Die Applikationsschnittstelle Security-Layer zur österreichischen Bürgerkarte		Konvention 1.2.3 Empfehlung
Kurzbeschreibung	Das vorliegende Dokument spezifiziert die Schnittstelle zwischen *Applikation* und *Bürgerkarten-Umgebung*.	
Autoren:	Arno Hollosi Gregor Karlinger Thomas Rössler Martin Centner et al.	Projektteam/Arbeitsgruppe AG Bürgerkarte
Datum:	20.2.2008	

Inhaltsverzeichnis

1. Allgemeines

Dieses Dokument legt die Schnittstelle *Security-Layer* fest. Das ist jene Schnittstelle, über die eine *Applikation* auf Funktionen der *Bürgerkarte* zugreift, um beispielsweise eine elektronische Signatur zu erstellen oder vom Datenspeicher der *Bürgerkarten-Umgebung* zu lesen.

Für die Festlegung, welche der in diesem Dokument beschriebenen Schnittstellenbefehle von einer *Bürgerkarten-Umgebung* jedenfalls zur Verfügung gestellt werden MÜSSEN, siehe Minimale Umsetzung des Security-Layers.

Für Vorgaben an die *Benutzer-Schnittstelle* bei der Ausführung eines in diesem Dokument beschriebenen Schnittstellenbefehls siehe Anforderungen an die Benutzer-Schnittstelle.

1.1. Protokollelemente

Das Protokoll besteht aus einfachen Anfrage/Antwort-Mustern. Die *Applikation* schickt eine in XML kodierte Anfrage an die *Bürgerkarten-Umgebung*. Diese schickt eine entsprechende in XML kodierte Antwort zurück an die *Applikation*.

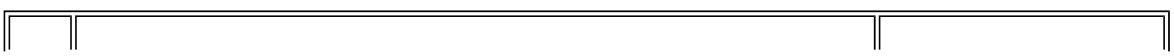
Die einzelnen Protokoll-Elemente für diese XML-Schnittstellenbefehle sind als [XML-Schema] in Core-1.2.xsd spezifiziert. Zusammen mit der vorliegenden Schnittstellenspezifikation bildet dieses XML-Schema die normative Quelle für die Protokoll-Elemente.

1.2. Namenskonventionen

Zur besseren Lesbarkeit wurde in diesem Dokument auf geschlechtsneutrale Formulierungen verzichtet. Die verwendeten Formulierungen richten sich jedoch ausdrücklich an beide Geschlechter.

Die einzelnen Protokoll-Elemente wurden mit aussagekräftigen Namen belegt, wobei Abkürzungen so weit wie möglich vermieden worden sind. Anfragen enden stets mit dem Suffix `Request`, die korrespondierenden Antworten enden stets mit dem Suffix `Response`.

Folgende Namenraum-Präfixe werden in dieser Spezifikation zur Kennzeichnung der Namenräume von XML-Elementen verwendet:



dsig	http://www.w3.org/2000/09/xmldsig#	Elemente aus [XMLDSIG]
xenc	http://www.w3.org/2001/04/xmlenc#	Elemente aus [XMLEnc]
etsi	http://uri.etsi.org/01903/v1.2.2#	Elemente aus [ETSIXML]
sl	http://www.buergerkarte.at/namespaces/securitylayer/1.2#	Elemente dieser Spezifikation

1.3. Schlüsselwörter

Dieses Dokument verwendet die Schlüsselwörter MUSS, DARF NICHT, ERFORDERLICH, SOLLTE, SOLLTE NICHT, EMPFOHLEN, DARF, und OPTIONAL zur Kategorisierung der Anforderungen. Diese Schlüsselwörter sind analog zu ihren englischsprachigen Entsprechungen MUST, MUST NOT, REQUIRED, SHOULD, SHOULD NOT, RECOMMENDED, MAY, und OPTIONAL zu handhaben, deren Interpretation in [Keywords] festgelegt ist.

2. Signaturerstellung

Die Schnittstelle *Security-Layer* unterstützt zwei mögliche Formate für die Erzeugung einer elektronischen Signatur: [CMS] und [XMLDSIG].

2.1. Signatur nach CMS

2.1.1. Anfrage

Mit einer Signatur nach [CMS] kann genau ein Datenobjekt signiert werden.

2.1.1.1. Struktur der Signatur

Zunächst muss im Attribut `Structure` der Signaturanfrage (`sl:CreateCMSSignatureRequest`) angegeben werden, ob das nachfolgend spezifizierte Datenobjekt in die Signaturstruktur eingebunden werden soll (Wert `"enveloping"`), oder nicht (Wert `"detached"`).

