b1ff367d4e7aef1f7a1df1a"
      Service="http://test.xmltrustcenter.org/XKMS"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <LocateRequest Id="I8c26be5f1b4dd228b43fb6eaee285faa"
       Service="http://test.xmltrustcenter.org/XKMS">
    <RespondWith>KeyValue</RespondWith>
    <QueryKeyBinding>
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <X509Data>

<X509Certificate>MIICEDCCAX2gAwIBAgIQimXeUAxYJbJMady9vV1bLjAJBgUrDg
                MCHQUAMBIxEDAOBgNVBAMTB1Rl
c3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMDUwODE1MDY1OTU5WjArMSkwJwYDVQQDEyBBbGljZSBB
YXJkdmFyayBPPUFsaWNlIENvcnAgQz1VUzCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA0nIs
mR+aVW2egl5MIfOKy4HuMKkk9AZ/IQuDLVPlhzOfgngjVQCjr8uvmnqtNu8HBupui8LgGthO6U9D
0CNT5mbmhIAErRADUMIAFsi7LzBarUvNWTqYNEJmcHsAUZdrdcDrkNnG7SzbuJx+GDNiHKVDQggP
BLc1XagW20RMvokCAwEAAaNWMFQwDQYDVR0KBAYwBAMCBkAwQwYDVR0BBDwwOoAQAaVOkaVLLKoF
mLN37pC8uqEUMBIxEDAOBgNVBAMTB1Rlc3QgQ0GCEC4MndUXjPG1TZxVKg+HutAwCQYFKw4DAh0F
AAOBgQABU91ka7IlkXCfv4Zh2Ohwgg2yObtY3+6C/BTFGrOEBJDy+DoxJ/NuBF18w3rrrR18xE6j
NKYLCQb8zUGk4QOG5Y+HT/QTTFvWkiOLXcpTuhnOhXatr42FoYpDkjx2QWK+J5Q2l/Rgjgc/0ZV8
U/kD8UuRkXp4AZh7QsiX8AcO0w==</X509Certificate>
        </X509Data>
      </KeyInfo>
      <KeyUsage>Signature</KeyUsage>
```

```
    </QueryKeyBinding>
  </LocateRequest>
  <LocateRequest Id="If8e63d729384ad35498e7b65b3dc785e"
        Service="http://test.xmltrustcenter.org/XKMS">
    <RespondWith>KeyName</RespondWith>
    <RespondWith>KeyValue</RespondWith>
    <RespondWith>X509Cert</RespondWith>
    <RespondWith>X509Chain</RespondWith>
    <RespondWith>PGPWeb</RespondWith>
    <RespondWith>PGP</RespondWith>
    <QueryKeyBinding>
      <KeyUsage>Encryption</KeyUsage>
      <UseKeyWith Application="urn:ietf:rfc:2440"
            Identifier="bob@bobcorp.test" />
      <UseKeyWith Application="urn:ietf:rfc:2633"
            Identifier="bob@bobcorp.test" />
    </QueryKeyBinding>
  </LocateRequest>
</CompoundRequest>
```

Resposta 1

```
<?xml version="1.0" encoding="utf-8"?>
<CompoundResult Id="If2d286d4a542bd92989aa606d9f1a5ca"
      Service="http://test.xmltrustcenter.org/XKMS" ResultMajor="Success"
      RequestId="I264f5da49b1ff367d4e7aef1f7a1df1a"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <LocateResult Id="I69044d458e0bceef5f78c79c32fa9ddf"
        Service="http://test.xmltrustcenter.org/XKMS" ResultMajor="Success"
        RequestId="I8c26be5f1b4dd228b43fb6eaee285faa">
    <UnverifiedKeyBinding Id="I8f7367375ac134872eab7acf42a8d1bd">
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <KeyValue>
          <RSAKeyValue>

<Modulus>0nIsmR+aVW2egl5MIfOKy4HuMKkk9AZ/IQuDLVPlhzOfgngjVQCjr8uv
                mnqtNu8HBupui8LgGthO
6U9D0CNT5mbmhIAErRADUMIAFsi7LzBarUvNWTqYNEJmcHsAUZdrdcDrkNnG7SzbuJx+GDNiHKVD
QggPBLc1XagW20RMvok=</Modulus>
            <Exponent>AQAB</Exponent>
          </RSAKeyValue>
        </KeyValue>
      </KeyInfo>
      <KeyUsage>Signature</KeyUsage>
      <KeyUsage>Encryption</KeyUsage>
      <KeyUsage>Exchange</KeyUsage>
      <UseKeyWith Application="urn:ietf:rfc:2633"
            Identifier="alice@alicecorp.test" />
    </UnverifiedKeyBinding>
  </LocateResult>
  <LocateResult Id="Ic3d02a8b1f63ba694a8fad11a74fb499"
        Service="http://test.xmltrustcenter.org/XKMS" ResultMajor="Success"
        RequestId="If8e63d729384ad35498e7b65b3dc785e">
    <UnverifiedKeyBinding Id="I42604b6f40f46b74b5c30077100fe8e9">
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <KeyValue>
          <RSAKeyValue>

<Modulus>3FFtWUsvEajQt2SeSF+RvAxWdPPh5GSlQnp8SDvvqvCwE6PXcRWrIGmV
                7twNf2TUXCxYuztUUClM
Iy14B0Q+k1ej2nekmYL7+Ic3DDGVFVaYPoxaRY0Y2lV8tOreynWegpFbITXc8V6Y02QfR5O7Pn1/
10ElslaF/TF8MQGqYE8=</Modulus>
            <Exponent>AQAB</Exponent>
          </RSAKeyValue>
        </KeyValue>
        <X509Data>

<X509Certificate>MIICCTCCAXagAwIBAgIQe0Sk4xr1VolGFFNMkCx07TAJBgUrDg
```

```
                MCHQUAMBIxEDAOBgNVBAMTB1Rl
c3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMDUwODE1MDY1OTU5WjAkMSIwIAYDVQQDExlCb2IgQmFr
ZXIgTz1Cb2IgQ29ycCBDPVVTMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDcUW1ZSy8RqNC3
ZJ5IX5G8DFZO8+HkZKVCenxIO++q8LATo9dxFasgaZXu3A1/ZNRcLFi7O1RQKUwjLXgHRD6TV6Pa
d6SZgvv4hzcMMZUVVpg+jFpFjRjaVXy06t7KdZ6CkVshNdzxXpjTZB9Hk7s+fX/XQSWyVoX9MXwx
AapgTwIDAQABo1YwVDANBgNVHQoEBjAEAwIGQDBDBgNVHQEEPDA6gBABpU6RpUssqgWYs3fukLy6
oRQwEjEQMA4GA1UEAxMHVGVzdCBDQYIQLgyd1ReM8bVNnFUqD4e60DAJBgUrDgMCHQUAA4GBAF4j
P1gGDbaq3rg/Vo3JY7EDNTp0HmwLiPMLmdnB3WTIGFcjS/jZFzRCbvKPeiPTZ6kRkGgydFOuCo5H
MAxIks/LtnKFd/0qYT+AODq/rCrwSx+F+Ro2rf9tPpja9o7gANqxs6Pm7f1QSPZO57bT/6afiVm7
NdaCfjgMphb+XNyn</X509Certificate>

<X509Certificate>MIIB9zCCAWSgAwIBAgIQLgyd1ReM8bVNnFUqD4e60DAJBgUrDg
                MCHQUAMBIxEDAOBgNVBAMTB1Rl
c3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMTAwODE1MDcwMDAwWjASMRAwDgYDVQQDEwdUZXN0IENB
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCn23HHp+HtXpiyKVSDtdE3dO0r0oLB/H9sxUEk
eXB8oMxwbhdcizWH92zrtm1VfVtxkfmwF14ZXoyDZHeZXuCOtAfz/mW6s2gmfD45TfFFVGksDGVR
NK5XmKXA5sEC51RCvaxzGBdGDlCuVPqX7Cq3IcZpRU1IXbi5YzGwV7j6LwIDAQABo1YwVDANBgNV
HQoEBjAEAwIHgDBDBgNVHQEEPDA6gBABpU6RpUssqgWYs3fukLy6oRQwEjEQMA4GA1UEAxMHVGVz
dCBDQYIQLgyd1ReM8bVNnFUqD4e60DAJBgUrDgMCHQUAA4GBABDYD4Fwx2dscu+BgYcZ+GoQQtCJ
kwJEXytb4zlNl7HLFKbXSw4m0blQquIsfsiQgFYAQBXSbu7aeUqqmSGHvILu3BGwVOKjxbHfcM4/
MefuTtpOpCN40wy3YwwngDtHTaIqm8NwS966PE+W9f8kD70q5FNwf+GF/lX9qGc/x435</X509Ce
r
      tificate>
        </X509Data>
      </KeyInfo>
      <KeyUsage>Signature</KeyUsage>
      <KeyUsage>Encryption</KeyUsage>
      <KeyUsage>Exchange</KeyUsage>
      <UseKeyWith Application="urn:ietf:rfc:2633"
            Identifier="bob@bobcorp.test" />
    </UnverifiedKeyBinding>
  </LocateResult>
</CompoundResult>
```

4.3.2 Descrição do Serviço de informação de Chaves

Na especificação de assinatura do *XML* [*XML-SIG*], um assinante pode optar por incluir informação sobre a sua chave pública ("<ds: KeyIfo>") dentro do bloco de assinatura. Essa informação da chave é feita para permitir ao assinante comunicar dicas a um verificador sobre qual chave publica selecionar.

Outra propriedade importante da <ds:KeyInfo> é que ela pode ou não ser ligada criptograficamente a assinatura. Isso permite que a <ds:KeyInfo> possa ser substituida ou suplementada sem "quebrar" a assinatura digital.

Por exemplo, Alice assina um documento e o envia para Bob com o elemento <ds:KeyInfo> que especifica apenas a informação da chave. Ao receber a mensagem, Bob recupera informações adicionais necessárias para validar a assinatura e adiciona essa informação ao elemento <ds:KeyInfo> ao passar o documento adiante para Carol (Figura 1).

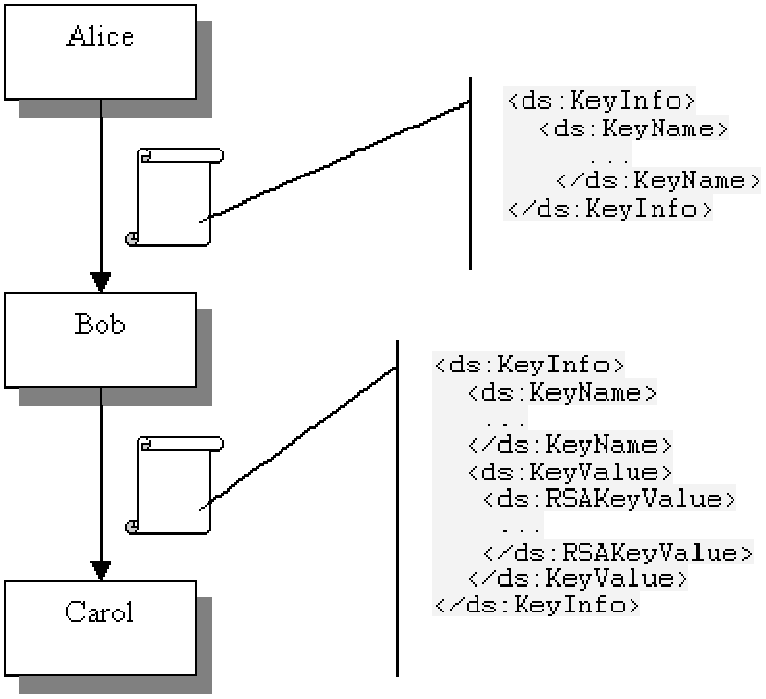

Figura 1: Substituição do elemento <ds:KeyInfo>.

4.3.2.1 *XKISS* Serviço de Localização

O serviço de localização *XKISS* resolve o elemento `<ds: Keyinfo>` mas não requer que se faça uma confirmação da validade da ligação entre a informação no elemento `<ds: KeyInfo>`.

O serviço *XKMS* pode resolver o elemento `<ds: KeyInfo>` usando informações locais ou pode requisitar a outros servidores. Por exemplo, o serviço *XKMS* pode resolver um elemento `<ds: RetrievalMethod>` (Figura 2) ou pode atuar como um *gateway* para uma camada *PKI* baseada em uma sintaxe não *XML*.

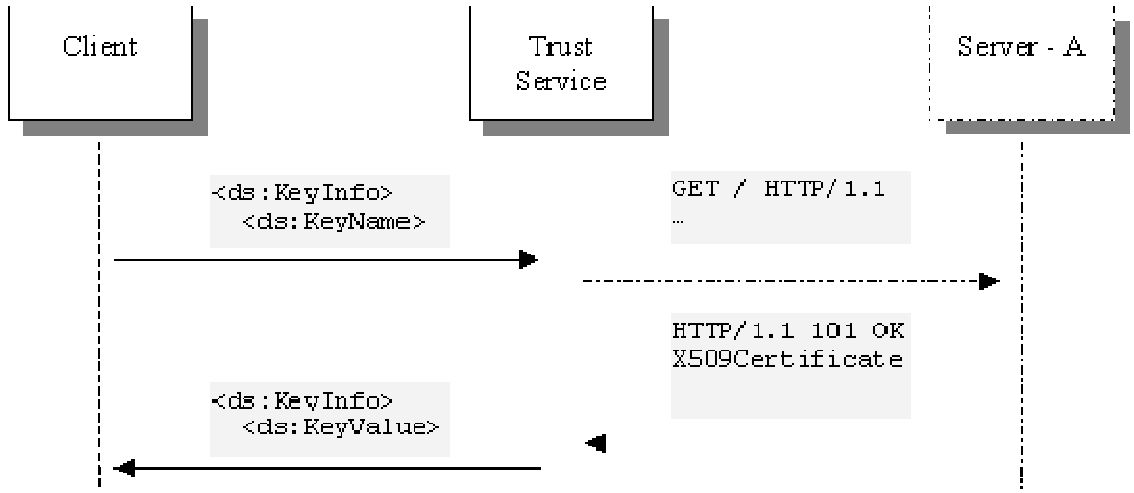

Figura 2: O Serviço de localização fornece a resolução de nome.

Tanto a requisição quanto a resposta podem ser assinadas, para ambas autenticarem o remetente e proteger a integridade da informação sendo transmitida, usando uma assinatura *XML* [*XML-SIG*].

4.3.2.1.1 Exemplo: Encriptação de dados

Alice quer enviar um email encriptografado para Bob, mas não sabe sua chave criptográfica. Alice pode usar tanto o *S/MIME* e o *PGP*, ambos os formatos de email seguros. O cliente de Alice usa o *DNS* para localizar o serviço *XKMS* que fornece um serviço de localização de chaves ligadas ao domínio *example.com* e então envia uma requisição de localização *XKMS* para esse serviço *XKMS* localizado com uma ligação de chave para bob@example.com e o protocolo *S/MIME* ou *PGP*. A aplicação, então, verifica se o certificado obtido obedece aos seus critérios de confiança.

Requisição:

```
<?xml version="1.0" encoding="utf-8"?>
<LocateRequest xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="I8fc9f97052a34073312b22a69b3843b6"
      Service="http://www.example.org/XKMS"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <RespondWith>http://www.w3.org/2002/03/xkms#KeyName</RespondWith>
  <RespondWith>http://www.w3.org/2002/03/xkms#KeyValue</RespondWith>
  <RespondWith>http://www.w3.org/2002/03/xkms#X509Cert</RespondWith>
  <RespondWith>http://www.w3.org/2002/03/xkms#X509Chain</RespondWith>
  <RespondWith>http://www.w3.org/2002/03/xkms#PGPWeb</RespondWith>
  <RespondWith>http://www.w3.org/2002/03/xkms#PGP</RespondWith>
  <QueryKeyBinding>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Encryption</KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:2440"
          Identifier="bob@example.com" />
    <UseKeyWith Application="urn:ietf:rfc:2633"
          Identifier="bob@example.com" />
  </QueryKeyBinding>
</LocateRequest>
```

Resposta:

```
<?xml version="1.0" encoding="utf-8"?>
<LocateResult xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="I8ce3809ab23500015cc27704b7eb0912"
      Service="http://www.example.org/XKMS"
      ResultMajor="http://www.w3.org/2002/03/xkms#Success"
      RequestId="I8fc9f97052a34073312b22a69b3843b6"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <UnverifiedKeyBinding Id="I809ca03cf85b3cb466859694dbd0627d">
    <ds:KeyInfo>
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>
3FFtWUsvEajQt2SeSF+RvAxWdPPh5GSlQnp8SDvvqvCwE6PXcRWrIGmV7twNf2T
UXCxYuztUUClMIy14B0Q+k1ej2nekmYL7+Ic3DDGVFVaYPoxaRY0Y2lV8tOreyn
            WegpFbITXc8V6Y02QfR5O7Pn1/10ElslaF/TF8MQGqYE8=
          </ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
      <ds:X509Data>
        <ds:X509Certificate>
MIICCTCCAXagAwIBAgIQe0Sk4xr1VolGFFNMkCx07TAJBgUrDgMCHQUAMBIxEDA
OBgNVBAMTB1Rlc3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMDUwODE1MDY1OTU5Wj
AkMSIwIAYDVQQDExlCb2IgQmFrZXIgTz1Cb2IgQ29ycCBDPVVTMIGfMA0GCSqGS
```

```
Ib3DQEBAQUAA4GNADCBiQKBgQDcUW1ZSy8RqNC3ZJ5IX5G8DFZ08+HkZKVCenxI
O++q8LATo9dxFasgaZXu3A1/ZNRcLFi7O1RQKUwjLXgHRD6TV6Pad6SZgvv4hzc
MMZUVVpg+jFpFjRjaVXy06t7KdZ6CkVshNdzxXpjTZB9Hk7s+fX/XQSWyVoX9MX
wxAapgTwIDAQABo1YwVDANBgNVHQoEBjAEAwIGQDBDBgNVHQEEPDA6gBABpU6Rp
UssqgWYs3fukLy6oRQwEjEQMA4GA1UEAxMHVGVzdCBDQYIQLgyd1ReM8bVNnFUq
D4e60DAJBgUrDgMCHQUAA4GBAF4jP1gGDbaq3rg/Vo3JY7EDNTp0HmwLiPMLmdn
B3WTIGFcjS/jZFzRCbvKPeiPTZ6kRkGgydFOuCo5HMAxIks/LtnKFd/0qYT+AOD
q/rCrwSx+F+Ro2rf9tPpja9o7gANqxs6Pm7f1QSPZO57bT/6afiVm7NdaCfjgMp
          hb+XNyn
        </ds:X509Certificate>
        <ds:X509Certificate>
MIIB9zCCAWSgAwIBAgIQLgyd1ReM8bVNnFUqD4e60DAJBgUrDgMCHQUAMBIxEDA
OBgNVBAMTB1Rlc3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMTAwODE1MDcwMDAwWj
ASMRAwDgYDVQQDEwdUZXN0IENBMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBg
QCn23HHp+HtXpiyKVSDtdE3dO0r0oLB/H9sxUEkeXB8oMxwbhdcizWH92zrtm1V
fVtxkfmwF14ZXoyDZHeZXuCOtAfz/mW6s2gmfD45TfFFVGksDGVRNK5XmKXA5sE
C51RCvaxzGBdGDlCuVPqX7Cq3IcZpRU1IXbi5YzGwV7j6LwIDAQABo1YwVDANBg
NVHQoEBjAEAwIHgDBDBgNVHQEEPDA6gBABpU6RpUssqgWYs3fukLy6oRQwEjEQM
A4GA1UEAxMHVGVzdCBDQYIQLgyd1ReM8bVNnFUqD4e60DAJBgUrDgMCHQUAA4GB
ABDYD4Fwx2dscu+BgYcZ+GoQQtCJkwJEXytb4zlNl7HLFKbXSw4m0blQquIsfsi
QgFYAQBXSbu7aeUqqmSGHvILu3BGwVOKjxbHfcM4/MefuTtpOpCN40wy3YwwngD
          tHTaIqm8NwS966PE+W9f8kD70q5FNwf+GF/lX9qGc/x435
        </ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature</KeyUsage>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Encryption</KeyUsage>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Exchange</KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:2633"
          Identifier="bob@example.com" />
  </UnverifiedKeyBinding>
</LocateResult>
```

4.3.2.1.2 Exemplo: Assinatura de Documento

Bob recebe o documento assinado por Alice que especifica o certificado X.509v3 de Alice, mas não o valor de sua chave. O cliente de email de Bob não é capaz de processar o certificado X.509v3, mas pode obter os parâmetros da chave no serviço *XKMS* através do serviço de Localização. O cliente de email de Bob envia o elemento `<ds:Keyinfo>` para o serviço de localização requisitando que o elemento `<KeyValue>` correspondente seja retornado.

Requisição:

```
<?xml version="1.0" encoding="utf-8"?>
<LocateRequest xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="I045c66f6c525a9bf3842ecd3466cd422"
      Service="http://www.example.org/XKMS"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <RespondWith>http://www.w3.org/2002/03/xkms#KeyValue</RespondWith>
  <QueryKeyBinding>
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>
MIICEDCCAX2gAwIBAgIQimXeUAxYJbJMady9vV1bLjAJBgUrDgMCHQUAMBIxEDA
OBgNVBAMTB1Rlc3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMDUwODE1MDY1OTU5Wj
ArMSkwJwYDVQQDEyBBbGljZSBBYXJkdmFyayBPPUFsaWNlIENvcnAgQz1VUzCBn
zANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA0nIsmR+aVW2egl5MIfOKy4HuMKkk
9AZ/IQuDLVPlhzOfgngjVQCjr8uvmnqtNu8HBupui8LgGthO6U9D0CNT5mbmhIA
ErRADUMIAFsi7LzBarUvNWTqYNEJmcHsAUZdrdcDrkNnG7SzbuJx+GDNiHKVDQg
```

```
gPBLc1XagW20RMvokCAwEAAaNWMFQwDQYDVR0KBAYwBAMCBkAwQwYDVR0BBDwwO
oAQAaVOkaVLLKoFmLN37pC8uqEUMBIxEDAOBgNVBAMTB1Rlc3QgQ0GCEC4MndUX
jPG1TZxVKg+HutAwCQYFKw4DAh0FAAOBgQABU91ka7IlkXCfv4Zh2Ohwgg2yObt
Y3+6C/BTFGrOEBJDy+DoxJ/NuBF18w3rrrR18xE6jNKYLCQb8zUGk4QOG5Y+HT/
QTTFvWkiOLXcpTuhnOhXatr42FoYpDkjx2QWK+J5Q2l/Rgjgc/0ZV8U/kD8UuRk
          Xp4AZh7QsiX8AcO0w==
        </ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature</KeyUsage>
  </QueryKeyBinding>
</LocateRequest>
```

O serviço de localização extrai o certificado X.509v3 do elemento `<ds:Keyinfo>` e retorna os valores da chave. O serviço de localização não reporta o status de revogação ou se o certificado é confiável.

Resposta:

```
<?xml version="1.0" encoding="utf-8"?>
<LocateResult xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="I04cd4f17d0656413d744f55488369264"
      Service="http://www.example.org/XKMS"
      ResultMajor="http://www.w3.org/2002/03/xkms#Success"
      RequestId="I045c66f6c525a9bf3842ecd3466cd422"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <UnverifiedKeyBinding Id="I012f61e1d7b7b9944fe8d954bcb2d946">
    <ds:KeyInfo>
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>
0nIsmR+aVW2egl5MIfOKy4HuMKkk9AZ/IQuDLVPlhzOfgngjVQCjr8uvmnqtNu8
HBupui8LgGthO6U9D0CNT5mbmhIAErRADUMIAFsi7LzBarUvNWTqYNEJmcHsAUZ
            drdcDrkNnG7SzbuJx+GDNiHKVDQggPBLc1XagW20RMvok=
          </ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature</KeyUsage>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Encryption</KeyUsage>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Exchange</KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:2633"
          Identifier="alice@example.com" />
  </UnverifiedKeyBinding>
</LocateResult>
```

4.3.2.2 *XKISS*: Serviço de Validação

O serviço de validação *XKISS* provê tudo que o serviço de localização provê, e ainda permite que o cliente obtenha uma afirmação com relação ao status da ligação entre a chave publica e outra informação, por exemplo, um nome ou um conjunto de atributos estendidos. Além disso, o serviço descreve que o status de cada um dos elementos da informação retornado é válido e que todos são ligados a mesma chave pública. O cliente envia ao serviço *XKMS* um protótipo contendo parte ou todos os elementos os quais são necessários o status da ligação com a chave. Se a informação no protótipo é incompleta, o serviço *XKMS* pode obter a informação adicional necessária de um serviço *PKI*. Uma vez que a validade da ligação com a chave tenha sido determinada, o serviço *XKMS* retorna o resultado do status ao cliente (Figura 3).

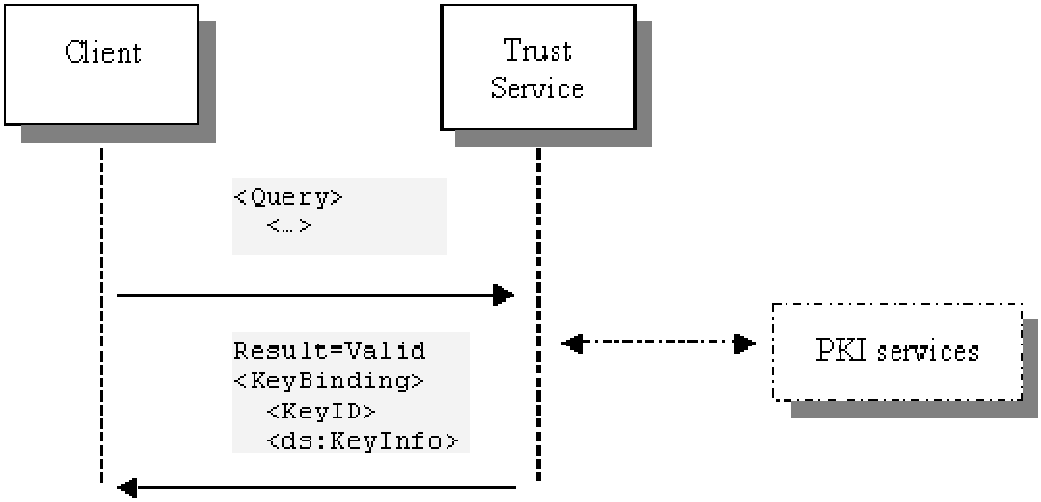

Figura 3: Serviço de validação fornece a validação da chave

4.3.2.2.1 Exemplo: Assinatura de Documento

Bob recebeu uma mensagem de Alice, e seu cliente email verificou a assinatura do documento com a chave pública no certificado fornecido por Alice. Ainda não se sabe porem, se o certificado é confiável. Para determinar isso, o cliente de email de Bob envia o certificado para um serviço de validação *XKMS*. O serviço retorna então, que conseguiu determinar com sucesso que a ligação de chave possui um remetente confiável e que não foi revogada.

Requisição:

```
<?xml version="1.0" encoding="utf-8"?>
<ValidateRequest xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="Ie26380bfeb9d0c5bc526d5213a162d46"
      Service="http://www.example.org/XKMS"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <RespondWith>http://www.w3.org/2002/03/xkms#X509Cert</RespondWith>
  <QueryKeyBinding>
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>
MIICEDCCAX2gAwIBAgIQimXeUAxYJbJMady9vV1bLjAJBgUrDgMCHQUAMBIxEDA
OBgNVBAMTB1Rlc3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMDUwODE1MDY1OTU5Wj
ArMSkwJwYDVQQDEyBBbGljZSBBYXJkdmFyayBPPUFsaWNlIENvcnAgQz1VUzCBn
zANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA0nIsmR+aVW2egl5MIfOKy4HuMKkk
9AZ/IQuDLVPlhzOfgngjVQCjr8uvmnqtNu8HBupui8LgGthO6U9D0CNT5mbmhIA
ErRADUMIAFsi7LzBarUvNWTqYNEJmcHsAUZdrdcDrkNnG7SzbuJx+GDNiHKVDQg
gPBLc1XagW20RMvokCAwEAAaNWMFQwDQYDVR0KBAYwBAMCBkAwQwYDVR0BBDwwO
oAQAaVOkaVLLKoFmLN37pC8uqEUMBIxEDAOBgNVBAMTB1Rlc3QgQ0GCEC4MndUX
jPG1TZxVKg+HutAwCQYFKw4DAh0FAAOBgQABU91ka7IlkXCfv4Zh2Ohwgg2yObt
Y3+6C/BTFGrOEBJDy+DoxJ/NuBF18w3rrrR18xE6jNKYLCQb8zUGk4QOG5Y+HT/
QTTFvWkiOLXcpTuhnOhXatr42FoYpDkjx2QWK+J5Q2l/Rgjgc/0ZV8U/kD8UuRk
          Xp4AZh7QsiX8AcO0w==
        </ds:X509Certificate>
        <ds:X509Certificate>
MIIB9zCCAWSgAwIBAgIQLgyd1ReM8bVNnFUqD4e60DAJBgUrDgMCHQUAMBIxEDA
OBgNVBAMTB1Rlc3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMTAwODE1MDcwMDAwWj
ASMRAwDgYDVQQDEwdUZXN0IENBMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBg
QCn23HHp+HtXpiyKVSDtdE3dO0r0oLB/H9sxUEkeXB8oMxwbhdcizWH92zrtm1V
fVtxkfmwF14ZXoyDZHeZXuCOtAfz/mW6s2gmfD45TfFFVGksDGVRNK5XmKXA5sE
```

```
C51RCvaxzGBdGDlCuVPqX7Cq3IcZpRU1IXbi5YzGwV7j6LwIDAQABo1YwVDANBg
NVHQoEBjAEAwIHgDBDBgNVHQEEPDA6gBABpU6RpUssqgWYs3fukLy6oRQwEjEQM
A4GA1UEAxMHVGVzdCBDQYIQLgyd1ReM8bVNnFUqD4e60DAJBgUrDgMCHQUAA4GB
ABDYD4Fwx2dscu+BgYcZ+GoQQtCJkwJEXytb4zlNl7HLFKbXSw4m0blQquIsfsi
QgFYAQBXSbu7aeUqqmSGHvILu3BGwVOKjxbHfcM4/MefuTtpOpCN40wy3YwwngD
          tHTaIqm8NwS966PE+W9f8kD70q5FNwf+GF/lX9qGc/x435
        </ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature</KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:2633"
          Identifier="alice@example.com" />
  </QueryKeyBinding>
</ValidateRequest>
```

Resposta:

```
<?xml version="1.0" encoding="utf-8"?>
<ValidateResult xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="I34ef61b96f7db2250c229d37a17edfc0"
      Service="http://www.example.org/XKMS"
      ResultMajor="http://www.w3.org/2002/03/xkms#Success"
      RequestId="Ie26380bfeb9d0c5bc526d5213a162d46"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <KeyBinding Id="Icf608e9e8b07468fde1b7ee5449fe831">
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>
MIICEDCCAX2gAwIBAgIQimXeUAxYJbJMady9vV1bLjAJBgUrDgMCHQUAMBIxEDA
OBgNVBAMTB1Rlc3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMDUwODE1MDY1OTU5Wj
ArMSkwJwYDVQQDEyBBbGljZSBBYXJkdmFyayBPPUFsaWNlIENvcnAgQz1VUzCBn
zANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA0nIsmR+aVW2egl5MIfOKy4HuMKkk
9AZ/IQuDLVPlhzOfgngjVQCjr8uvmnqtNu8HBupui8LgGthO6U9D0CNT5mbmhIA
ErRADUMIAFsi7LzBarUvNWTqYNEJmcHsAUZdrdcDrkNnG7SzbuJx+GDNiHKVDQg
gPBLc1XagW20RMvokCAwEAAaNWMFQwDQYDVR0KBAYwBAMCBkAwQwYDVR0BBDwwO
oAQAaVOkaVLLKoFmLN37pC8uqEUMBIxEDAOBgNVBAMTB1Rlc3QgQ0GCEC4MndUX
jPG1TZxVKg+HutAwCQYFKw4DAh0FAAOBgQABU91ka7IlkXCfv4Zh2Ohwgg2yObt
Y3+6C/BTFGrOEBJDy+DoxJ/NuBF18w3rrrR18xE6jNKYLCQb8zUGk4QOG5Y+HT/
QTTFvWkiOLXcpTuhnOhXatr42FoYpDkjx2QWK+J5Q2l/Rgjgc/0ZV8U/kD8UuRk
          Xp4AZh7QsiX8AcO0w==
        </ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature</KeyUsage>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Encryption</KeyUsage>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Exchange</KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:2633"
          Identifier="alice@example.com" />
    <Status StatusValue="http://www.w3.org/2002/03/xkms#Valid">
<ValidReason>http://www.w3.org/2002/03/xkms#Signature</ValidReason>
<ValidReason>http://www.w3.org/2002/03/xkms#IssuerTrust</ValidReason>
<ValidReason>http://www.w3.org/2002/03/xkms#RevocationStatus</ValidRe
ason>
<ValidReason>http://www.w3.org/2002/03/xkms#ValidityInterval</ValidRe
ason>
    </Status>
  </KeyBinding>
</ValidateResult>
```

4.3.2.3 Usando os serviços de Localização e Validação

As operações de validação e localização são ambas usadas para obter informação sobre uma chave publica de um serviço *XKMS*. Espera-se que ambos os serviços de localização e validação tentem fornecer informações corretas ao requisitor

O serviço de localização deve fornecer apenas informações que são confiáveis mas não fornece nenhuma garantia sobre essa informação. A informação obtida de um serviço de localização não deve ser confiada a não ser que seja validada. A validação pode ser obtida direcionando os dados para um serviço de validação.

Um seviço de validação apenas fornece informações que foram validadas positivamente pelo serviço *XKMS*. O cliente pode confiar na informação retornada pelo serviço sem a necessidade de uma validação posterior, levando em consideração de que o cliente pode determinar que a informação retornada é autentica e que o serviço de validação aplicou os meios de validação apropriados a circunstância.

Nenhum conjunto de critérios de validação é apropriado para todas as circunstancias. Aplicações envolvendo transações financeiras provavelmente necessitam de aplicações com critérios de validação muito específicos que garantam que certa política contratual e/ou regulamentatória seja cumprida. O serviço de localização provê uma função de descoberta de chave que é neutra com relação ao critério de validação que o cliente de uma aplicação possa aplicar. O serviço de validação provê uma função de descoberta e validação de chave que produz resultados que são específicos para um conjunto de critérios de validação.

4.3.2.3.1 Integração *DNS*

Em muitos casos a informação da chave que o cliente necessita está ligada com alguma forma de endereço especificado por uma parte de um protocolo de internet que consiste em um endereço *DNS*. Por exemplo, o endereço eletrônico de um cliente pode precisar de uma chave confiável para enviar um email encriptografado para bob@exemplo.com . A menos que o endereço eletrônico no domínio exemplo.com seja conhecido a priori, alguns meios de localizar o correto serviço *XKMS* são necessários.

A figura 4 mostra um exemplo usando um registro *DNS SRV* para localizar serviços *XKMS*. O cliente email requisita uma chave para bob@exemplo.com de um serviço de validação confiável. O serviço de validação, então, consulta o *DNS* para localizar um serviço *XKMS* que provê informação para chaves ligadas ao endereço na zona exemplo.com. O serviço de validação não possui uma relação confiável estabelecida com o serviço *XKMS* do exemplo.com, então uma requisição de localização é feita para determinar se alguma informação ligada a bob@exemplo.com pode ser localizada.O serviço de validação, então, valida a informação recebida da maneira apropriada e a resposta é retornada para o cliente email.

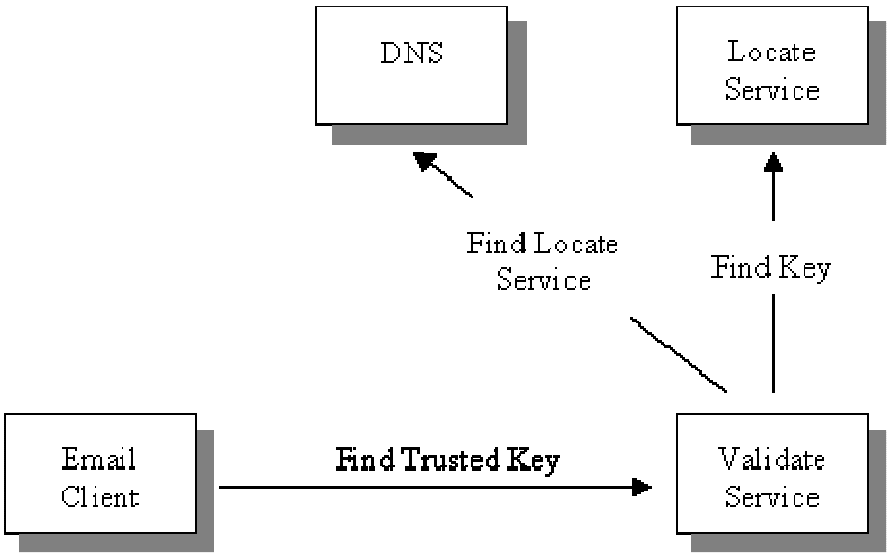

Figura 4: Uso combinado do service de Localização e Validação.

4.3.3 Descrição do Serviço de Registro de Chaves

A especificação do serviço de registro de chaves *XML* permite gerenciar a informação que está ligada a um par de chaves publicas. A especificação do serviço *XKRSS* suporta as seguintes operações:

Registro (Register)
Informação é ligada a um par de chaves publicas através de uma chave.

Alteração (Reissue)
O registro de uma chave registrada previamente é alterada.

Revogação (Revoke)
O registro de uma chave registrada previamente é revogada.

Recuperação (Recover)
A chave privada associada com a ligação da chave é recuperada.

Um serviço *XKMS* pode oferecer todos ou nenhum desses serviços.

A operação ***Register*** por si só não impõe que o serviço de registro comunique essa informação para outros. No entanto, na maioria das aplicações, o serviço de registro provê informações sobre a chave para outros serviços Web como os descritos na especificação.

A operação ***Register*** pode ser usada de um modo que as requisições de um cliente sejam aceitas através de um intermediário como uma autoridade de registro local (*LRA*) e encaminhada para uma autoridade de registro principal (*MRA*). Esse modo de operação é inteiramente transparente para o cliente que atua como se o *LRA* fosse a única autoridade de registro envolvida. A construção da prova de possessão da chave privada e a informação de autenticação é completamente separada da mensagem de autenticação de assinatura. Isso permite a verificação pelo *LRA* e qualquer outra autoridade de registro, mesmo que a mensagem de autenticação da assinatura tenha sido alterada pelo *LRA*.

4.3.3.1 Registro

A requisição ***Register*** é usada para confirmar a ligação de uma informação a um par de chaves publicas. A geração do par de chaves pública pode ser feita tanto pelo cliente quanto pelo serviço de registro.

A mensagem de requisição de registro contém um protótipo da ligação de chave requisitada. O serviço de registro pode requisitar que o cliente envie informação adicional para autenticar a requisição. Se o par de chaves publica é gerado pelo cliente, o serviço pode requisitar que o cliente envie a prova de possessão da chave privada.

O protótipo da ligação de chave requisitada pode conter apenas informação parcial, uma chave sem o nome ou um nome sem a chave. Nesse caso, o cliente estará requisitando que o serviço de registro forneça a informação adicional necessária para completar a ligação. Toda informação contida no protótipo da ligação de chave requisitada é recomendado ao serviço e pode ser ignorada ou sobreposta a escolha do serviço.

Ao receber uma requisição de registro, o serviço verifica a autenticidade e a informação *POP* fornecida (se existente). Se o serviço de registro aceitar a requisição uma ligação de chave é registrada. Essa ligação de chave pode conter nenhuma, parte ou toda a informação fornecida no protótipo e pode incluir informações adicionais.

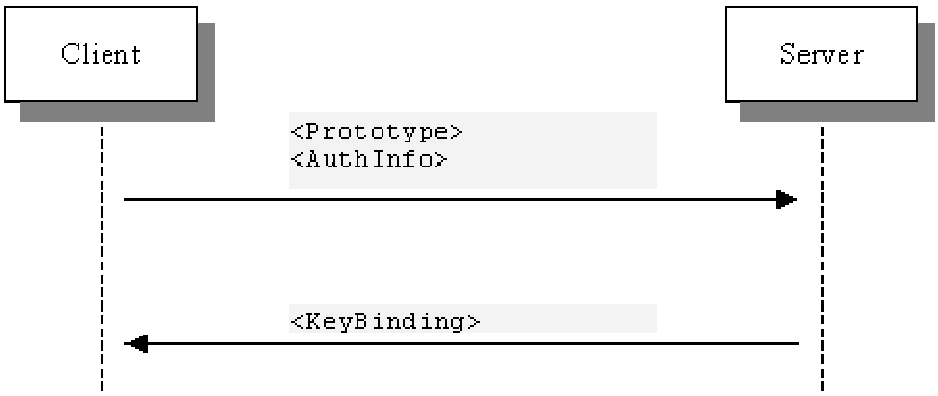

Figura 5: Registro de uma ligação de chave

A escolha de gerar o par de chaves publica no cliente ou no serviço depende da aplicação e do tipo de chave. No caso da chave ser usada para assinaturas, é, geralmente, indesejável que qualquer outro tenha acesso a chave a não ser o responsável pela chave. Caso o acesso a chave privada seja perdido, uma nova chave pode ser criada sem afetar a validade das assinaturas criadas usando a chave privada antiga. É preferível que tais chaves sejam criadas no cliente ao invés do servidor.

No caso da chave privada ser usada exclusivamente para certos tipos de criptografia, a perda do acesso a chave privada pode resultar na perda de acesso a informações armazenadas criptografadas com a chave. Nessas circunstâncias é geralmente desejável alguma forma de recuperação da chave. Nesse caso o par de chaves é geralmente criado no serviço e entregue ao cliente.

Uma chave usada tanto para criptografia quanto para assinatura pode ser gerada no cliente ou no servidor, dependendo se a recuperação de chave deve ser suportada.

4.3.3.1.1 Exemplo: Registro de um par de chaves gerado no Cliente

Alice requisita o registro de um par de chaves *RSA* para seu endereço de email *Alice@example.com*. Ela recebeu, previamente, do serviço *XKMS* o código '024837' que autentica a sua requisição. Alice seleciona a sua frase chave 'Help I have revealed my key' para se autenticar caso seja necessário revogar o registro em uma data posterior.

A mensagem de requisição *X-KRSS* contem o elemento `<RegisterRequest>` . Como a requisição de registro é para um par de chaves gerado pelo cliente, o elemento de autenticação contem tanto um elemento `<ProofOfPossession>`, que demonstra que a requisição é autorizada pelo dono da chave privada, como o elemento `<KeyBindingAuthentication>` que demonstra que a requisição foi feita pela pessoa que conhece o código de autenticação '024837'.

```
<?xml version="1.0" encoding="utf-8"?>
<RegisterRequest xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
   Id="I1494ac4351b7de5c174d455b7000e18f"
   Service="http://www.example.org/XKMS"
   xmlns="http://www.w3.org/2002/03/xkms#">
  <RespondWith>http://www.w3.org/2002/03/xkms#X509Cert</RespondWith>
  <RespondWith>http://www.w3.org/2002/03/xkms#X509Chain</RespondWith>
  <PrototypeKeyBinding Id="I269e655567dbae568591c0a06957529e">
    <ds:KeyInfo>
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>
0nIsmR+aVW2egl5MIfOKy4HuMKkk9AZ/IQuDLVPlhzOfgngjVQCjr8uvmnqtNu8
HBupui8LgGthO6U9D0CNT5mbmhIAErRADUMIAFsi7LzBarUvNWTqYNEJmcHsAUZ
            drdcDrkNnG7SzbuJx+GDNiHKVDQggPBLc1XagW20RMvok=
          </ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature</KeyUsage>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Encryption</KeyUsage>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Exchange</KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:2459"
         Identifier='C="US" O="Alice Corp" CN="Alice Aardvark"' />
    <UseKeyWith Application="urn:ietf:rfc:2633"
         Identifier="alice@example.com" />
    <UseKeyWith
Application="http://ca.example.com/cps/20030401/class3"
         Identifier="alice@example.com" />
    <RevocationCodeIdentifier>
      5AEAai06hFJEkuqyDyqNh8k/u3M=
    </RevocationCodeIdentifier>
  </PrototypeKeyBinding>
  <Authentication>
    <KeyBindingAuthentication>
      <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
          <CanonicalizationMethod
                  Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"
/>
          <SignatureMethod
                  Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-
sha1" />
```

```
          <Reference URI="#I269e655567dbae568591c0a06957529e">
            <Transforms>
              <Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#" />
            </Transforms>
            <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <DigestValue>WCbpkifxJ1zIJ+V6/knZgxRhR34=</DigestValue>
          </Reference>
        </SignedInfo>
        <SignatureValue>iJSKM+98hj5ae+btC2WjwBYP+/k=</SignatureValue>
      </Signature>
    </KeyBindingAuthentication>
  </Authentication>
  <ProofOfPossession>
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
        <CanonicalizationMethod
               Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"
/>
        <SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
        <Reference URI="#I269e655567dbae568591c0a06957529e">
          <Transforms>
            <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#" />
          </Transforms>
          <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
          <DigestValue>WCbpkifxJ1zIJ+V6/knZgxRhR34=</DigestValue>
        </Reference>
      </SignedInfo>
      <SignatureValue>
DcPw742vN120QNrCjCKw0jiCX3pUvbMeRkYjktZkn4nbgo1b7leXU0sJgXM2CY/
oQugaRsgz18+qUzM0UX+jr1t1wtCMci5fjzVKZB63oZyKZ9+CJLcBCbirsgJAId
        +Pq9w4WiwKDf2AytsdXHlN5V1byQIkpfR1CypvBzQa1b4=
      </SignatureValue>
    </Signature>
  </ProofOfPossession>
</RegisterRequest>
```

O serviço aceita o registro e retorna a seguinte resposta:

```
<?xml version="1.0" encoding="utf-8"?>
<RegisterResult xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="I92ed24772e43843b3d23020ad9ec9754"
      Service="http://www.example.org/XKMS"
      ResultMajor="http://www.w3.org/2002/03/xkms#Success"
      RequestId="I1494ac4351b7de5c174d455b7000e18f"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <KeyBinding Id="Ia26450ebe93f62b3b3ab137fc6a61c36">
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>
MIICEDCCAX2gAwIBAgIQimXeUAxYJbJMady9vV1bLjAJBgUrDgMCHQUAMBIxEDA
OBgNVBAMTB1Rlc3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMDUwODE1MDY1OTU5Wj
ArMSkwJwYDVQQDEyBBbGljZSBBYXJkdmFyayBPPUFsaWNlIENvcnAgQz1VUzCBn
zANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA0nIsmR+aVW2egl5MIfOKy4HuMKkk
9AZ/IQuDLVPlhzOfgngjVQCjr8uvmnqtNu8HBupui8LgGthO6U9D0CNT5mbmhIA
ErRADUMIAFsi7LzBarUvNWTqYNEJmcHsAUZdrdcDrkNnG7SzbuJx+GDNiHKVDQg
gPBLc1XagW20RMvokCAwEAAaNWMFQwDQYDVR0KBAYwBAMCBkAwQwYDVR0BBDwwO
oAQAaVOkaVLLKoFmLN37pC8uqEUMBIxEDAOBgNVBAMTB1Rlc3QgQ0GCEC4MndUX
```

```
jPG1TZxVKg+HutAwCQYFKw4DAh0FAAOBgQABU91ka7IlkXCfv4Zh2Ohwgg2yObt
Y3+6C/BTFGrOEBJDy+DoxJ/NuBF18w3rrrR18xE6jNKYLCQb8zUGk4QOG5Y+HT/
QTTFvWkiOLXcpTuhnOhXatr42FoYpDkjx2QWK+J5Q2l/Rgjgc/0ZV8U/kD8UuRk
          Xp4AZh7QsiX8AcO0w==
        </ds:X509Certificate>
        <ds:X509Certificate>
MIIB9zCCAWSgAwIBAgIQLgyd1ReM8bVNnFUqD4e60DAJBgUrDgMCHQUAMBIxEDA
OBgNVBAMTB1Rlc3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMTAwODE1MDcwMDAwWj
ASMRAwDgYDVQQDEwdUZXN0IENBMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBg
QCn23HHp+HtXpiyKVSDtdE3dO0r0oLB/H9sxUEkeXB8oMxwbhdcizWH92zrtm1V
fVtxkfmwF14ZXoyDZHeZXuCOtAfz/mW6s2gmfD45TfFFVGksDGVRNK5XmKXA5sE
C51RCvaxzGBdGDlCuVPqX7Cq3IcZpRU1IXbi5YzGwV7j6LwIDAQABo1YwVDANBg
NVHQoEBjAEAwIHgDBDBgNVHQEEPDA6gBABpU6RpUssqgWYs3fukLy6oRQwEjEQM
A4GA1UEAxMHVGVzdCBDQYIQLgyd1ReM8bVNnFUqD4e60DAJBgUrDgMCHQUAA4GB
ABDYD4Fwx2dscu+BgYcZ+GoQQtCJkwJEXytb4zlNl7HLFKbXSw4m0blQquIsfsi
QgFYAQBXSbu7aeUqqmSGHvILu3BGwVOKjxbHfcM4/MefuTtpOpCN40wy3YwwngD
          tHTaIqm8NwS966PE+W9f8kD70q5FNwf+GF/lX9qGc/x435
        </ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature</KeyUsage>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Encryption</KeyUsage>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Exchange</KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:2459" Identifier="C="US"
          O="Alice Corp" CN="Alice Aardvark"" />
    <UseKeyWith Application="urn:ietf:rfc:2633"
          Identifier="alice@example.com" />
    <UseKeyWith
Application="http://ca.example.com/cps/20030401/class3"
          Identifier="alice@example.com" />
    <Status StatusValue="http://www.w3.org/2002/03/xkms#Valid">
<ValidReason>http://www.w3.org/2002/03/xkms#Signature</ValidReason>
<ValidReason>http://www.w3.org/2002/03/xkms#IssuerTrust</ValidReason>
<ValidReason>http://www.w3.org/2002/03/xkms#RevocationStatus</ValidRe
ason>
<ValidReason>http://www.w3.org/2002/03/xkms#ValidityInterval</ValidRe
ason>
    </Status>
  </KeyBinding>
</RegisterResult>
```

4.3.3.1.2 Exemplo: Registro de um par de chaves gerado no Serviço

A requisição de registro de um par de chaves gerado no serviço omite os dados da chave pública e requer que os dados da chave privada sejam retornados com a resposta.

Bob requisita um par de chaves gerado pelo serviço após receber o código de autenticação 3N9CJ-K4JKS-04JWF-0934J-SR09JW-IK4 através de um mecanismo “out-of-band”. A requisição especifica apenas *Encryption* e *Exchange*.

O serviço gera um par de chaves (publica-privada) em resposta a requisição, gera os certificados apropriados, e retorna como resultado para o cliente. O resultado inclui o valor da chave privada encriptografada usando uma chave derivada do valor do código de autenticação. O cliente pode decriptografar a chave privada computando-a com o valor do código de autenticação da mesma maneira que o serviço.

Para evitar que entidades não autorizadas tenham acesso a chave privada, é necessário que o serviço e o cliente protejam o código de autenticação. O serviço não

deve reutilizar valores de código de autenticação nem deve reutilizar chaves derivadas de um código de autenticação para encriptar mais de uma comunicação de chave privada.

A resposta inclui tanto os dados da chave pública quanto a chave privada encriptografada.

```
<?xml version="1.0" encoding="utf-8"?>
<RegisterResult xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="I2eb0b29bf38eeecfc5f099c8ca149f98"
      Service="http://www.example.org/XKMS"
      ResultMajor="http://www.w3.org/2002/03/xkms#Success"
      RequestId="I4e442fc461a83f320d7a3afb4f2454a9"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <KeyBinding Id="Ia500663f4e4e578447407a38b9049c8b">
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>
MIICCTCCAXagAwIBAgIQe0Sk4xr1VolGFFNMkCx07TAJBgUrDgMCHQUAMBIxEDA
OBgNVBAMTB1Rlc3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMDUwODE1MDY1OTU5Wj
AkMSIwIAYDVQQDExlCb2IgQmFrZXIgTz1Cb2IgQ29ycCBDPVVTMIGfMA0GCSqGS
Ib3DQEBAQUAA4GNADCBiQKBgQDcUW1ZSy8RqNC3ZJ5IX5G8DFZ08+HkZKVCenxI
O++q8LATo9dxFasgaZXu3A1/ZNRcLFi7O1RQKUwjLXgHRD6TV6Pad6SZgvv4hzc
MMZUVVpg+jFpFjRjaVXy06t7KdZ6CkVshNdzxXpjTZB9Hk7s+fX/XQSWyVoX9MX
wxAapgTwIDAQABo1YwVDANBgNVHQoEBjAEAwIGQDBDBgNVHQEEPDA6gBABpU6Rp
UssqgWYs3fukLy6oRQwEjEQMA4GA1UEAxMHVGVzdCBDQYIQLgyd1ReM8bVNnFUq
D4e60DAJBgUrDgMCHQUAA4GBAF4jP1gGDbaq3rg/Vo3JY7EDNTp0HmwLiPMLmdn
B3WTIGFcjS/jZFzRCbvKPeiPTZ6kRkGgydFOuCo5HMAxIks/LtnKFd/0qYT+AOD
q/rCrwSx+F+Ro2rf9tPpja9o7gANqxs6Pm7f1QSPZO57bT/6afiVm7NdaCfjgMp
          hb+XNyn
        </ds:X509Certificate>
        <ds:X509Certificate>
MIIB9zCCAWSgAwIBAgIQLgyd1ReM8bVNnFUqD4e60DAJBgUrDgMCHQUAMBIxEDA
OBgNVBAMTB1Rlc3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMTAwODE1MDcwMDAwWj
ASMRAwDgYDVQQDEwdUZXN0IENBMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBg
QCn23HHp+HtXpiyKVSDtdE3dO0r0oLB/H9sxUEkeXB8oMxwbhdcizWH92zrtm1V
fVtxkfmwF14ZXoyDZHeZXuCOtAfz/mW6s2gmfD45TfFFVGksDGVRNK5XmKXA5sE
C51RCvaxzGBdGDlCuVPqX7Cq3IcZpRU1IXbi5YzGwV7j6LwIDAQABo1YwVDANBg
NVHQoEBjAEAwIHgDBDBgNVHQEEPDA6gBABpU6RpUssqgWYs3fukLy6oRQwEjEQM
A4GA1UEAxMHVGVzdCBDQYIQLgyd1ReM8bVNnFUqD4e60DAJBgUrDgMCHQUAA4GB
ABDYD4Fwx2dscu+BgYcZ+GoQQtCJkwJEXytb4zlNl7HLFKbXSw4m0blQquIsfsi
QgFYAQBXSbu7aeUqqmSGHvILu3BGwVOKjxbHfcM4/MefuTtpOpCN40wy3YwwngD
          tHTaIqm8NwS966PE+W9f8kD70q5FNwf+GF/lX9qGc/x435
        </ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Encryption</KeyUsage>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Exchange</KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:2459" Identifier="C="UK"
          O="Bob Corp" CN="Bob Baker"" />
    <UseKeyWith Application="urn:ietf:rfc:2633"
          Identifier="bob@example.com" />
    <Status StatusValue="http://www.w3.org/2002/03/xkms#Valid">
<ValidReason>http://www.w3.org/2002/03/xkms#Signature</ValidReason>
<ValidReason>http://www.w3.org/2002/03/xkms#IssuerTrust</ValidReason>
<ValidReason>http://www.w3.org/2002/03/xkms#RevocationStatus</ValidRe
ason>
<ValidReason>http://www.w3.org/2002/03/xkms#ValidityInterval</ValidRe
ason>
    </Status>
```

```
  </KeyBinding>
  <PrivateKey>
    <xenc:EncryptedData>
      <xenc:EncryptionMethod
            Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
cbc" />
      <xenc:CipherData>
        <xenc:CipherValue>
Em6xEIvjjlqqeEVOdf9Fq2E6ycz5OZrUU3jw7fBMhiM8BciSC3qry7GAlvRFp/D
iD5zZduSaDriD27HGOuWo1fLihgOcw5sw+G9nyhfxKo7IZ2DEss0DD5H3AAZnTW
reTYwposEBiBcuea4nYwzVWYFsIcplPYsHnXg1cq7MtMfaSu1WA4P/tQLhM1mdO
V6FHkEHdyL4FBaZbjfmkB1Z++Yb0OlOUTNCQLcNxDkF6lNM75sprB+2FtwG1u9x
ZnFXwP0Yt5euwCKeBRFm8Sfsf67CTIjKQ0+l9b66l/W1VszBgq2hMgSi8w+qILw
mFP7p4AhJJNaYlOXHbJAQL6dOe4jtyEkjhEfXXJAO8497Dat9JU5l4Aeb8Mw5BA
KDBT8RKnQTHqRTQZ8h5FPTapD1Av3KllxrIwRJG155enFrVwlxm6mLD+z4WeAdn
U5l9gUSaS13E0PlBApi98zDjILihycV1m90SzUNRIuSnw/8tV0ykS3uquDSmNAO
5YX7UZtieFMyQ2U8XBNRYYftLaN6RfPCejtxXZxIsGtvxyzzL+Yf3bl595J+IOt
n3M73bBvkdq3ACgOG0SCaETlTE5i60Trw19um+f7gAD4QKXawKw72gyQ70GccML
Dh4mypfstCbXxCG5nntGAPkCqT/c+t0TdPc7VrtyFLB3ta9z9yiRkeKqvkVZpvb
lQwykzDd3fgn9ds1liuy8RagrVO6Zczii9b7AQmQ9dekw+QDVkeS53xNTbByau/
aYAjOjIu4Xb6QNPOrNfHYOLMwGuf19o5nAjZEadZU83QhJq/ofwv3EL7tVMlOpF
K7n6Apk7cwmS4OtU71cbAVBMcAsh3KbfrHm0TCy67DEZLtdE9ksTekZMpDKXayQ
uRI0nyKugmwOVvI6oxyMt9JHRiSJRfyYF6yQoyGHGyX8nLX5imG8WNxHBur7j9h
rt8Nf9XRvXY2N9RRfpoDTexN/KZJIZJhDFsY5gsjs1xy0kZqushvZ+jeElGylXs
U/CSXwu3fgeM4qTnMKvJvHl9K4KlDtjIyYIRlBt22n238ADjMLFgUWS4+lyoWZ+
EJPzo0y9CIqb2uUNqp5cSAnb0v7s7gsq6yTqaB9fFSrV0oLH/I65Xoa7XW4mY06
xgwshdC/HHCaTyP/OEgq5Iv2VZUlWTGp1KuEmfko6MaRB/JqMhjNupi1esYTpre
JsBexGjBiCpPPieykC8ngC8GEWLRUyWPdDCybiM/CRFdCvgD5n/Js7nkNY0JGLf
BY0cGnkrxv7e5+5S1Be33yWonJSs8QQ6+y4/V5vPe+5tH8++VFm6dZXtL/oOU0S
DyHruc0yGmdu9GCibRcaU49rPLWbwSfg/g7mVibah2YBXkDDolW1U6pOpry9fHM
kEODQrnoIvGZLaOZ8UQ7jnI92WOTySdxylFL2ZvEHadlUKKN2KtW4zs7nhcQ6Tf
6gjYLNXX9ztkNgCKCZRzI4TXOM2khtPhxTv83rfV0hlx1mtRjlifDdbiWInrCW7
7IPgMEIkEa1oXoKcYb1pUw+W9xzeTp4hTx16izBqC9aWNSYJrT1AvX/Xa+oY8F4
          p+YGGgOSvn9Cb2h1Va8YTb3ntqWaFSE+0/YyOHYgCIsaeYXV9YFN8A+fw
        </xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </PrivateKey>
</RegisterResult>
```

4.3.3.2 Alteração

Um serviço de registro pode permitir que os clientes editem uma ligação de chave registrada previamente. Uma requisição de alteração é similar ao registro inicial de uma chave.

A principal razão pela qual o cliente poderia querer fazer uma requisição de alteração seria para que o serviço de registro gerasse novas credenciais na camada *PKI*. Ex: Certificados X.509.

4.3.3.2.1 Exemplo: Alteração

Alice faz uma requisição de alteração de seu par de chaves *RSA* registrados previamente para seu email.

A mensagem de requisição *X-KRSS* contém o seguinte elemento `<ReissueRequest>`:

```
<?xml version="1.0" encoding="utf-8"?>
<ReissueRequest xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
```

```
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="I3a682dfb94cc8e9b3b648026783a8094"
      Service="http://www.example.org/XKMS"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <RespondWith>http://www.w3.org/2002/03/xkms#X509Cert</RespondWith>
  <RespondWith>http://www.w3.org/2002/03/xkms#X509Chain</RespondWith>
  <ReissueKeyBinding Id="I518fc89b03369bccec3d1ee9d985c436">
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>
MIICEDCCAX2gAwIBAgIQimXeUAxYJbJMady9vV1bLjAJBgUrDgMCHQUAMBIxEDA
OBgNVBAMTB1Rlc3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMDUwODE1MDY1OTU5Wj
ArMSkwJwYDVQQDEyBBbGljZSBBYXJkdmFyayBPPUFsaWNlIENvcnAgQz1VUzCBn
zANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA0nIsmR+aVW2egl5MIfOKy4HuMKkk
9AZ/IQuDLVPlhzOfgngjVQCjr8uvmnqtNu8HBupui8LgGthO6U9D0CNT5mbmhIA
ErRADUMIAFsi7LzBarUvNWTqYNEJmcHsAUZdrdcDrkNnG7SzbuJx+GDNiHKVDQg
gPBLc1XagW20RMvokCAwEAAaNWMFQwDQYDVR0KBAYwBAMCBkAwQwYDVR0BBDwwO
oAQAaVOkaVLLKoFmLN37pC8uqEUMBIxEDAOBgNVBAMTB1Rlc3QgQ0GCEC4MndUX
jPG1TZxVKg+HutAwCQYFKw4DAh0FAAOBgQABU91ka7IlkXCfv4Zh2Ohwgg2yObt
Y3+6C/BTFGrOEBJDy+DoxJ/NuBF18w3rrrR18xE6jNKYLCQb8zUGk4QOG5Y+HT/
QTTFvWkiOLXcpTuhnOhXatr42FoYpDkjx2QWK+J5Q2l/Rgjgc/0ZV8U/kD8UuRk
         Xp4AZh7QsiX8AcO0w==
        </ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <Status StatusValue="http://www.w3.org/2002/03/xkms#Valid" />
  </ReissueKeyBinding>
  <Authentication>
    <KeyBindingAuthentication>
      <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
          <CanonicalizationMethod
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"
/>
          <SignatureMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-
sha1" />
          <Reference URI="#I518fc89b03369bccec3d1ee9d985c436">
            <Transforms>
              <Transform
                    Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#">
                <ec:InclusiveNamespaces PrefixList="ds xenc #default"
                      xmlns:ec="http://www.w3.org/2001/10/xml-exc-
c14n#" />
              </Transform>
            </Transforms>
            <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <DigestValue>qnhsUF9RMxxGydG/5KdJjWhtBFE=</DigestValue>
          </Reference>
        </SignedInfo>
        <SignatureValue>+gKw3b0qi9BaIqmN1gIyvj1UxRs=</SignatureValue>
      </Signature>
    </KeyBindingAuthentication>
  </Authentication>
  <ProofOfPossession>
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
        <CanonicalizationMethod
              Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
```

```
        <SignatureMethod
             Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"
/>
        <Reference URI="#I518fc89b03369bccec3d1ee9d985c436">
          <Transforms>
            <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#">
              <ec:InclusiveNamespaces PrefixList="ds xenc #default"
                   xmlns:ec="http://www.w3.org/2001/10/xml-exc-
c14n#" />
            </Transform>
          </Transforms>
          <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"
               />
          <DigestValue>qnhsUF9RMxxGydG/5KdJjWhtBFE=</DigestValue>
        </Reference>
      </SignedInfo>
      <SignatureValue>
sP/RWAA7fnv86ZgwOlfxTwN05akxyf65rCw7rwXNkJmx0fxUNFJ+qKDqmIh2KyvFyBut6
FredSXj
t3iDIXUKMmjA2/VPGEX8yyd71DbRqf9dXb2FzkvkKrCbYumlavbrChpEwiMUqk2rd5tjk
FAZjYRA
tuURoFfmoOYY/M+mNUU=
      </SignatureValue>
    </Signature>
  </ProofOfPossession>
</ReissueRequest>
```

O serviço aceita o registro e retorna a seguinte resposta:

```
<?xml version="1.0" encoding="utf-8"?>
<ReissueResult xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="I4f0f13b32e4f43f0c1b390b5186fa997"
      Service="http://www.example.org/XKMS"
      ResultMajor="http://www.w3.org/2002/03/xkms#Success"
      RequestId="I3a682dfb94cc8e9b3b648026783a8094"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <KeyBinding Id="I9a894fff4149d2351c24241886e3900e">
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>
MIICEDCCAX2gAwIBAgIQimXeUAxYJbJMady9vV1bLjAJBgUrDgMCHQUAMBIxEDA
OBgNVBAMTB1Rlc3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMDUwODE1MDY1OTU5Wj
ArMSkwJwYDVQQDEyBBbGljZSBBYXJkdmFyayBPPUFsaWNlIENvcnAgQz1VUzCBn
zANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA0nIsmR+aVW2egl5MIfOKy4HuMKkk
9AZ/IQuDLVPlhzOfgngjVQCjr8uvmnqtNu8HBupui8LgGthO6U9D0CNT5mbmhIA
ErRADUMIAFsi7LzBarUvNWTqYNEJmcHsAUZdrdcDrkNnG7SzbuJx+GDNiHKVDQg
gPBLc1XagW20RMvokCAwEAAaNWMFQwDQYDVR0KBAYwBAMCBkAwQwYDVR0BBDwwO
oAQAaVOkaVLLKoFmLN37pC8uqEUMBIxEDAOBgNVBAMTB1Rlc3QgQ0GCEC4MndUX
jPG1TZxVKg+HutAwCQYFKw4DAh0FAAOBgQABU91ka7IlkXCfv4Zh2Ohwgg2yObt
Y3+6C/BTFGrOEBJDy+DoxJ/NuBF18w3rrrR18xE6jNKYLCQb8zUGk4QOG5Y+HT/
QTTFvWkiOLXcpTuhnOhXatr42FoYpDkjx2QWK+J5Q2l/Rgjgc/0ZV8U/kD8UuRk
          Xp4AZh7QsiX8AcO0w==
        </ds:X509Certificate>
        <ds:X509Certificate>
MIIB9zCCAWSgAwIBAgIQLgyd1ReM8bVNnFUqD4e60DAJBgUrDgMCHQUAMBIxEDA
OBgNVBAMTB1Rlc3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMTAwODE1MDcwMDAwWj
ASMRAwDgYDVQQDEwdUZXN0IENBMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBg
QCn23HHp+HtXpiyKVSDtdE3dO0r0oLB/H9sxUEkeXB8oMxwbhdcizWH92zrtm1V
fVtxkfmwF14ZXoyDZHeZXuCOtAfz/mW6s2gmfD45TfFFVGksDGVRNK5XmKXA5sE
C51RCvaxzGBdGDlCuVPqX7Cq3IcZpRU1IXbi5YzGwV7j6LwIDAQABo1YwVDANBg
```

```
NVHQoEBjAEAwIHgDBDBgNVHQEEPDA6gBABpU6RpUssqgWYs3fukLy6oRQwEjEQM
A4GA1UEAxMHVGVzdCBDQYIQLgyd1ReM8bVNnFUqD4e60DAJBgUrDgMCHQUAA4GB
ABDYD4Fwx2dscu+BgYcZ+GoQQtCJkwJEXytb4zlNl7HLFKbXSw4m0blQquIsfsi
QgFYAQBXSbu7aeUqqmSGHvILu3BGwVOKjxbHfcM4/MefuTtpOpCN40wy3YwwngD
          tHTaIqm8NwS966PE+W9f8kD70q5FNwf+GF/lX9qGc/x435
        </ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature</KeyUsage>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Encryption</KeyUsage>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Exchange</KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:2633"
          Identifier="alice@example.com" />
    <Status StatusValue="http://www.w3.org/2002/03/xkms#Valid">
<ValidReason>http://www.w3.org/2002/03/xkms#Signature</ValidReason>
<ValidReason>http://www.w3.org/2002/03/xkms#IssuerTrust</ValidReason>
<ValidReason>http://www.w3.org/2002/03/xkms#RevocationStatus</ValidRe
ason>
<ValidReason>http://www.w3.org/2002/03/xkms#ValidityInterval</ValidRe
ason>
    </Status>
  </KeyBinding>
</ReissueResult>
```

4.3.3.3 Revogação

O serviço de registro pode permitir que os clientes revoguem ligações de chaves registradas previamente. Uma requisição de revogação necessita apenas conter informação suficiente para identificar a ligação de chave a ser revogada e a autoridade para a requisição de revogação.

Se uma ligação de chave *XKMS* está ligada a um objeto de dados da camada *PKI* a revogação para essa ligação de chave deve resultar na revogação do objeto de dados dessa camada.

4.3.3.3.1 Exemplo: Revogação

Por alguma razão Alice faz uma requisição de revogação da ligação de sua chave pública para o serviço. Alice se autentica usando a frase chave estabelecida durante o registro.

A mensagem de requisição é a seguinte:

```
<?xml version="1.0" encoding="utf-8"?>
<RevokeRequest xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="I2aa2c2f37195c9c4364c55f15df68091"
      Service="http://www.example.org/XKMS"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <RevokeKeyBinding Id="Ie91dfbf1c948d5cf142099676968caf1">
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>
MIICEDCCAX2gAwIBAgIQimXeUAxYJbJMady9vV1bLjAJBgUrDgMCHQUAMBIxEDA
OBgNVBAMTB1Rlc3QgQ0EwHhcNMDMwODE1MDcwMDAwWhcNMDUwODE1MDY1OTU5Wj
ArMSkwJwYDVQQDEyBBbGljZSBBYXJkdmFyayBPPUFsaWNlIENvcnAgQz1VUzCBn
zANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA0nIsmR+aVW2egl5MIfOKy4HuMKkk
9AZ/IQuDLVPlhzOfgngjVQCjr8uvmnqtNu8HBupui8LgGthO6U9D0CNT5mbmhIA
ErRADUMIAFsi7LzBarUvNWTqYNEJmcHsAUZdrdcDrkNnG7SzbuJx+GDNiHKVDQg
gPBLc1XagW20RMvokCAwEAAaNWMFQwDQYDVR0KBAYwBAMCBkAwQwYDVR0BBDwwO
```

```
oAQAaVOkaVLLKoFmLN37pC8uqEUMBIxEDAOBgNVBAMTB1Rlc3QgQ0GCEC4MndUX
jPG1TZxVKg+HutAwCQYFKw4DAh0FAAOBgQABU91ka7IlkXCfv4Zh2Ohwgg2yObt
Y3+6C/BTFGrOEBJDy+DoxJ/NuBF18w3rrrR18xE6jNKYLCQb8zUGk4QOG5Y+HT/
QTTFvWkiOLXcpTuhnOhXatr42FoYpDkjx2QWK+J5Q2l/Rgjgc/0ZV8U/kD8UuRk
          Xp4AZh7QsiX8AcO0w==
        </ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <Status
StatusValue="http://www.w3.org/2002/03/xkms#Indeterminate" />
  </RevokeKeyBinding>
  <RevocationCode>PHx8li2SUhrJv2e1DyeWbGbD6rs=</RevocationCode>
</RevokeRequest>
```

O serviço responde que a ligação de chave foi revogada:

```
<?xml version="1.0" encoding="utf-8"?>
<RevokeResult xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="I252433a097631dca9a2775493f39c7d7"
      Service="http://www.example.org/XKMS"
      ResultMajor="http://www.w3.org/2002/03/xkms#Success"
      RequestId="I2aa2c2f37195c9c4364c55f15df68091"
      xmlns="http://www.w3.org/2002/03/xkms#" />
```

4.3.3.4 Recuperação de Chave

Um serviço de registro pode suportar a recuperação de chaves. Para a recuperação de chaves ser possível, a chave privada a ser recuperada deve ter sido previamente inserida no serviço de recuperação, por exemplo, pelo registro de uma chave gerada pelo servidor *XKRSS*. Uma requisição de recuperação é similar ao registro inicial de uma chave com a exceção de que o código de resultado *NotFound* pode ser retornado se o serviço de registro não tiver registro sobre a ligação de chave a ser recuperada.

O serviço de recuperação de chave deve, provavelmente, necessitar de tempo para responder a requisição de recuperação. Os clientes que suportam a recuperação de chaves devem suportar processamento assíncrono.

A política de segurança do remetente pode considerar o processo de recuperação de chave como um potencial risco de comprometimento da chave recuperada e, portanto requisitar a revogação de todas as ligações de chaves associadas, principalmente se a requisição de recuperação de chave tenha sido feita por um terceiro ator como o supervisor do responsável pela chave.

4.3.3.4.1 Exemplo: Recuperação de chave

Bob esqueceu sua chave privada que havia sido obtida no exmplo de registro anterior. Ele primeiro contacta o administrador do serviço de recuperação de chaves usando um procedimento de autenticação “out-of-band” determinado pela política do site. O administrador envia a bob (usando um método “out-of-band”) o código de autorização de recuperação de chave “A8YUT VUHHU C9H29 8Y43U H9J3I 23”.

Os parâmetros de requisição para a recuperação da chave são:

```
<?xml version="1.0" encoding="utf-8"?>
<RecoverRequest xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="I66f40510c322d281602ce76b9eb04d7d"
      Service="http://www.example.org/XKMS"
```

```
      xmlns="http://www.w3.org/2002/03/xkms#">
<RespondWith>http://www.w3.org/2002/03/xkms#PrivateKey</RespondWith>
  <RecoverKeyBinding Id="I29cb8ac8a2ad878f7be44edfe53ea77a">
    <ds:KeyInfo>
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>
3FFtWUsvEajQt2SeSF+RvAxWdPPh5GSlQnp8SDvvqvCwE6PXcRWrIGmV7twNf2T
UXCxYuztUUClMIy14B0Q+k1ej2nekmYL7+Ic3DDGVFVaYPoxaRY0Y2lV8tOreyn
            WegpFbITXc8V6Y02QfR5O7Pn1/10ElslaF/TF8MQGqYE8=
          </ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
    <Status
StatusValue="http://www.w3.org/2002/03/xkms#Indeterminate" />
  </RecoverKeyBinding>
  <Authentication>
    <KeyBindingAuthentication>
      <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
          <CanonicalizationMethod
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"
/>
          <SignatureMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-
sha1" />
          <Reference URI="#I29cb8ac8a2ad878f7be44edfe53ea77a">
            <Transforms>
              <Transform
                    Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#">
                <ec:InclusiveNamespaces PrefixList="ds xenc #default"
                      xmlns:ec="http://www.w3.org/2001/10/xml-exc-
c14n#" />
              </Transform>
            </Transforms>
            <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <DigestValue>GfV3xa/OL6EQAoo5sFL/nHQJCeo=</DigestValue>
          </Reference>
        </SignedInfo>
        <SignatureValue>TtHM/i5L6ynzQHh2Xym8wnbjQ+w=</SignatureValue>
      </Signature>
    </KeyBindingAuthentication>
  </Authentication>
</RecoverRequest>
```

A política desse serviço de registro em particular é a de revogar a chave privada quando uma recuperação de chave for executada. O serviço de registro pode adotar a política "revoke on recover" por várias razões das quais se inclui a preocupação de que o processo de recuperação pode ter comprometido a chave de alguma maneira. O serviço retorna a ligação de chave revogada e os parâmetros da chave privada:

```
<?xml version="1.0" encoding="utf-8"?>
<RecoverResult xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      Id="Iacd24dbd4b3c79660f4d26aca7aaaea2"
      Service="http://www.example.org/XKMS"
      ResultMajor="http://www.w3.org/2002/03/xkms#Success"
```

```
      RequestId="I66f40510c322d281602ce76b9eb04d7d"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <KeyBinding Id="I29cb8ac8a2ad878f7be44edfe53ea77a">
    <ds:KeyInfo>
      <ds:KeyValue>
        <ds:RSAKeyValue>
          <ds:Modulus>
3FFtWUsvEajQt2SeSF+RvAxWdPPh5GSlQnp8SDvvqvCwE6PXcRWrIGmV7twNf2T
UXCxYuztUUClMIy14B0Q+k1ej2nekmYL7+Ic3DDGVFVaYPoxaRY0Y2lV8tOreyn
            WegpFbITXc8V6Y02QfR5O7Pn1/10ElslaF/TF8MQGqYE8=
          </ds:Modulus>
          <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
    <Status StatusValue="http://www.w3.org/2002/03/xkms#Invalid">
<InvalidReason>http://www.w3.org/2002/03/xkms#Signature</InvalidReaso
n>
<InvalidReason>http://www.w3.org/2002/03/xkms#IssuerTrust</InvalidRea
son>
<InvalidReason>http://www.w3.org/2002/03/xkms#RevocationStatus</Inval
idReason>

<InvalidReason>http://www.w3.org/2002/03/xkms#ValidityInterval</Inval
idReason>
    </Status>
  </KeyBinding>
  <PrivateKey>
    <xenc:EncryptedData
Type="http://www.w3.org/2001/04/xmlenc#Content"
                        MimeType="text/xml">
      <xenc:EncryptionMethod
            Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
cbc" />
      <xenc:CipherData>
        <xenc:CipherValue>
DDSIEvW/tshnuCwCC+jX6y/srzMpt3qCIQ5zXmk/cHN4o/BItHsi9BJF85a3hnN
C4/aFfPVJ3WgP6vIZNAUaDY2FbAJckWRgWhGku36p7DZTB14vmt5P5C1bXGU5ps
CDw5Sbm+s/cFkReyfGk6khJTpbgQABPGIZBy2hiZMdSnu0eRgaVUXk9X5oPAhYU
BjiQRq56ckHgEhkuwa+RiA+ybDNn/Ttjt5Uu5BjCONkOdeE8eJMlu0ykj99Vn7G
NKUvt86bJsuwu5ZD1vSmVEwUAvHKV09UfVWfcKEINoj30t8Imj9naJ37oVRNPXl
EKqZY9cxqzHiYEEhu0wxiTMiLbkEyh1DIcLNW78JXpWHRuRTrU5hgNzGEO1pKo7
1uRT1/lArojeJGCJKAwQJvCDXU9zZSXVLzU/AqUshR3L0AoY8pJ/p+LbmlTh43E
4TeT0iXNWkLz0mdgWdmwhhZtj8NcP/4auqfpv7+4NAP5OFVOEYJgE1I60F49K9m
7FbNygiAczfN1YZwjc5lLoIXo75cXduxOZgTWN8ZnFKwrhG1IMhstrauywur6lr
lyxZ1JXEj2aohE4Msa7HKx1LSzDi3dejtK3ZFRqnJcJ1bQ/liOA1IgonN0wvUaH
DM9ibo2xUie2DfoSw3kWCDf0bcZEcV33UFNR3w8kOHgpSAwdHJi0pRyHdgfyd8w
3NzvfCNy9AlrU2MbTdFF7hBxmgFK3fvaX7aEcgdqY17dqkiK75TAwzkVh5WkVjS
WZGSiN49C8e4bY9zzq33lZwZabd5ts2Dvy3RuKc0hQj2rnCZcowXC+XJ7tVtMG+
lNu1ykyeYmvR8VI5Ame5h1DFPjoFLAjkt/tUu2uZlqLYoSKvJU4FWNMAXRUge+f
L3f35lObqwxPJN/LVJgvgGqoMt5hO+/uwgsb3nbR7rTHavPX2kS5LDAtW5xNcfF
dJz8l+dDjlYBJMN8cgEKnNtHTcnVJ5NiFPsGIFv/3IGUZsiw7M4dff3GN6quv9A
601e5rqG1ObMT6/7y7T0Z5IBXwiqs4HcdV+kyRfJwX1QpGat3nQsOZ59PTsIt5n
oKSH0sB5AZmLJa1zgeOILJ574r+F6kD44R32NoLjqu0QL5IqfQ/0lQJuYhn0uEr
FeZIn/lvjqzgf+rGPtgI5wtZ9Fv3qKrTPJOGM8atkzPkUtyJ8kR+WRhdAdFH9HM
0PgyrSGjcCGgFIppsN1KJawrvcXokGRzF81cD/3pVaZC/ZIBtvp4DXM2JSLGoa1
GpLuaIFUP5T/uxFf6MpW2v07bB/jqEZrcsB/ofmvv6RXD/gXrrw99iIiv0k2lyR
sHDN5/syXglGGeskPvCUOZZ5oXrZruxER/IXKRnlsD+0wJ3JZCSuPy9wYmQk77F
pynJ5//7w8UA2qWvkZ0B4rKXOgZYp2pCWaZIDknHJoY+VL7J3sQyAp7qlkQxSBj
bhTEjSYXpHWA+Vj/TiH1ue7/ULlCKfDNvDaWFuEqGT/9H+xUJ5POfTDBhUh+Row
rwCcfYe71B+pB/tylQEERKNpqqgu3TbNJZk5G8U9p41+PwJ0kw9EZnv+z8UEyFl
```

```
          qAZZxj64rTelurPUsiehBFwVwh2fATuMrW9fhGew11npVlF5k+WxE1Cz+
        </xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </PrivateKey>
</RecoverResult>
```

Os parametros da chave privada decriptograda são:

```
<?xml version="1.0" encoding="utf-8"?>
<RSAKeyPair xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns="http://www.w3.org/2002/03/xkms#">
  <Modulus>
3FFtWUsvEajQt2SeSF+RvAxWdPPh5GSlQnp8SDvvqvCwE6PXcRWrIGmV7twNf2TUXCxYu
ztUUClMIy14B0Q+k1ej2nekmYL7+Ic3DDGVFVaYPoxaRY0Y2lV8tOreynWegpFbITXc8V
6Y02QfR5O7Pn1/10ElslaF/TF8MQGqYE8=
  </Modulus>
  <Exponent>AQAB</Exponent>
  <P>
8dnXAObQvcKL0Rr/A+5Ufp1yJgLpH+uohFcJV2kUDCzv5VWKcN+LTq2mciKlnFbAQXKa5
dNPOx4qi3An/4NSMQ==
  </P>
  <Q>
6TUX4d9+enb/aLHtck39x7cPQWlKkNFFJAqXaaRScDhjf4d8b009uW/pu3O2BdIJPY8M/
QnURlqXGLqGGl26fw==
  </Q>
  <DP>
LVEcMFcPlf72I+BjGGF4A6GM2gKBnDGUCfglD/Pohb+F0/sLTcsVV1DCd3B2h6zZqWnID
HhjyDgG0MnbNM1ugQ==
  </DP>
  <DQ>
0DWm7PMtaQl1X3P8G2Gmgvjdlfj7qfAtWtBZ/ufG8oplyyab4oOD6FwSwlm82dV8iol9f
y2XaHjZDir6L/Ae4Q==
  </DQ>
  <InverseQ>
sD2Vl/CCVTDbhhLwdfc4IQDBOh8xpBUV7PPM5LFGjiLetlfwaYi7Bp2ol8WF1MX88iCV2
E3xOPCNfbMhvEB5dA==
  </InverseQ>
  <D>
REUnMUhO6ZX6NxoCwkJ7El5wXAVGtlNJsnPqFygbeEjlBvD6TZx9TqnpP/8IX7WK6JUFW
d9knQJvCWeJjhbjnImSS/3xc+v+m4glnnebZbaghvfunbI++fQaNAFRVT1hLvEGknqC/7
zsrUM04ogU7hP+XgdFTJ1QYGfGHl5c0IE=
  </D>
</RSAKeyPair>
```

4.3.3.5 Autenticação de Requisição

O *X-KRSS* especifica um mecanismo de autenticação de requisição que é independente de qualquer outro mecanismo de autenticação fornecido pela ligação da mensagem. Pela sua natureza, é necessário que o protocolo *X-KRSS* suporte requisições de partes que precisam ainda registrar as suas credenciais ou possuam credenciais danificadas que deverão ser revogadas.

Um serviço *X-KRSS* deve assegurar que todas as requisições são autenticas e autorizadas.

Autenticidade: A mensagem de requisição é originada da parte especificada.
Integridade: A mensagem de requisição não foi modificada.

Possessão: Se uma chave publica é especificada em uma mensagem de requisição, prova que a requisição é autorizada por uma parte que tem acesso a chave privada correspondente.

Os serviços de registro definem as suas próprias políticas de autenticação. Essa especificação define um mecanismo de autenticação que emprega um segredo compartilhado estabelecido entre o cliente e o serviço de registro.

Os serviços devem requisitar que o cliente demonstre a prova de possessão dos componentes da chave privada de uma chave pública se uma requisição é feita para registrar uma ligação de chave válida dessa chave pública.

Os serviços devem aceitar a prova de possessão do componente de chave privada de uma chave pública para efetuar a revogação de qualquer ligação com relação a essa chave.

4.4 *Extensible Access Control Markup Language (XACML)*

O *XACML* (*Extensible Access Control Markup Language*) é uma linguagem baseada no *XML* feito especificamente para criar políticas e automatizar o uso delas, para controlar o acesso de diferentes aplicações em plataformas distintas em uma rede.

Ele descreve tanto uma linguagem de políticas de controle de acesso quanto uma linguagem de requisição/resposta. A linguagem de políticas é usada para expressar as politicas de controle de acesso (quem pode fazer o que, quando). A linguagem de requisição/resposta expressa consultas de acesso (requisições) e descreve respostas a essas consultas.

4.4.1 Requisitos

Os requisitos básicos de uma política de linguagem para expressar uma política de segurança de um sistema de informação são:

- Fornecer um método para combinar regras e políticas individuais em uma única política que se aplique a uma requisição de decisão particular;

- Fornecer um método para definição flexível de um procedimento no qual as regras e políticas são combinadas;

- Fornecer um método para lidar com múltiplos ***subjects*** atuando em diferentes níveis;

- Fornecer um método para basear a decisão de autorização em atributos do s***ubject*** e do recurso;

- Fornecer um método para lidar com atributos multivalorados;

- Fornecer um método para basear a decisão de autorização no conteúdo de um recurso de informação;

- Fornecer um conjunto de operadores lógicos e matemáticos para os atributos do s***ubject***, recurso e ambiente;

• Fornecer um método para manipular um conjunto distribuído de componentes da política, abstraindo os métodos de localização, recuperação e autenticação dos componentes da política;

• Fornecer um método para rapidamente identificar a política que se aplica a determinada ação baseada nos valores dos atributos do s***ubject***, recurso e ação;

• Fornecer uma camada abstrata que isole a política dos detalhes do ambiente da aplicação;

• Fornecer um método para especificar um conjunto de ações que devem ser executadas em conjunto com a política vigente.

A motivação por trás do *XACML* é a de expressar essas idéias bem elaboradas no campo da política de controle de acesso usando uma linguagem de extensão do *XML*. As soluções do *XACML* para cada um desses requisitos são discutidas nas sessões seguintes.

4.4.2 Combinação da regra e política

A política completa aplicada para uma requisição de decisão particular pode ser composta de várias regras e políticas individuais. Por exemplo, em uma aplicação privada pessoal, o dono da informação pode definir certos aspectos de política de divulgação, já uma empresa pode definir outros tipos de aspectos. Para criar uma decisão de autorização, deve ser possível combinar as duas políticas separadas para a requisição.

O *XACML* define três elementos de política: `<Rule>`, `<Policy>` e `<PolicySet>`. O Elemento `<Rule>` contêm uma expressão booleana que pode ser avaliada isoladamente, mas não se tem a intenção de ser acessado pelo *PDP* isoladamente. Portanto não se tem a intenção de formar a base para a decisão de autorização, e sim de existir isoladamente apenas com o *PAP XACML*, onde pode formar uma unidade básica de gerencia, e ser re-utilizada em múltiplas políticas.

O Elemento `<Policy>` contem um conjunto de elementos `<Rule>` e um procedimento específico para combinar os resultados de suas avaliações. É a unidade básica da política utilizada pelo *PDP* e, portanto deve formar a base da decisão de autorização.

O elemento `<PolicySet>` contem um conjunto de elementos `<Policy>` ou outros elementos `<PolicySet>` e um procedimento específico para combinar os resultados das suas avaliações.

4.4.3 Algoritmos de combinação

O *XACML* define alguns algoritmos de combinação que podem ser identificados pelos atributos *RuleCombiningAlgId* ou *PolicyCombiningAlgId* dos elementos `<Policy>` e `<PolicySet>` respectivamente.O algoritmo **rule-*combining*** define um procedimento para chegar a uma decisão de autorização dado os resultados individuais da avaliação de um conjunto de **regras**. Similarmente, o algoritmo ***policy-combining*** define um procedimento para chegar a uma decisão de

autorização dado os resultados individuais de avaliação de um conjunto de ***políticas***. Os algoritmos padrões de combinação são assim definidos:

- *Deny-overrides*,
- *Permit-overrides,*
- *First-applicable* **e**
- *Only-one-applicable*.

No caso do algoritmo *Deny-overrides*, se um único elemento `<Rule>` ou `<Policy>` obtém como resultado da avaliação o valor "*Deny*", então, independente do resultado da avaliação de outros elementos `<Rule>` ou `<Policy>` na politica aplicada, o resultado combinado é "*Deny*".

Do mesmo modo, no caso do algoritmo *Permit-overrides*, se um único resultado "*Permit*" é encontrado, então o resultado combinado será "*Permit*".

No caso do algoritmo de combinação *First-applicable*, o resultado combinado é o mesmo resultado da avaliação do primeiro elemento `<Rule>`, `<Policy>` ou `<PolicySet>` na lista de regras que se aplica a requisição de decisão.

O algoritmo *Only-one-applicable* apenas se aplica as políticas. O resultado desse algoritmo assegura que apenas uma ***policy*** ou ***policy set*** seja aplicada em virtude de suas ***targets***. Se nenhuma ***policy*** ou ***policy set*** se aplica, então o resultado é "NotApplicable", mas se mais de uma ***policy*** ou ***policy set*** se aplicam, então o resultado é "*Indeterminate*". Quando exatamente uma ***policy*** ou ***policy set*** se aplica, o resultado do algoritmo é o resultado da avaliação dessa única ***policy*** ou ***policy set*** aplicada.

Os elementos ***policy*** e ***policy set*** podem levar parâmetros que modificam o comportamento dos algoritmos de combinação. Entretanto, nenhum dos algoritmos padrões são afetados pelos parâmetros.

Os usuários dessa especificação podem, se necessário, definir os seus próprios algoritmos de combinação.

4.4.4 Múltiplos *Subjects*

As políticas de controle de acesso frequentemente colocam requisitos nas ações de mais de um ***subject***. Por exemplo, a politica que gerencia a execução de uma transação financeira de alto nível pode necessitar da aprovação de mais de um indivíduo, atuando em diferentes níveis. Portanto o *XACML* reconhece que pode haver mais de um ***subject*** relevante para a requisição de decisão. Um atributo chamado "subject-category" é usado para diferenciar ***subjects*** atuando em diferentes níveis. Alguns valores padrões para esse atributo são especificados, e os usuários podem definir atributos adicionais.

4.4.5 Políticas baseadas nos atributos de *subjects* e *resources*

Outro requisito comum é basear a decisão de autorização em alguma característica do ***subject*** ao invés de sua identidade. Provavelmente a aplicação mais comum com essa idéia é a ***subject´s role*** [RBAC]. O *XACML* fornece facilidades para suportar essa abordagem. Atributos de ***subjects*** contidos no contexto requisitado podem ser identificados pelo elemento `<SubjectAttributeDesignator>`. Esse elemento contém uma *URN* que identifica o atributo. Como alternativa, o

elemento <AttributeSelector> pode conter uma expressão *XPath* sobre o contexto da requisição para identificar o valor de um atributo particular do ***subject*** através de sua localização no contexto.

O *XACML* fornece um meio padrão de se referenciar atributos, definidos nas series de especificação LDAP [LDAP-1, LDAP-2]. Isso é feito para encorajar os implementadores a usarem os identificadores de atributos padrões ao invés de alguns atributos de ***subjects*** comuns.

Outro requisito comum é basear a decisão de autorização em alguma característica do ***resource*** ao invés de sua identidade. O *XACML* fornece facilidades para suportar essa abordagem. Os atributos do ***resource*** podem ser identificados pelo elemento <ResourceAttributeDesignator>. Esse elemento contém uma *URN* que identifica o atributo. Como alternativa o elemento <AttributeSelector> pode conter uma expressão *XPath* sobre o contexto da requisição para identificar o valor de um atributo de ***resource*** particular pela sua localização no contexto.

4.4.6 Atributos Multivalorados

As técnicas mais comuns para comunicar os atributos (*LDAP*, *XPath*, *SAML*, etc.) suportam múltiplos valores por atributo. Portanto, quando um ***PDP*** *XACML* recupera o valor de um determinado atributo, o resultado pode conter múltiplos valores. Uma coleção desses valores é chamada ***bag***. Uma ***bag*** se diferencia de um conjunto, pois pode conter valores duplicados, já um conjunto não. As vezes essa situação representa um erro. Em alguns casos a regra do *XACML* é satisfeita se qualquer um dos atributos possua um valor que se encaixe no critério expressado na regra.

O *XACML* fornece um conjunto de funções que permite ao criador de políticas ter absoluta clareza sobre como o ***PDP*** deve proceder no caso de haver atributos com múltiplos valores.

4.4.7 Políticas baseadas no conteúdo de resources

Em muitas aplicações é necessário basear a decisão de autorização nos dados contidos na informação do ***resource*** na qual o acesso é requisitado. Por exemplo, um componente comum de política privada é que a pessoa possa ler os registros na qual ele ou ela fazem parte do s***ubject***. A política correspondente deve conter uma referência ao s***ubject*** identificado na própria informação do ***resource***.

O *XACML* fornece facilidades para fazer isso quando a informação do recurso pode ser representada como um documento *XML*. O elemento <AttributeSelector> pode conter uma expressão *XPath* sobre o contexto da requisição para identificar o dado na informação do recurso a ser usada na avaliação da política.

Nos casos em que a informação do recurso não é um documento *XML*, podem ser referenciados atributos específicos do recurso.

4.4.8 Operadores

Políticas de segurança da informação operam sobre atributos de ***subjects***, ***resource, action*** e ***environment*** a fim de chegar a uma decisão de autorização. Nesse

processo, atributos de diferentes tipos poderão ter de ser comparados ou computados. Por exemplo, em uma aplicação financeira, a avaliação do crédito de uma pessoa pode ter de ser calculado pela soma do limite de seu credito com a balança de sua conta. O resultado pode ter então, de ser comparado com o valor da transação. Esse tipo de situação cria a necessidade de operadores aritméticos para atuar sobre atributos do s***ubject*** (balança da conta e limite de crédito) e ***resources*** (valor da transação).

Ainda mais comum, a politica pode identificar um conjunto de papeis que é permitido uma ação em particular executar. A operação correspondente envolve checar se há uma intersecção não nula entre o conjunto de papeis ocupados pelo s***ubject*** e o conjunto de papeis identificados na política. Eis então a necessidade por um conjunto de operações.

O *XACML* possui um vasto número de funções internas e um método para adição de funções. Essas funções podem ser aninhadas para criar expressões complexas, isso é feito utilizando o elemento `<Apply>`. O elemento `<Apply>` possui um atributo *XML* chamado *FunctionId* que identifica a função a ser aplicada e o conteúdo do elemento. Cada função padrão é definida para combinações específicas de argumentos de tipos de dados, e o tipo de dado retornado também é especificado. Portanto, a consistência de tipo de dado da politica pode ser checada no momento em que a política é escrita ou compilada, e os tipos dos valores dos dados presentes no contexto da requisição podem ser checados com os valores esperados pela politica para assegurar um resultado previsível.

Além de operadores para conjuntos de argumentos e argumentos numéricos, são definidos operadores para data, hora e duração de argumentos.

Operadores relacionais (comparação e igualdade) são também definidos para tipos de dados.

Há também os operadores sobre tipos de dados booleanos que permitem a cominação lógica de predicados em uma regra. Por exemplo, uma regra pode conter uma declaração de que o acesso pode ser permitido durante o horário comercial e de um terminal no local de trabalho.

4.4.9 Políticas Distribuídas

Em um sistema distribuído, declarações individuais da política podem ser escritas por diversos autores. Além disso, para facilitar a combinação de componentes independentes da politica, essa abordagem permite que as políticas sejam atualizadas quando necessárias. As declarações das políticas do *XACML* podem ser distribuídas de vários modos, porem não se é descrito nenhum modo normativo de como fazer isso. Independente dos meios de distribuição, se espera que os ***PDPs*** confirmem, examinando o elemento `<Target>` da politica , que a politica é aplicável a requisição de decisão que está sendo processada.

Os elementos `<Policy>` podem ser anexados aos recursos de informação aos quais se aplicam, como descrito por Perritt [Perritt93]. Alternativamente, os elementos `<Policy>` podem ser mantidos em um ou mais locais de onde eles são recuperados para avaliação. Nesses casos a politica aplicada pode ser referenciada por um identificador ou localizador associado com o recurso da informação.

4.4.10 Indexação de Políticas

Para melhorar a eficiência da avaliação e facilitar a gerência, a política em vigência pode ser expressa por múltiplos componentes ***policies*** independentes. Nesse caso é necessário identificar e obter a declaração da politica aplicada e verificar se é a correta para a ação requisitada antes de avaliá-la. Esse é o propósito do elemento no `<Target>` *XACML.*

Duas abordagens são suportadas:

1. Declarações da politica podem ser armazenadas em um banco de dados. Nesse caso o ***PDP*** deve formar uma consulta no banco de dados para obter apenas as políticas que são aplicáveis para o conjunto de requisições de decisão que deverão ser respondidas. Além disso, o ***PDP*** deve avaliar o elemento `<Target>` das declarações ***policy*** ou ***policy set*** obtida como definido pela especificação do *XACML.*

2. Alternativamente, o ***PDP*** pode carregar todas as políticas disponíveis e avaliar os seus elementos `<Target>` no contexto de uma requisição de decisão particular, com o objetivo de identificar as ***policies*** e ***policy sets*** que são aplicáveis a essa requisição.

4.4.11 Camada de Abstração

O ***PEP*** possui várias formas. Por exemplo, um ***PEP*** pode ser parte de um gateway de acesso remoto, servidor *WEB* ou de email, etc ... É irreal esperar que todos os ***PEPs*** em uma empresa façam, ou farão no futuro, uma requisição de decisão para um ***PDP*** em um formato comum, nem que uma política em particular seja aplicada por múltiplos ***PEPs***. Seria ineficiente forçar um criador de políticas a escrever a mesma política de vários modos diferentes para acomodar o formato requerido para cada ***PEP***. Atributos similares podem estar contidos em vários tipos de envelopes (ex.atributos dos certificados X.509, atributos de diretrizes *SAML*, etc.). Portanto existe uma necessidade de uma forma canônica de requisição e resposta manipulada por um ***PDP*** do *XACML.* Essa forma canônica é chamada de contexto *XACML.* Sua sintaxe é definida no *XML* schema.

Naturalmente, ***PEPs*** em conformidade com o *XACML* podem enviar requisições e receber respostas no formado de um contexto *XACML*, mas onde essa situação não existe, um passo intermediário é requerido para converter entre o formato requisição/reposta entendido pelo ***PEP*** e o contexto *XACML* entendido pelo ***PDP***.

O benefício dessa abordagem é que a política pode ser escrita e analisada independente de um ambiente específico na qual ela será aplicada.

No caso onde o formato nativo requisição/resposta é especificado no *XML* schema, a transformação entre o formato nativo e o contexto *XACML* pode ser especificado na forma de um *XSLT* (*Extensible Stylesheet Language Transformation*).

Similarmente, no caso onde o recurso no qual o acesso está sendo requerido ser um documento *XML*, o recurso em si pode ser incluído, ou referenciado, pelo contexto da requisição e através das expressões *XPath* na política, os valores no recurso podem ser incluídos na avaliação da política.

4.4.12 Ações

Em muitas aplicações, as políticas especificam ações que devem ser executadas no lugar de ou em adição a ações que podem ser executadas. Essa idéia foi descrita por Sloman [Sloman94]. O *XACML* fornece facilidades para especificar ações que devem ser executas em conjunto com avaliações da política com o elemento `<Obligations>`. Essa idéia foi descrita como uma ação provisória por Kudo [Kudo00]. Não há definições padrões para esses tipos de ações na versão 2.0 do *XACML*. Entretanto, são necessários acordos bilaterais entre o ***PAP*** e ***PEP*** para reforçar a correta interpretação de suas políticas. ***PEPs*** que estão de acordo com a versão 2.0 do *XACML* são necessários para negar o acesso a menos que entendam e possam satisfazer todos os elementos `<Obligations>` associados com a política aplicada. Os elementos `<Obligations>` são retornados para o ***PEP***.

4.4.13 Modelos (não-normativo)

Os modelos de fluxo de dados e de linguagem do *XACML* são descritos nas seguintes subseções.

4.4.13.1 Modelo de fluxo de dados

Os principais atores no domínio do *XACML* são mostrados no diagrama de fluxo de dados da Figura 6.

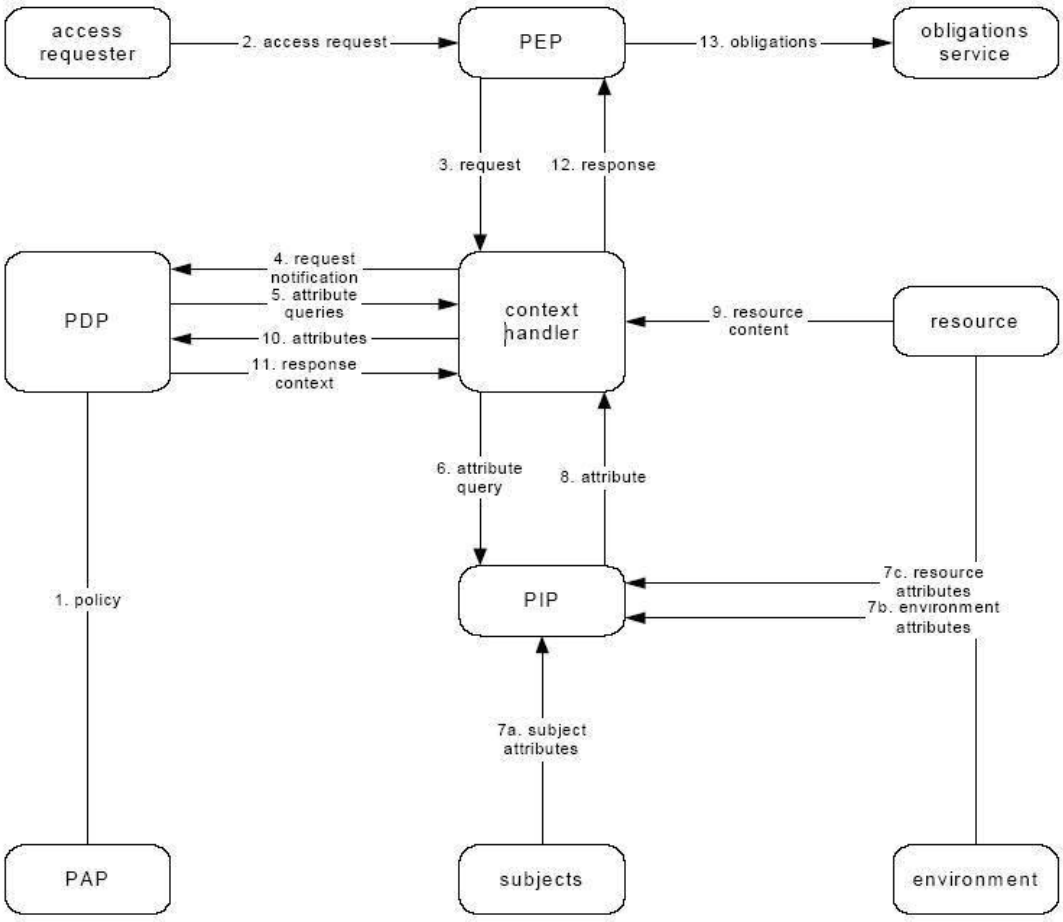

Figura 6: Diagrama de Fluxo de Dados.

Nota: Alguns fluxos de dados mostrados no diagrama podem ser facilitados por um repositório. Por exemplo, as comunicações entre o manipulador de contexto e o ***PIP*** ou a comunicação entre o ***PDP*** e o ***PAP*** podem ser facilitadas pelo repositório. A especificação do *XACML* não pretende colocar restrições sobre a localização de tais repositórios ou de prescrever um protocolo de comunicação particular para qualquer fluxo de dados.

O modelo opera da seguinte maneira:

1. Os ***PAPs*** escrevem políticas e conjuntos de politica e as deixam disponíveis para o ***PDP***. Essas políticas ou conjuntos de política representam a política completa para um alvo especifico.

2. O requisitor de acesso envia uma requisição de acesso para o ***PEP***.

3. O ***PEP*** envia a requisição de acesso para o manipulador de contexto no seu formato nativo de requisição, incluindo, opcionalmente, atributos dos s***ubjects***, recurso, ação e ambiente.

4. O manipulador de contexto constrói um contexto de requisição *XACML* e o envia para o ***PDP***.

5. O ***PDP*** requisita qualquer atributo adicional do s***ubject***, recurso, ação e ambiente para o manipulador de contexto.

6. O manipulador de contexto requisite os atributos de um ***PIP***.

7. O ***PIP*** obtém os atributos requisitados.

8. O ***PIP*** retorna os atributos requisitados para o manipulador de contexto.

9. Opcionalmente, o manipulador de contexto inclui o recurso no contexto.

10. O manipulador de contexto envia os atributos requisitados e (opcionalmente) o recurso para o ***PDP***. O ***PDP*** avalia a política.

11. O ***PDP*** retorna o contexto de resposta (incluindo a decisão de autorização) para o manipulador de contexto.

12. O manipulador de contexto traduz o contexto de resposta para a forma nativa de resposta do ***PEP*** e retorna a resposta para o ***PEP***.

13. O ***PEP*** cumpre as obrigações.

14. (Não mostrado) Se o acesso é permitido, o ***PEP*** permite o acesso ao recurso, caso contrário ele nega o acesso.

4.4.13.2 Contexto *XACML*

O *XACML* foi desenvolvido para ser suportado em vários ambientes de aplicação. O núcleo da linguagem é separado do ambiente da aplicação pelo contexto *XACML*, como mostrado na Figura 7 na qual o escopo da especificação *XACML* é indicado pela área sombreada. O contexto do *XACML* definido no *XML* schema, descrevendo uma representação canônica para as entradas e saídas dos atributos do *PDP* referenciados por uma instancia da política do *XACML* pode estar presente na forma de expressões *XPath* sobre o contexto ou designadores de atributos que identificam o atributo pelo s***ubject***, recurso, ação ou ambiente e seu identificador, tipo de dados e (opcionalmente) seu remetente. As implementações devem converter as representações dos atributos no ambiente da aplicação (ex. *SAML*, *J2SE*, *CORBA*, etc) para a representação de atributos do contexto *XACML*. Como isso é alcançado não faz parte do escopo da especificação do *XACML*. Em alguns casos, como o do *SAML*, a conversão pode ser feita de um modo automático através do uso de uma transformação *XSLT*.

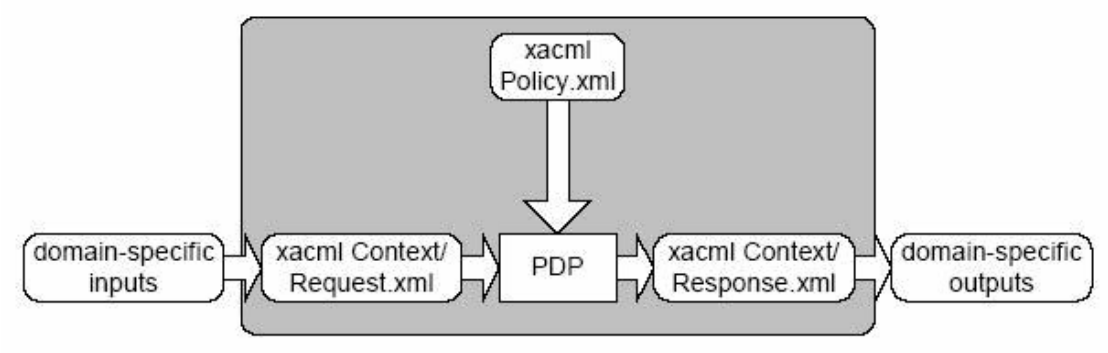

Figura 7: Contexto *XACML*.

Nota: O ***PDP*** não é necessário para operar diretamente sobre a representação da política *XACML*. Ele pode operar diretamente em uma representação alternativa.

4.4.13.3 Modelo de Linguagem da Política

O modelo de linguagem da política é mostrado na Figura 8. Os componentes principais do modelo são:

- ***Regra (Rule);***
- ***Politica (Policy)***; e
- ***Conjunto de politicas (Policy set).***

Esses componentes são descritos nas subseções seguintes.

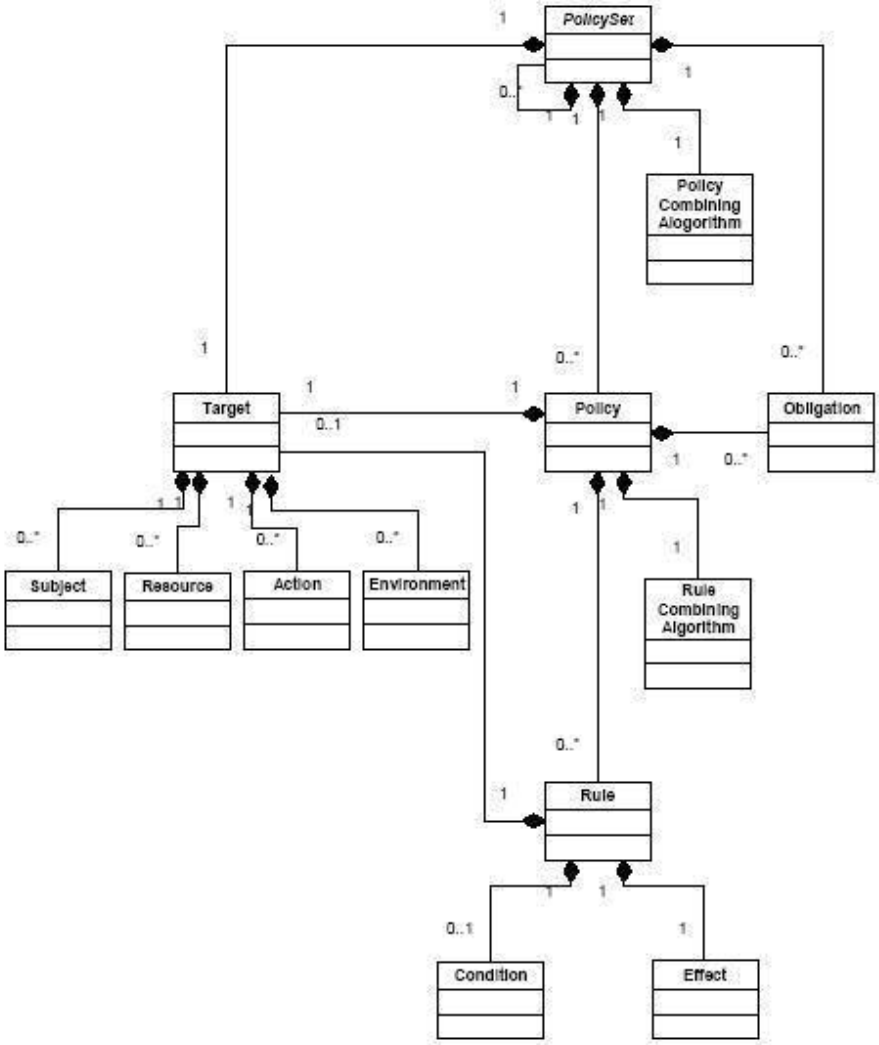

Figura 8: Modelo de linguagem da política.

4.4.13.3.1 Regra (Rule)

A **regra** é a unidade mais elementar da politica. Ela pode existir isolada apenas com um dos atores principais do domínio *XACML*. Para poder haver uma troca de regras entre os atores principais, elas devem estar encapsuladas em uma política. Uma regra pode ser avaliada com base em seu conteúdo. Os principais componentes da regra são:

- um **alvo** ***(target);***
- um **efeito** ***(effect)*** e
- uma **condição** ***(condition).***

Eles são discutidos nas subseções seguintes.

Alvo da regra (*Rule target*)

O alvo define o conjunto de:

- **recursos** ***(resources***);
- **assuntos** ***(subjects);***
- **ações** ***(actions)*** **e**
- **ambiente** ***(environment)***

na qual a **regra** deve ser aplicada. Através do elemento `<Condition>` pode-se ainda refinar a aplicabilidade definida pelo **alvo**. Se a **regra** é definida para ser aplicada em todas as entidades de um tipo de dado em particular, então a entidade correspondente é omitida do **alvo**. Um PDP do XACML verifica se o que foi definido no **alvo** é satisfeito pelos atributos dos **assuntos**, **recursos**, **ações** e **ambiente** no contexto requisitado. As definições de **alvo** são discretas para que as **regras** aplicadas sejam identificas eficientemente pelo ***PDP***.

O elemento `<Target>` pode não estar presente em uma **regra**. Nesse caso o **alvo** de uma **regra** é herdado do elemento pai (`<Policy>`).

Efeito (*Effect*)

O efeito de uma regra indica a conseqüência esperado pelo criador da regra de uma avaliação "verdadeira" para a regra. Dois valores são permitidos "*Permit*" e "*Deny*".

Condição (*Condition*)

A condição representa uma expressão booleana que define a aplicabilidade da regra alem do predicado implicado pelo seu alvo. Portanto pode-se estar ausente.

4.4.13.3.2 Política (Policy)

Pelo modelo de fluxo de dados nota-se que as regras não são trocadas entre as entidades do sistema. Portanto o ***PAP*** combina as regras em uma política. Uma política é compreendida por quatro componentes principais:

- um alvo (***target***);
- um identificador de algoritmo de combinação de regra (***rule-combining algorithm***-identifier);
- um conjunto de regras (*set of* ***rules***);
- obrigações (***obligations***).

As ***regras*** estão descritas acima. Os demais componentes estão descritos nas subseções seguintes.

Alvo da política (Policy target)

Um elemento *XACML* `<PolicySet>`, `<Policy>` ou `<Rule>` contem um elemento `<Target>` que especifica o conjunto de assuntos, recursos, ações e ambientes nos quais ele se aplica. O `<Target>` de um elemento `<PolicySet>` ou `<Policy>` pode ser declarado pelo escritor da `<PolicySet>` ou `<Policy>`,

ou pode ser calculado pelos elementos <Target> contidos nos elementos <PolicySet>, <Policy> e <Rule>.

Uma entidade do sistema que calcula o elemento <Target> desse modo é definido pelo *XACML*, mas há dois métodos lógicos que podem ser usados. Em um dos métodos, o elemento <Target> da <PolicySet> ou <Policy> exterior (o "componente exterior") é calculado como a união de todos os elementos <Target> dos elementos <PolicySet>, <Policy> ou <Rule> referenciados (os "componentes internos"). No outro método, o elemento <Target> do componente exterior é calculado como a intersecção de todos os elementos <Target> dos componentes interiores. O resultado da avaliação em cada caso será muito diferente. No primeiro caso o elemento <Target> do componente exterior o faz aplicável para qualquer requisição de decisão que combine o elemento <Target> a pelo menos um componente interior; no segundo caso, o elemento <Target> do componente exterior o faz aplicável apenas as requisições de decisão que combine os elementos <Target> de todos os componentes interiores. Note que computar a intersecção de um conjunto de elementos <Target> é possível se o modelo de dados alvo é relativamente simples.

Nos casos onde o <Target> da <Policy> é declarado pelo escritor da politica, qualquer elemento <Rule> na <Policy> que possua o mesmo elemento <Target> que o elemento <Policy> pode omitir o elemento <Target>. Tais elementos <Rule> herdam o <Target> da <Policy> na qual estão contidos.

Algoritmo de combinação de regras

O algoritmo de combinação de regras (***rule-combining algorithm***) especifica o procedimento no qual os resultados da avaliação dos componentes de regras são combinados para avaliar a politica. ex. O valor de decisão colocado no contexto de resposta pelo *PDP* é o valor da politica, como definido pelo algoritmo de combinação de regras. Uma politica pode ter combinações de parâmetros que afetam a operação do algoritmo de combinação de regras.

Obrigações (Obligations)

Obrigações podem ser adicionadas pelo criador da politica. Quando um ***PDP*** avalia uma politica que contem obrigações, ele retorna algumas dessas obrigações ao ***PEP*** no contexto de resposta.

4.4.13.3.3 Conjunto de Políticas (Policy set)

Um conjunto de políticas é compreendida de quatro componentes principais:

- um alvo (***target***);
- um identificador de algoritmo de combinação de políticas (***policy-combining algorithm-identifier***)
- um conjunto de políticas (***set of policies)***; e
- obrigações (***obligations***).

O alvo e os componentes da politica estão descritos acima. Os outros componentes são descritos nas subseções seguintes.

Algoritmo de combinação de Políticas

O algoritmo de combinação de políticas especifica o procedimento no qual o resultado da avaliação dos componentes das políticas é combinado para avaliar o conjunto de políticas. ex. O valor de decisão colocado no contexto de resposta pelo ***PDP*** é o resultado da avaliação do conjunto de políticas como definido pelo algoritmo de combinação de políticas. Um conjunto de políticas pode ter combinações de parâmetros que afetam a operação do algoritmo de combinação de políticas.

Obrigações (Obligations)

O escritor do conjunto de políticas pode adicionar obrigações para o conjunto de políticas, além daquelas contidas nos componentes de politica e de conjunto de políticas. Quando um ***PDP*** avalia um conjunto de políticas que possui obrigações, ele retorna algumas dessas obrigações para o ***PEP*** em seu contexto de resposta.

Capítulo 5 Uso da biblioteca XmlSec (C)

Utilizamos a biblioteca '*xmlsec*' que implementa as especificações de assinaturas digitais e criptografia de documentos *XML*. A biblioteca foi escrita na linguagem *C*.

5.1 A estrutura da biblioteca

Para fornecer a capacidade de usar máquinas de crifragem diferentes, a Biblioteca de Segurança *XML* é dividida em duas partes: centro da biblioteca (*xmlsec*) e biblioteca de criptografia (*xmlsec-openssl*, *xmlsec-gnutls*, *xmlsec-nss*, etc).

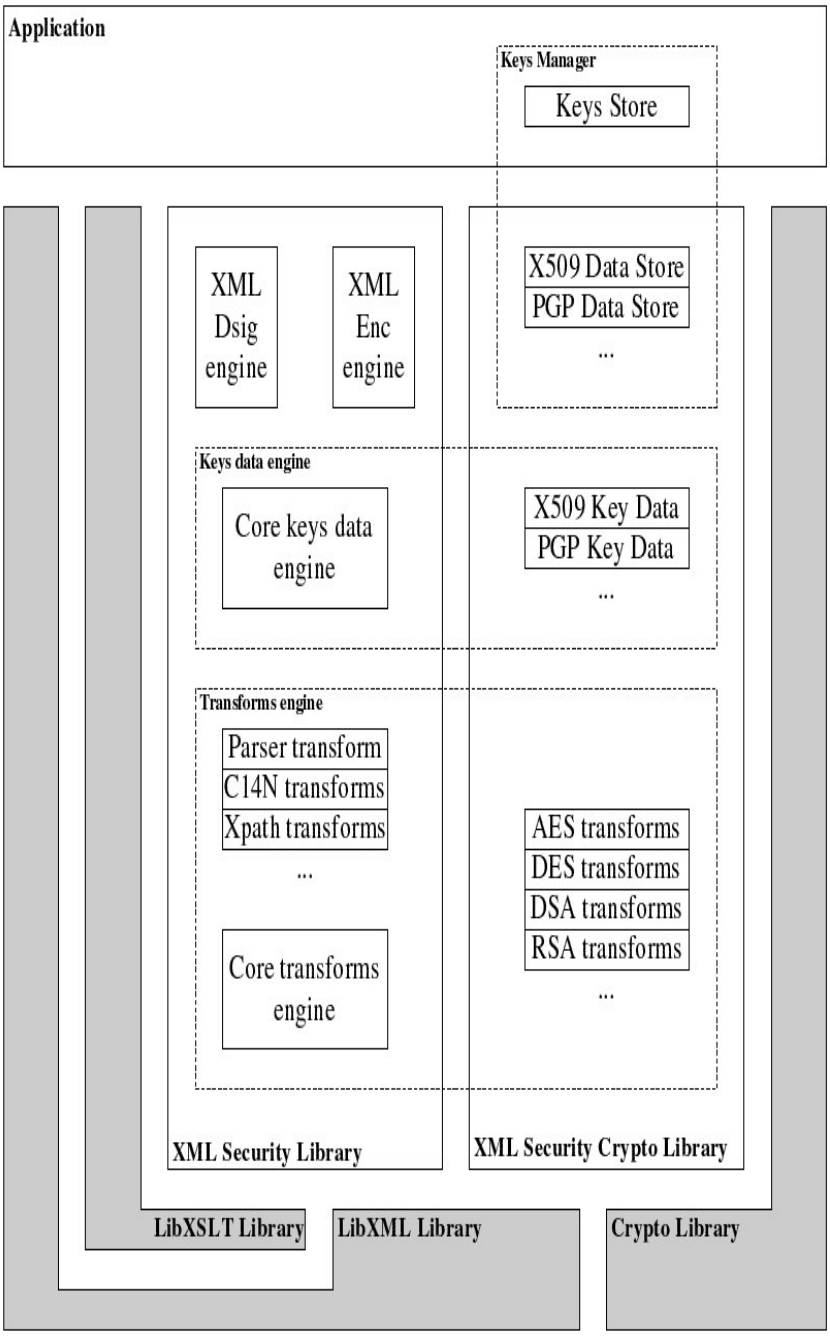

Figura 9: Biblioteca xmlsec.

5.2 Assinando e Criptografando Documentos

A biblioteca executa a assinatura ou criptografia processando o xml de entrada ou dados binários e um modelo que especifica a assinatura ou esqueleto da criptografia: o *transforms*, algoritmos, o processo de seleção da chave. Um modelo tem a mesma estrutura que o resultado desejado, mas alguns dos nodos estão vazios. A biblioteca obtém a chave para assinatura/criptografia de gerentes de chaves que usam a informação do modelo, fazem as computações necessárias e põem os resultados no modelo. O contexto da assinatura ou da criptografia controla o processo inteiro e guarda os dados temporários necessários.

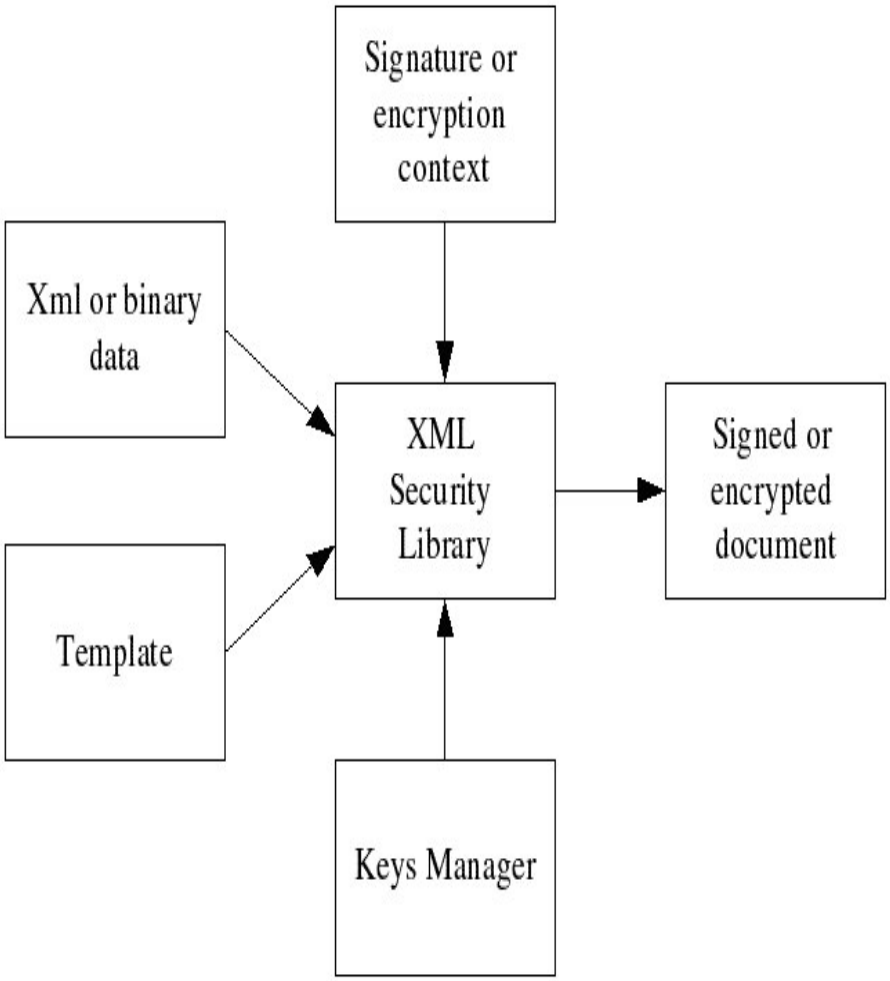

Figura 10: Assinando e criptografando documentos.

5.2.1 Assinando dados

O típico processo de assinatura inclui os seguintes passos:
- Criar ou carregar um modelo de assinatura e selecionar o nodo inicial, <dsig:Signature/>

```
/* create signature template for RSA-SHA1 enveloped signature */
    signNode = xmlSecTmplSignatureCreate(doc, xmlSecTransformExclC14NId,

/* add <dsig:Signature/> node to the doc */
    xmlAddChild(xmlDocGetRootElement(doc), signNode);
```

```
dsigCtx = xmlSecDSigCtxCreate(NULL);
```

- Carregar a chave de assinatura no gerente de chaves ou gerar uma chave de sessão e fixá-la no contexto da assinatura (*signKey*, membro da estrutura xmlSecDSigCtx).

```
dsigCtx->signKey = xmlSecCryptoAppKeyLoad(key_file,
xmlSecKeyDataFormatPem, NULL, NULL, NULL);
```

- Assinar os dados chamando a função xmlSecDSigCtxSign

```
xmlSecDSigCtxSign(dsigCtx, signNode);
```

- Destruir contexto da assinatura xmlSecDSigCtx usando as funções xmlSecDSigCtxDestroy ou xmlSecDSigCtxFinalize

```
xmlSecDSigCtxDestroy(dsigCtx);
```

5.2.2 Criptografando Dados

O típico processo de criptografia inclui os seguintes passos:

- Criar ou carregar o modelo de criptografia e selecionar o nodo inicial <enc:EncryptedData />

```
/* create encryption template to encrypt XML file and replace
     * its content with encryption result */
    encDataNode = xmlSecTmplEncDataCreate(doc, xmlSecTransformDes3CbcId,
                                NULL, xmlSecTypeEncElement, NULL, NULL);
```

- Criar o contexto da criptografia xmlSecEncCtx usando a função xmlSecEncCtxCreate ou xmlSecEncCtxInitialize

```
encCtx = xmlSecEncCtxCreate(mngr);
```

- Carregar a chave para criptografia no gerente de chaves ou gerar uma chave de sessão e fixá-la no contexto da criptografia (encKey membro da estrutura xmlSecEncCtx)

```
encCtx->encKey = xmlSecKeyGenerate(xmlSecKeyDataDesId, 192,
xmlSecKeyDataTypeSession);
```

- Criptografar os dados chamando uma das seguintes funções:
  - xmlSecEncCtxBinaryEncrypt
  - xmlSecEncCtxXmlEncrypt
  - xmlSecEncCtxUriEncrypt

```
xmlSecEncCtxXmlEncrypt(encCtx, encDataNode, xmlDocGetRootElement(doc));
```

- Destruir o contexto da criptografia xmlSecEncCtx usando a função xmlSecEncCtxDestroy ou xmlSecEncCtxFinalize

```
xmlSecEncCtxDestroy(encCtx);
```

5.2.3 Verificando e decriptografando documentos

Considerando que o modelo é apenas um arquivo *XML*, este poderia ser criado com antecedência e poderia ser salvo em um arquivo. Também é possível a aplicação criar modelos sem usar as funções da biblioteca *xmlsec*. Em alguns casos o modelo deve ser inserido nos dados assinados ou criptografados.

A verificação da assinatura e dos dados decriptografados não requer um modelo porque toda a informação necessária é provida no documento assinado ou criptografado.

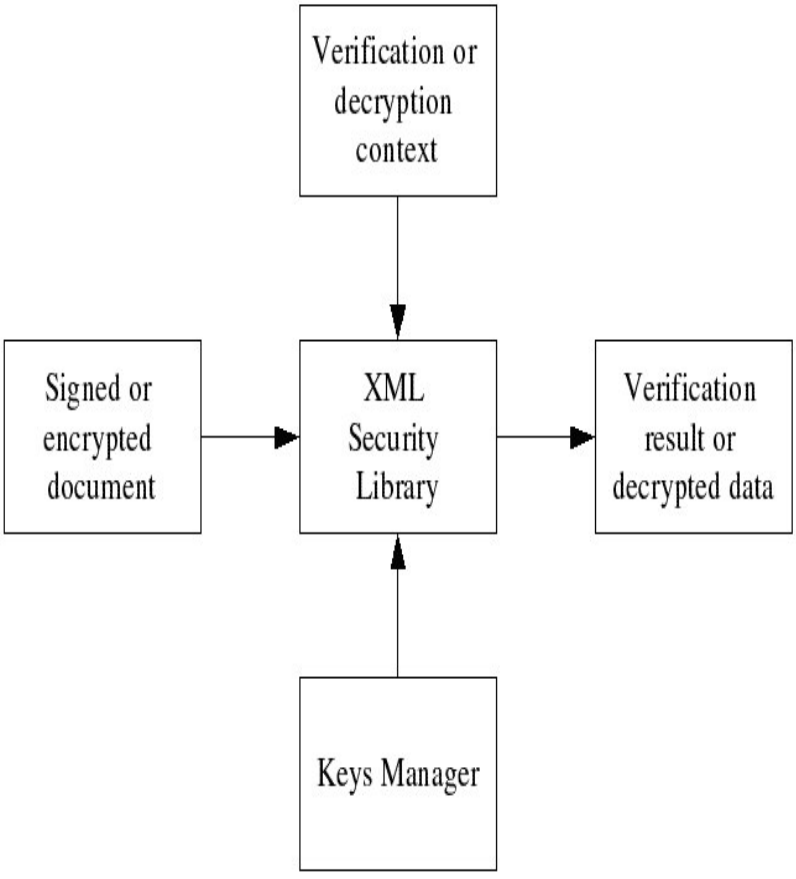

Figura 11: Verificando e decriptografando documentos.

5.2.4 Verificando um documento assinado

O processo típico de verificação de uma assinatura inclui os seguintes passos:

- Carregar chaves, certificado X509 , etc. no gerente de chaves

```
/* load public key */
    dsigCtx->signKey = xmlSecCryptoAppKeyLoad(key_file,
xmlSecKeyDataFormatPem, NULL, NULL, NULL);
```

- Criar o contexto da assinatura xmlSecDSigCtx usando a função xmlSecDSigCtxCreate ou xmlSecDSigCtxInitialize

```
dsigCtx = xmlSecDSigCtxCreate(NULL);
```

- Selecionar o nodo inicial de verificação <dsig:Signature/> no documento XML assinado

```
node = xmlSecFindNode(xmlDocGetRootElement(doc), xmlSecNodeSignature,
xmlSecDSigNs);
```

- Verificar a assinatura chamando a função xmlSecDSigCtxVerify

```
xmlSecDSigCtxVerify(dsigCtx, node);
```

- Checar o valor de retorno e estado da verificação (status membro da estrutura xmlSecDSigCtx)

```
if(xmlSecDSigCtxVerify(dsigCtx, node) < 0)
```

- Destruir o contexto da assinatura xmlSecDSigCtx usando a função xmlSecDSigCtxDestroy ou xmlSecDSigCtxFinalize

```
xmlSecDSigCtxDestroy(dsigCtx);
```

5.2.5 Decifrando um documento criptografado

O processo típico de decriptografia inclui os seguintes passos:

- Carregar chaves, certificado X509 , etc. no gerente de chaves

```
key = xmlSecCryptoAppKeyLoad(name, xmlSecKeyDataFormatPem, NULL, NULL,
NULL);
```

- Criar o contexto da criptografia xmlSecEncCtx usando a função xmlSecEncCtxCreate ou xmlSecEncCtxInitialize

```
encCtx = xmlSecEncCtxCreate(mngr);
```

- Selecionar o nodo inicial de decriptografia <enc:EncryptedData>

```
/* find start node */
    node = xmlSecFindNode(xmlDocGetRootElement(doc),
xmlSecNodeEncryptedData, xmlSecEncNs);
```

- Decriptografar utilizando a função xmlSecencCtxDecrypt
- Verificar o valor de retorno

```
if((xmlSecEncCtxDecrypt(encCtx, node) < 0) || (encCtx->result == NULL))
```

- Destruir o contexto da criptografia xmlSecEncCtx usando a função xmlSecEncCtxDestroy ou xmlSecEncCtxFinalize

```
xmlSecEncCtxDestroy(encCtx);
```

Capítulo 6 Considerações Finais

6.1 Conclusão

Descrevemos a utilização de modelos *XML* para obtenção de autenticação, autorização, não repúdio, sigilo e integridade. Percebemos que com o aumento do uso de dados no formato *XML*, a segurança pode ser inserida dentro dos próprios documentos, e de forma padronizada, para que um documento possa ser entendido por qualquer sistema compatível, sendo assim intercambiável, ou seja, não é necessário uma análise prévia da estrutura do documento, já que está é conhecida pelo padrão *XML*.

Vimos também que o *XKMS* fornece um mecanismo abstraido para utilizar e integrar o *PKI* (*Public Key Infrastructure*) em aplicações, possibilitando o registro e recuperação de informações relacionadas a chaves públicas e/ou privadas, facilitando e diminuindo o tempo de desenvolvimento de aplicações cliente que necessitem de um ou mais serviços providos pelo *PKI*, já que todo processo criptografico ficaria por parte de um servidor *XKMS*.

Por último, analisamos o *XACML*, que é uma tentativa de padronizar uma linguagem para políticas de segurança e decisões de acesso e que provê estruturas para manipular múltiplas regras, políticas e conjuntos de políticas e diferentes algoritmos para processar as requisições de forma eficiente. Hoje cada sistema possui seu próprio padrão, sendo necessário a reescrita da regra para o mesmo recurso em cada diferente sistema, sendo necessário um esforço múltiplo para uma regra já existente. O *XACML* seria uma solução, já que todos os sistemas compatíveis com o padrão, podem utilizar a mesma regra para um determinado recurso.

Referências Bibliográficas

[STA99] STALLINGS, W. **Cryptography and Network Security: Principles and** Practice. 1999.

[BUR02] BURNETT, S.; PAINE, S. **Criptografia e segurança - O guia oficial RSA**. Rio de Janeiro: Editora Campus, 2002.

[XML-SIG] D. Eastlake, J. R., D. Solo, M. Bartel, J. Boyer , B. Fox , E. Simon. *XML-Signature Syntax and Processing*, W3C Recommendation, 12 February 2002. http://www.w3.org/TR/xmldsig-core/

[Kudo00] Kudo M and Hada S, XML document security based on provisional authorization, Proceedings of the Seventh ACM Conference on Computer and Communications Security, Nov 2000, Athens, Greece, pp 87-96.

[LDAP-1] RFC2256, A summary of the X500(96) User Schema for use with LDAPv3, Section 5, M Wahl, December 1997 **http://www.ietf.org/rfc/rfc2798.txt**

[LDAP-2] RFC2798, Definition of the inetOrgPerson, M. Smith, April 2000 **http://www.ietf.org/rfc/rfc2798.txt**

[Perritt93] Perritt, H. Knowbots, Permissions Headers and Contract Law, Conference on Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment, April 1993. Disponível em: **http://www.ifla.org/documents/infopol/copyright/perh2.txt**

[RBAC] Role-Based Access Controls, David Ferraiolo and Richard Kuhn, 15th National Computer Security Conference, 1992. Disponível em: **http://csrc.nist.gov/rbac**

[Sloman94] Sloman, M. Policy Driven Management for Distributed Systems. Journal of Network and Systems Management, Volume 2, part 4. Plenum Press. 1994.

[W3C] W3C, World Wide Web Consortium <http://www.w3.org/ > Acesso em 06/2005.

Anexos

A-1) Documento XML a ser criptografado

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="urn:envelope">
  <Data>
        Hello, World!
  </Data>
</Envelope>
```

A-2) Documento XML resultante da criptografia

```
<?xml version="1.0" encoding="UTF-8"?>
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
Type="http://www.w3.org/2001/04/xmlenc#Element">
<EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-
oaep-mgf1p"/>
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<KeyName>rsakey.pem</KeyName>
</KeyInfo>
<CipherData>
<CipherValue>IPiEu9Nv+EsGyvVeXO9nl5iZhhi+uzQH1I3/DTs3+eamBvioyaawRIlv
Tql7LYL5
Mi91Qs8ozfW/fWZ8zB8AE2PosaX37SqiuEta68+65/Ed4v1rkGN0Awux8+gJqJmp
c2kJhzAoQIAIGAW4nTGP9tl9QUHfwKh2KPA104vezk70ijvF7TrbTmhdfmULAuWK
Tbsg8sXAPhGmPh5KckM2Xe387iPh4ue2+2TGdWqwXygVdvIUIbcIMq6F+/mWlcmf
Gs5FVI7CTjaLmeyO4ho+FGmicmqH2hEkZW0a2ktDh4BU/MxYF6L7oayrVWDGp2IH
dzQAwUT2qJcFjElO8xUz3g==</CipherValue>
</CipherData>
</EncryptedKey>
</KeyInfo>
<CipherData>
<CipherValue>xrfPSA+BEI+8ca23RN34gtee5lOMx8Cn+ZGWyxitiktdZ1+XREH+57li
63VutCwp
s6ifbZgXIBsFdxPpMBUFlyTWAAO+NLooIwGoczXi14z62lHr7Ck6FA==</CipherValue
>
</CipherData>
</EncryptedData>
```

A-3) Documento XML a ser assinado

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="urn:envelope">
  <Data>
        Hello, World!
  </Data>
</Envelope>
```

A-4) Documento XML resultante da assinatura

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
XML Security Library example: Signed XML doc file (sign2 example).
-->
<Envelope xmlns="urn:envelope">
  <Data>
        Hello, World!
  </Data>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
<CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"/>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1"/>
<Reference>
<Transforms>
<Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>HjY8ilZAIEM2tBbPn5mYO1ieIX4=</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>SIaj/6KY3C1SmDXU2++Gm31U1xTadFp04WhBgfsJFbxrL+q7GKSKN
9kfQ+UpN9+i
D5fWmuavXEHe4Gw6RMaMEkq2URQo7F68+d5J/ajq8/l4n+xE6/reGScVwT6L4dEP
XXVJcAi2ZnQ3O7GTNvNGCPibL9mUcyCWBFZ92Uemtc/vJFCQ7ZyKMdMfACgxOwyN
T/9971oog241/2doudhonc0I/3mgPYWkZdX6yvr62mEjnG+oUZkhWYJ4ewZJ4hM4
JjbFqZO+OEzDRSbw3DkmuBA/mtlx+3t13SESfEub5hqoMdVmtth/eTb64dsPdl9r
3k1ACVX9f8aHfQQdJOmLFQ==</SignatureValue>
<KeyInfo>
<KeyName>rsakey.pem</KeyName>
</KeyInfo>
</Signature></Envelope>
```

A) Assinando um documento

```
/**
 * Assina um documento XML
 *
 *
 * Uso:
 *      sign <xml-doc> <pem-key>
 *
 * Examplo:
 *      ./sign sign-doc.xml rsakey.pem > sign-res.xml
 */
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#include <libxml/tree.h>
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>

#ifndef XMLSEC_NO_XSLT
#include <libxslt/xslt.h>
#endif /* XMLSEC_NO_XSLT */
```

```
#include <xmlsec/xmlsec.h>
#include <xmlsec/xmltree.h>
#include <xmlsec/xmldsig.h>
#include <xmlsec/templates.h>
#include <xmlsec/crypto.h>

int sign_file(const char* xml_file, const char* key_file);

int
main(int argc, char **argv) {
    assert(argv);

    if(argc != 3) {
        fprintf(stderr, "Error: wrong number of arguments.\n");
        fprintf(stderr, "Usage: %s <xml-file> <key-file>\n",
argv[0]);
        return(1);
    }

    /* Init libxml and libxslt libraries */
    xmlInitParser();
    LIBXML_TEST_VERSION
    xmlLoadExtDtdDefaultValue = XML_DETECT_IDS | XML_COMPLETE_ATTRS;
    xmlSubstituteEntitiesDefault(1);
#ifndef XMLSEC_NO_XSLT
    xmlIndentTreeOutput = 1;
#endif /* XMLSEC_NO_XSLT */

    /* Init xmlsec library */
    if(xmlSecInit() < 0) {
        fprintf(stderr, "Error: xmlsec initialization failed.\n");
        return(-1);
    }

    /* Check loaded library version */
    if(xmlSecCheckVersion() != 1) {
        fprintf(stderr, "Error: loaded xmlsec library version is not
compatible.\n");
        return(-1);
    }

    /* Load default crypto engine if we are supporting dynamic
     * loading for xmlsec-crypto libraries. Use the crypto library
     * name ("openssl", "nss", etc.) to load corresponding
     * xmlsec-crypto library.
     */
#ifdef XMLSEC_CRYPTO_DYNAMIC_LOADING
    if(xmlSecCryptoDLLoadLibrary(BAD_CAST XMLSEC_CRYPTO) < 0) {
        fprintf(stderr, "Error: unable to load default xmlsec-crypto
library. Make sure\n"
                        "that you have it installed and check shared
libraries path\n"
                        "(LD_LIBRARY_PATH) envornment variable.\n");
        return(-1);
    }
#endif /* XMLSEC_CRYPTO_DYNAMIC_LOADING */

    /* Init crypto library */
    if(xmlSecCryptoAppInit(NULL) < 0) {
        fprintf(stderr, "Error: crypto initialization failed.\n");
```

```
        return(-1);
    }

    /* Init xmlsec-crypto library */
    if(xmlSecCryptoInit() < 0) {
        fprintf(stderr, "Error: xmlsec-crypto initialization
failed.\n");
        return(-1);
    }

    if(sign_file(argv[1], argv[2]) < 0) {
        return(-1);
    }

    /* Shutdown xmlsec-crypto library */
    xmlSecCryptoShutdown();

    /* Shutdown crypto library */
    xmlSecCryptoAppShutdown();

    /* Shutdown xmlsec library */
    xmlSecShutdown();

    /* Shutdown libxslt/libxml */
#ifndef XMLSEC_NO_XSLT
    xsltCleanupGlobals();
#endif /* XMLSEC_NO_XSLT */
    xmlCleanupParser();

    return(0);
}

/**
 * sign_file:
 * @xml_file:          the XML file name.
 * @key_file:          the PEM private key file name.
 *
 * Signs the #xml_file using private key from #key_file and
dynamicaly
 * created enveloped signature template.
 *
 * Returns 0 on success or a negative value if an error occurs.
 */
int
sign_file(const char* xml_file, const char* key_file) {
    xmlDocPtr doc = NULL;
    xmlNodePtr signNode = NULL;
    xmlNodePtr refNode = NULL;
    xmlNodePtr keyInfoNode = NULL;
    xmlSecDSigCtxPtr dsigCtx = NULL;
    int res = -1;

    assert(xml_file);
    assert(key_file);

    /* load doc file */
    doc = xmlParseFile(xml_file);
    if ((doc == NULL) || (xmlDocGetRootElement(doc) == NULL)){
        fprintf(stderr, "Error: unable to parse file \"%s\"\n",
xml_file);
        goto done;
```

```
    }

    /* create signature template for RSA-SHA1 enveloped signature */
    signNode = xmlSecTmplSignatureCreate(doc,
xmlSecTransformExclC14NId,
                                         xmlSecTransformRsaSha1Id,
NULL);
    if(signNode == NULL) {
        fprintf(stderr, "Error: failed to create signature
template\n");
        goto done;
    }

    /* add <dsig:Signature/> node to the doc */
    xmlAddChild(xmlDocGetRootElement(doc), signNode);

    /* add reference */
    refNode = xmlSecTmplSignatureAddReference(signNode,
xmlSecTransformSha1Id,
                                      NULL, NULL, NULL);
    if(refNode == NULL) {
        fprintf(stderr, "Error: failed to add reference to signature
template\n");
        goto done;
    }

    /* add enveloped transform */
    if(xmlSecTmplReferenceAddTransform(refNode,
xmlSecTransformEnvelopedId) == NULL) {
        fprintf(stderr, "Error: failed to add enveloped transform to
reference\n");
        goto done;
    }

    /* add <dsig:KeyInfo/> and <dsig:KeyName/> nodes to put key name
in the signed document */
    keyInfoNode = xmlSecTmplSignatureEnsureKeyInfo(signNode, NULL);
    if(keyInfoNode == NULL) {
        fprintf(stderr, "Error: failed to add key info\n");
        goto done;
    }

    if(xmlSecTmplKeyInfoAddKeyName(keyInfoNode, NULL) == NULL) {
        fprintf(stderr, "Error: failed to add key name\n");
        goto done;
    }

    /* create signature context, we don't need keys manager in this
example */
    dsigCtx = xmlSecDSigCtxCreate(NULL);
    if(dsigCtx == NULL) {
        fprintf(stderr,"Error: failed to create signature
context\n");
        goto done;
    }

    /* load private key, assuming that there is not password */
    dsigCtx->signKey = xmlSecCryptoAppKeyLoad(key_file,
xmlSecKeyDataFormatPem, NULL, NULL, NULL);
    if(dsigCtx->signKey == NULL) {
```

```
        fprintf(stderr,"Error: failed to load private pem key from
\"%s\"\n", key_file);
        goto done;
    }

    /* set key name to the file name, this is just an example! */
    if(xmlSecKeySetName(dsigCtx->signKey, key_file) < 0) {
        fprintf(stderr,"Error: failed to set key name for key from
\"%s\"\n", key_file);
        goto done;
    }

    /* sign the template */
    if(xmlSecDSigCtxSign(dsigCtx, signNode) < 0) {
        fprintf(stderr,"Error: signature failed\n");
        goto done;
    }

    /* print signed document to stdout */
    xmlDocDump(stdout, doc);

    /* success */
    res = 0;

done:
    /* cleanup */
    if(dsigCtx != NULL) {
        xmlSecDSigCtxDestroy(dsigCtx);
    }

    if(doc != NULL) {
        xmlFreeDoc(doc);
    }
    return(res);
}
```

B) Criptografando um documento

```
/**
 * Criptografa um documento XML com uma chave de sessão DES *
 * Uso:
 *      ./encrypt <xml-doc> <rsa-pem-key-file>
 *
 * Examplo:
 *      ./encrypt encrypt-doc.xml rsakey.pem > encrypt-res.xml
 *
*/
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#include <libxml/tree.h>
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>

#ifndef XMLSEC_NO_XSLT
#include <libxslt/xslt.h>
#endif /* XMLSEC_NO_XSLT */

#include <xmlsec/xmlsec.h>
```

```
#include <xmlsec/xmltree.h>
#include <xmlsec/xmlenc.h>
#include <xmlsec/templates.h>
#include <xmlsec/crypto.h>

xmlSecKeysMngrPtr load_rsa_keys(char* key_file);
int encrypt_file(xmlSecKeysMngrPtr mngr, const char* xml_file, const
char* key_name);

int
main(int argc, char **argv) {
    xmlSecKeysMngrPtr mngr;

    assert(argv);

    if(argc != 3) {
        fprintf(stderr, "Error: wrong number of arguments.\n");
        fprintf(stderr, "Usage: %s <xml-file> <key-file>\n",
argv[0]);
        return(1);
    }

    /* Init libxml and libxslt libraries */
    xmlInitParser();
    LIBXML_TEST_VERSION
    xmlLoadExtDtdDefaultValue = XML_DETECT_IDS | XML_COMPLETE_ATTRS;
    xmlSubstituteEntitiesDefault(1);
#ifndef XMLSEC_NO_XSLT
    xmlIndentTreeOutput = 1;
#endif /* XMLSEC_NO_XSLT */

    /* Init xmlsec library */
    if(xmlSecInit() < 0) {
        fprintf(stderr, "Error: xmlsec initialization failed.\n");
        return(-1);
    }

    /* Check loaded library version */
    if(xmlSecCheckVersion() != 1) {
        fprintf(stderr, "Error: loaded xmlsec library version is not
compatible.\n");
        return(-1);
    }

    /* Load default crypto engine if we are supporting dynamic
     * loading for xmlsec-crypto libraries. Use the crypto library
     * name ("openssl", "nss", etc.) to load corresponding
     * xmlsec-crypto library.
     */
#ifdef XMLSEC_CRYPTO_DYNAMIC_LOADING
    if(xmlSecCryptoDLLoadLibrary(BAD_CAST XMLSEC_CRYPTO) < 0) {
        fprintf(stderr, "Error: unable to load default xmlsec-crypto
library. Make sure\n"
                        "that you have it installed and check shared
libraries path\n"
                        "(LD_LIBRARY_PATH) envornment variable.\n");
        return(-1);
    }
#endif /* XMLSEC_CRYPTO_DYNAMIC_LOADING */

    /* Init crypto library */
```

```
    if(xmlSecCryptoAppInit(NULL) < 0) {
        fprintf(stderr, "Error: crypto initialization failed.\n");
        return(-1);
    }

    /* Init xmlsec-crypto library */
    if(xmlSecCryptoInit() < 0) {
        fprintf(stderr, "Error: xmlsec-crypto initialization
failed.\n");
        return(-1);
    }

    /* create keys manager and load keys */
    mngr = load_rsa_keys(argv[2]);
    if(mngr == NULL) {
        return(-1);
    }

    /* we use key filename as key name here */
    if(encrypt_file(mngr, argv[1], argv[2]) < 0) {
        xmlSecKeysMngrDestroy(mngr);
        return(-1);
    }

    /* destroy keys manager */
    xmlSecKeysMngrDestroy(mngr);

    /* Shutdown xmlsec-crypto library */
    xmlSecCryptoShutdown();

    /* Shutdown crypto library */
    xmlSecCryptoAppShutdown();

    /* Shutdown xmlsec library */
    xmlSecShutdown();

    /* Shutdown libxslt/libxml */
#ifndef XMLSEC_NO_XSLT
    xsltCleanupGlobals();
#endif /* XMLSEC_NO_XSLT */
    xmlCleanupParser();

    return(0);
}

/**
 * load_rsa_keys:
 * @key_file:          the key filename.
 *
 * Creates simple keys manager and load RSA key from #key_file in it.
 * The caller is responsible for destroing returned keys manager
using
 * @xmlSecKeysMngrDestroy.
 *
 * Returns the pointer to newly created keys manager or NULL if an
error
 * occurs.
 */
xmlSecKeysMngrPtr
load_rsa_keys(char* key_file) {
    xmlSecKeysMngrPtr mngr;
```

```
    xmlSecKeyPtr key;

    assert(key_file);

    /* create and initialize keys manager, we use a simple list based
     * keys manager, implement your own xmlSecKeysStore klass if you
need
     * something more sophisticated
     */
    mngr = xmlSecKeysMngrCreate();
    if(mngr == NULL) {
        fprintf(stderr, "Error: failed to create keys manager.\n");
        return(NULL);
    }
    if(xmlSecCryptoAppDefaultKeysMngrInit(mngr) < 0) {
        fprintf(stderr, "Error: failed to initialize keys
manager.\n");
        xmlSecKeysMngrDestroy(mngr);
        return(NULL);
    }

    /* load private RSA key */
    key = xmlSecCryptoAppKeyLoad(key_file, xmlSecKeyDataFormatPem,
NULL, NULL, NULL);
    if(key == NULL) {
        fprintf(stderr,"Error: failed to load rsa key from file
\"%s\"\n", key_file);
        xmlSecKeysMngrDestroy(mngr);
        return(NULL);
    }

    /* set key name to the file name, this is just an example! */
    if(xmlSecKeySetName(key, BAD_CAST key_file) < 0) {
        fprintf(stderr,"Error: failed to set key name for key from
\"%s\"\n", key_file);
        xmlSecKeyDestroy(key);
        xmlSecKeysMngrDestroy(mngr);
        return(NULL);
    }

    /* add key to keys manager, from now on keys manager is
responsible
     * for destroying key
     */
    if(xmlSecCryptoAppDefaultKeysMngrAdoptKey(mngr, key) < 0) {
        fprintf(stderr,"Error: failed to add key from \"%s\" to keys
manager\n", key_file);
        xmlSecKeyDestroy(key);
        xmlSecKeysMngrDestroy(mngr);
        return(NULL);
    }

    return(mngr);
}

/**
 * encrypt_file:
 * @mngr:              the pointer to keys manager.
 * @xml_file:          the encryption template file name.
 * @key_name:          the RSA key name.
 *
```

```
 * Encrypts #xml_file using a dynamicaly created template, a session
DES key
 * and an RSA key from keys manager.
 *
 * Returns 0 on success or a negative value if an error occurs.
 */
int
encrypt_file(xmlSecKeysMngrPtr mngr, const char* xml_file, const
char* key_name) {
    xmlDocPtr doc = NULL;
    xmlNodePtr encDataNode = NULL;
    xmlNodePtr keyInfoNode = NULL;
    xmlNodePtr encKeyNode = NULL;
    xmlNodePtr keyInfoNode2 = NULL;
    xmlSecEncCtxPtr encCtx = NULL;
    int res = -1;

    assert(mngr);
    assert(xml_file);
    assert(key_name);

    /* load template */
    doc = xmlParseFile(xml_file);
    if ((doc == NULL) || (xmlDocGetRootElement(doc) == NULL)){
        fprintf(stderr, "Error: unable to parse file \"%s\"\n",
xml_file);
        goto done;
    }

    /* create encryption template to encrypt XML file and replace
     * its content with encryption result */
    encDataNode = xmlSecTmplEncDataCreate(doc,
xmlSecTransformDes3CbcId,
                                NULL, xmlSecTypeEncElement, NULL,
NULL);
    if(encDataNode == NULL) {
        fprintf(stderr, "Error: failed to create encryption
template\n");
        goto done;
    }

    /* we want to put encrypted data in the <enc:CipherValue/> node
*/
    if(xmlSecTmplEncDataEnsureCipherValue(encDataNode) == NULL) {
        fprintf(stderr, "Error: failed to add CipherValue node\n");
        goto done;
    }

    /* add <dsig:KeyInfo/> */
    keyInfoNode = xmlSecTmplEncDataEnsureKeyInfo(encDataNode, NULL);
    if(keyInfoNode == NULL) {
        fprintf(stderr, "Error: failed to add key info\n");
        goto done;
    }

    /* add <enc:EncryptedKey/> to store the encrypted session key */
    encKeyNode = xmlSecTmplKeyInfoAddEncryptedKey(keyInfoNode,
                                    xmlSecTransformRsaPkcs1Id,
                                    NULL, NULL, NULL);
    if(encKeyNode == NULL) {
        fprintf(stderr, "Error: failed to add key info\n");
```

```
        goto done;
    }

    /* we want to put encrypted key in the <enc:CipherValue/> node */
    if(xmlSecTmplEncDataEnsureCipherValue(encKeyNode) == NULL) {
        fprintf(stderr, "Error: failed to add CipherValue node\n");
        goto done;
    }

    /* add <dsig:KeyInfo/> and <dsig:KeyName/> nodes to
<enc:EncryptedKey/> */
    keyInfoNode2 = xmlSecTmplEncDataEnsureKeyInfo(encKeyNode, NULL);
    if(keyInfoNode2 == NULL) {
        fprintf(stderr, "Error: failed to add key info\n");
        goto done;
    }

    /* set key name so we can lookup key when needed */
    if(xmlSecTmplKeyInfoAddKeyName(keyInfoNode2, key_name) == NULL) {
        fprintf(stderr, "Error: failed to add key name\n");
        goto done;
    }

    /* create encryption context */
    encCtx = xmlSecEncCtxCreate(mngr);
    if(encCtx == NULL) {
        fprintf(stderr,"Error: failed to create encryption
context\n");
        goto done;
    }

    /* generate a Triple DES key */
    encCtx->encKey = xmlSecKeyGenerate(xmlSecKeyDataDesId, 192,
xmlSecKeyDataTypeSession);
    if(encCtx->encKey == NULL) {
        fprintf(stderr,"Error: failed to generate session des
key\n");
        goto done;
    }

    /* encrypt the data */
    if(xmlSecEncCtxXmlEncrypt(encCtx, encDataNode,
xmlDocGetRootElement(doc)) < 0) {
        fprintf(stderr,"Error: encryption failed\n");
        goto done;
    }

    /* we template is inserted in the doc */
    encDataNode = NULL;

    /* print encrypted data with document to stdout */
    xmlDocDump(stdout, doc);

    /* success */
    res = 0;

done:

    /* cleanup */
    if(encCtx != NULL) {
        xmlSecEncCtxDestroy(encCtx);
```

```
    }

    if(encDataNode != NULL) {
        xmlFreeNode(encDataNode);
    }

    if(doc != NULL) {
        xmlFreeDoc(doc);
    }
    return(res);
}
```

C) Verificando um documento

```
/**
 * Verifica validade da assinatura
 *
 * Uso:
 *      ./verify sign-res.xml rsapub.pem
 *
 * Retorna 0 caso sucesso
*/
int
verify_file(const char* xml_file, const char* key_file) {
    xmlDocPtr doc = NULL;
    xmlNodePtr node = NULL;
    xmlSecDSigCtxPtr dsigCtx = NULL;
    int res = -1;

    assert(xml_file);
    assert(key_file);

    /* load file */
    doc = xmlParseFile(xml_file);
    if ((doc == NULL) || (xmlDocGetRootElement(doc) == NULL)){
        fprintf(stderr, "Error: unable to parse file \"%s\"\n",
xml_file);
        goto done;
    }

    /* find start node */
    node = xmlSecFindNode(xmlDocGetRootElement(doc),
xmlSecNodeSignature, xmlSecDSigNs);
    if(node == NULL) {
        fprintf(stderr, "Error: start node not found in \"%s\"\n",
xml_file);
        goto done;
    }

    /* create signature context, we don't need keys manager in this
example */
    dsigCtx = xmlSecDSigCtxCreate(NULL);
    if(dsigCtx == NULL) {
        fprintf(stderr,"Error: failed to create signature
context\n");
        goto done;
    }

    /* load public key */
```

```
    dsigCtx->signKey =
xmlSecCryptoAppKeyLoad(key_file,xmlSecKeyDataFormatPem, NULL, NULL,
NULL);
    if(dsigCtx->signKey == NULL) {
        fprintf(stderr,"Error: failed to load public pem key from
\"%s\"\n", key_file);
        goto done;
    }

    /* set key name to the file name, this is just an example! */
    if(xmlSecKeySetName(dsigCtx->signKey, key_file) < 0) {
        fprintf(stderr,"Error: failed to set key name for key from
\"%s\"\n", key_file);
        goto done;
    }

    /* Verify signature */
    if(xmlSecDSigCtxVerify(dsigCtx, node) < 0) {
        fprintf(stderr,"Error: signature verify\n");
        goto done;
    }

    /* print verification result to stdout */
    if(dsigCtx->status == xmlSecDSigStatusSucceeded) {
        fprintf(stdout, "Signature is OK\n");
    } else {
        fprintf(stdout, "Signature is INVALID\n");
    }

    /* success */
    res = 0;

done:
    /* cleanup */
    if(dsigCtx != NULL) {
        xmlSecDSigCtxDestroy(dsigCtx);
    }

    if(doc != NULL) {
        xmlFreeDoc(doc);
    }
    return(res);
}
```

D) Decriptografando um documento

```
/* Decriptografa um documento XML
 *
 * Uso:
 *      ./decrypt <xml-enc>
 *
 * Examplo:
 *      ./decrypt encrypt-res.xml
 *
*/
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <assert.h>
```

```
#include <libxml/tree.h>
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>

#ifndef XMLSEC_NO_XSLT
#include <libxslt/xslt.h>
#endif /* XMLSEC_NO_XSLT */

#include <xmlsec/xmlsec.h>
#include <xmlsec/xmltree.h>
#include <xmlsec/xmlenc.h>
#include <xmlsec/crypto.h>

xmlSecKeyStoreId  files_keys_store_get_klass(void);
xmlSecKeysMngrPtr create_files_keys_mngr(void);
int decrypt_file(xmlSecKeysMngrPtr mngr, const char* enc_file);

int
main(int argc, char **argv) {
    xmlSecKeysMngrPtr mngr;

    assert(argv);

    if(argc != 2) {
        fprintf(stderr, "Error: wrong number of arguments.\n");
        fprintf(stderr, "Usage: %s <enc-file>\n", argv[0]);
        return(1);
    }

    /* Init libxml and libxslt libraries */
    xmlInitParser();
    LIBXML_TEST_VERSION
    xmlLoadExtDtdDefaultValue = XML_DETECT_IDS | XML_COMPLETE_ATTRS;
    xmlSubstituteEntitiesDefault(1);
#ifndef XMLSEC_NO_XSLT
    xmlIndentTreeOutput = 1;
#endif /* XMLSEC_NO_XSLT */

    /* Init xmlsec library */
    if(xmlSecInit() < 0) {
        fprintf(stderr, "Error: xmlsec initialization failed.\n");
        return(-1);
    }

    /* Check loaded library version */
    if(xmlSecCheckVersion() != 1) {
        fprintf(stderr, "Error: loaded xmlsec library version is not
compatible.\n");
        return(-1);
    }

    /* Load default crypto engine if we are supporting dynamic
     * loading for xmlsec-crypto libraries. Use the crypto library
     * name ("openssl", "nss", etc.) to load corresponding
     * xmlsec-crypto library.
     */
#ifdef XMLSEC_CRYPTO_DYNAMIC_LOADING
    if(xmlSecCryptoDLLoadLibrary(BAD_CAST XMLSEC_CRYPTO) < 0) {
        fprintf(stderr, "Error: unable to load default xmlsec-crypto
library. Make sure\n"
```

```
                        "that you have it installed and check shared
libraries path\n"
                        "(LD_LIBRARY_PATH) envornment variable.\n");
        return(-1);
    }
#endif /* XMLSEC_CRYPTO_DYNAMIC_LOADING */

    /* Init crypto library */
    if(xmlSecCryptoAppInit(NULL) < 0) {
        fprintf(stderr, "Error: crypto initialization failed.\n");
        return(-1);
    }

    /* Init xmlsec-crypto library */
    if(xmlSecCryptoInit() < 0) {
        fprintf(stderr, "Error: xmlsec-crypto initialization
failed.\n");
        return(-1);
    }

    /* create keys manager and load keys */
    mngr = create_files_keys_mngr();
    if(mngr == NULL) {
        return(-1);
    }

    if(decrypt_file(mngr, argv[1]) < 0) {
        xmlSecKeysMngrDestroy(mngr);
        return(-1);
    }

    /* destroy keys manager */
    xmlSecKeysMngrDestroy(mngr);

    /* Shutdown xmlsec-crypto library */
    xmlSecCryptoShutdown();

    /* Shutdown crypto library */
    xmlSecCryptoAppShutdown();

    /* Shutdown xmlsec library */
    xmlSecShutdown();

    /* Shutdown libxslt/libxml */
#ifndef XMLSEC_NO_XSLT
    xsltCleanupGlobals();
#endif /* XMLSEC_NO_XSLT */
    xmlCleanupParser();

    return(0);
}

/**
 * decrypt_file:
 * @mngr:               the pointer to keys manager.
 * @enc_file:           the encrypted XML  file name.
 *
 * Decrypts the XML file #enc_file using DES key from #key_file and
 * prints results to stdout.
 *
 * Returns 0 on success or a negative value if an error occurs.
```

```
 */
int
decrypt_file(xmlSecKeysMngrPtr mngr, const char* enc_file) {
    xmlDocPtr doc = NULL;
    xmlNodePtr node = NULL;
    xmlSecEncCtxPtr encCtx = NULL;
    int res = -1;

    assert(mngr);
    assert(enc_file);

    /* load template */
    doc = xmlParseFile(enc_file);
    if ((doc == NULL) || (xmlDocGetRootElement(doc) == NULL)){
        fprintf(stderr, "Error: unable to parse file \"%s\"\n",
enc_file);
        goto done;
    }

    /* find start node */
    node = xmlSecFindNode(xmlDocGetRootElement(doc),
xmlSecNodeEncryptedData, xmlSecEncNs);
    if(node == NULL) {
        fprintf(stderr, "Error: start node not found in \"%s\"\n",
enc_file);
        goto done;
    }

    /* create encryption context */
    encCtx = xmlSecEncCtxCreate(mngr);
    if(encCtx == NULL) {
        fprintf(stderr,"Error: failed to create encryption
context\n");
        goto done;
    }

    /* decrypt the data */
    if((xmlSecEncCtxDecrypt(encCtx, node) < 0) || (encCtx->result ==
NULL)) {
        fprintf(stderr,"Error: decryption failed\n");
        goto done;
    }

    /* print decrypted data to stdout */
    if(encCtx->resultReplaced != 0) {
        fprintf(stdout, "Decrypted XML data:\n");
        xmlDocDump(stdout, doc);
    } else {
        fprintf(stdout, "Decrypted binary data (%d bytes):\n",
xmlSecBufferGetSize(encCtx->result));
        if(xmlSecBufferGetData(encCtx->result) != NULL) {
            fwrite(xmlSecBufferGetData(encCtx->result),
                  1,
                  xmlSecBufferGetSize(encCtx->result),
                  stdout);
        }
    }
    fprintf(stdout, "\n");

    /* success */
    res = 0;
```

```
done:
    /* cleanup */
    if(encCtx != NULL) {
        xmlSecEncCtxDestroy(encCtx);
    }

    if(doc != NULL) {
        xmlFreeDoc(doc);
    }
    return(res);
}

/**
 * create_files_keys_mngr:
 *
 * Creates a files based keys manager: we assume that key name is
 * the key file name,
 *
 * Returns pointer to newly created keys manager or NULL if an error
occurs.
 */
xmlSecKeysMngrPtr
create_files_keys_mngr(void) {
    xmlSecKeyStorePtr keysStore;
    xmlSecKeysMngrPtr mngr;

    /* create files based keys store */
    keysStore = xmlSecKeyStoreCreate(files_keys_store_get_klass());
    if(keysStore == NULL) {
        fprintf(stderr, "Error: failed to create keys store.\n");
        return(NULL);
    }

    /* create keys manager */
    mngr = xmlSecKeysMngrCreate();
    if(mngr == NULL) {
        fprintf(stderr, "Error: failed to create keys manager.\n");
        xmlSecKeyStoreDestroy(keysStore);
        return(NULL);
    }

    /* add store to keys manager, from now on keys manager destroys
the store if needed */
    if(xmlSecKeysMngrAdoptKeysStore(mngr, keysStore) < 0) {
        fprintf(stderr, "Error: failed to add keys store to keys
manager.\n");
        xmlSecKeyStoreDestroy(keysStore);
        xmlSecKeysMngrDestroy(mngr);
        return(NULL);
    }

    /* initialize crypto library specific data in keys manager */
    if(xmlSecCryptoKeysMngrInit(mngr) < 0) {
        fprintf(stderr, "Error: failed to initialize crypto data in
keys manager.\n");
        xmlSecKeysMngrDestroy(mngr);
        return(NULL);
    }

    /* set the get key callback */
```

```
    mngr->getKey = xmlSecKeysMngrGetKey;
    return(mngr);
}

/**************************************************************************
********
 *
 * Files Keys Store: we assume that key's name (content of the
 * <dsig:KeyName/> element is a name of the file with a key (in the
 * current folder).
 * Attention: this probably not a good solution for high traffic
systems.
 *

**************************************************************************
******/
static xmlSecKeyPtr           files_keys_store_find_key
        (xmlSecKeyStorePtr store,
                                                                  const
xmlChar* name,

xmlSecKeyInfoCtxPtr keyInfoCtx);
static xmlSecKeyStoreKlass files_keys_store_klass = {
    sizeof(xmlSecKeyStoreKlass),
    sizeof(xmlSecKeyStore),
    BAD_CAST "files-based-keys-store",/* const xmlChar* name; */
    NULL,                             /*
xmlSecKeyStoreInitializeMethod initialize; */
    NULL,                             /* xmlSecKeyStoreFinalizeMethod
finalize; */
    files_keys_store_find_key,        /* xmlSecKeyStoreFindKeyMethod
findKey; */

    /* reserved for the future */
    NULL,                             /* void* reserved0; */
    NULL,                             /* void* reserved1; */
};

/**
 * files_keys_store_get_klass:
 *
 * The files based keys store klass: we assume that key name is the
 * key file name,
 *
 * Returns files based keys store klass.
 */
xmlSecKeyStoreId
files_keys_store_get_klass(void) {
    return(&files_keys_store_klass);
}

/**
 * files_keys_store_find_key:
 * @store:             the pointer to simple keys store.
 * @name:              the desired key name.
 * @keyInfoCtx:        the pointer to <dsig:KeyInfo/> node processing
context.
 *
 * Lookups key in the @store. The caller is responsible for
destroying
 * returned key with #xmlSecKeyDestroy function.
```

```
 *
 * Returns pointer to key or NULL if key not found or an error
occurs.
 */
static xmlSecKeyPtr
files_keys_store_find_key(xmlSecKeyStorePtr store, const xmlChar*
name, xmlSecKeyInfoCtxPtr keyInfoCtx) {
    xmlSecKeyPtr key;
    const xmlChar* p;

    assert(store);
    assert(keyInfoCtx);

    /* it's possible to do not have the key name or desired key type
     * but we could do nothing in this case */
    if((name == NULL) || (keyInfoCtx->keyReq.keyId ==
xmlSecKeyDataIdUnknown)){
        return(NULL);
    }

    /* we don't want to open files in a folder other than "current";
     * to prevent it limit the characters in the key name to
alpha/digit,
     * '.', '-' or '_'.
     */
    for(p = name; (*p) != '\0'; ++p) {
        if(!isalnum((*p)) && ((*p) != '.') && ((*p) != '-') && ((*p)
!= '_')) {
            return(NULL);
        }
    }

    if((keyInfoCtx->keyReq.keyId == xmlSecKeyDataDsaId) ||
(keyInfoCtx->keyReq.keyId == xmlSecKeyDataRsaId)) {
        /* load key from a pem file, if key is not found then it's an
error (is it?) */
        key = xmlSecCryptoAppKeyLoad(name, xmlSecKeyDataFormatPem,
NULL, NULL, NULL);
        if(key == NULL) {
            fprintf(stderr,"Error: failed to load public pem key from
\"%s\"\n", name);
            return(NULL);
        }
    } else {
        /* otherwise it's a binary key, if key is not found then it's
an error (is it?) */
        key = xmlSecKeyReadBinaryFile(keyInfoCtx->keyReq.keyId,
name);
        if(key == NULL) {
            fprintf(stderr,"Error: failed to load key from binary
file \"%s\"\n", name);
            return(NULL);
        }
    }

    /* set key name */
    if(xmlSecKeySetName(key, name) < 0) {
        fprintf(stderr,"Error: failed to set key name for key from
\"%s\"\n", name);
        xmlSecKeyDestroy(key);
        return(NULL);
```

```
    }

    return(key);
}
```