2.1.1.2. Bezeichnung des Signaturschlüssels

Weiters muss in der Signaturanfrage der Bezeichner des für die Signaturerstellung zu verwendenden Schlüssels (`sl:KeyboxIdentifier`) angegeben werden. Bezeichner aller verfügbaren Schlüssel können mit Hilfe des Befehls `sl:GetPropertiesRequest` von der *Bürgerkarten-Umgebung* abgefragt werden.

2.1.1.3. Informationen zum Datenobjekt

Schließlich enthält die Signaturanfrage einen Behälter `sl:DataObject`, der das zu signierende Datenobjekt (`sl:Content`), sowie Metainformationen (`sl:MetaInfo`) für die Anfertigung der Signatur sowie für die gegebenenfalls notwendige Anzeige in der *Bürgerkarten-Umgebung* enthält.

Jedenfalls an Metainformation spezifiziert werden muss der Mime-Type (siehe [MIME]) des zu signierenden Datenobjekts. Weiters DARF eine verbale Beschreibung des Datenobjekts angegeben werden (`sl:Description`). Schließlich DÜRFEN noch weitere beliebige Elemente zu diesen Metainformationen hinzugefügt werden.

Die Angabe des zu signierenden Datenobjekts kann auf zwei Arten erfolgen: Entweder enthält das Element `sl:Content` die base64-kodierten Daten, oder das Element `sl:Content` ist leer, hat aber sein Attribut `Reference` gesetzt. Die *Bürgerkarten-Umgebung* MUSS im letzten Fall versuchen, die in diesem Attribut angegebene URI aufzulösen, um so die zu signierenden Daten zu erhalten.

2.1.2. Antwort

Die Antwort besteht aus dem Element `sl:CMSSignature`, welches die erzeugte Signatur nach [CMS] in base64-kodierter Form enthält. Für detailierte Vorgaben für die zu erzeugende Signatur hinsichtlich Digest-Algorithmen, Signatur-Algorithen und Schlüsselinformationen siehe Minimale Umsetzung des Security-Layers.

2.1.2.1. Signierte Metainformationen

In die [CMS]-Signatur MUSS ein signiertes Signaturattribut `ContentHints` nach [ESS-S/MIME], Abschnitt 2.9 aufgenommen werden. Zur Anfertigung dieses Signaturattributs sind jene Metainformationen (`sl:MetaInfo`) zu verwenden, die in der Anfrage im Behälter für das Datenobjekt (`sl:DataObject`) angegeben wurden.

Für das Element `sl:MimeType` aus `sl:MetaInfo` MUSS ein erstes Feld `contentDescription` in `ContentHints` verwendet werden; ist das Element `sl:Description` aus `sl:MetaInfo` vorhanden, MUSS es in einem weiteren Feld `contentDescription` in `ContentHints` codiert werden. Das Feld `contentType` MUSS jedenfalls mit dem Wert des Object Identifiers für `id-data` (siehe [CMS], Abschnitt 4) belegt werden.

2.1.2.2. Signierte Zertifikatsreferenz

Weiters MUSS ein signiertes Signaturattribut `OtherSigningCertificate` nach [ETSICMS] in die [CMS]-Signatur aufgenommen werden, mit dem das für die Verifikation der Signatur zu verwendende Signatorzertifikat eindeutig identifiziert wird.

2.1.2.3. Zeitpunkt der Signaturerstellung

Schließlich MUSS ein Signaturattribut `SigningTime` nach [ETSICMS] in die [CMS]-Signatur aufgenommen werden, das den vom Signator behaupteten Zeitpunkt der Signaturerstellung enthält.

> **Anmerkung**
>
> Für Beispiele zu diesem Befehl siehe Tutorium; für Anforderungen an die Benutzerschnittstelle siehe Abschnitt 3.1, „Signaturerstellung“ in Die österreichische Bürgerkarte - Anforderungen an die Benutzer-Schnittstelle.

2.2. Signatur nach XMLDSIG

2.2.1. Anfrage

Die Signatur nach [XMLDSIG] erlaubt im Gegensatz zur Signatur nach [CMS] auch die Signierung mehrerer Datenobjekte mittels einer einzigen Signatur.

2.2.1.1. Bezeichnung des Signaturschlüssels

In der Signaturanfrage muss der Bezeichner des für die Signaturerstellung zu verwendenden Schlüssels (`sl:KeyboxIdentifier`) angegeben werden. Bezeichner aller verfügbaren Schlüssel können mit Hilfe des Befehls `sl:GetPropertiesRequest` von der *Bürgerkarten-Umgebung* abgefragt werden.

2.2.1.2. Informationen zum Datenobjekt

In der Folge enthält die Signaturanfrage für jedes Datenobjekt, das von der Signatur unterschrieben werden soll, einen Behälter (`sl:DataObjectInfo`) , der Informationen für die Anfertigung der Signatur sowie für die gegebenenfalls notwendige Anzeige in der *Bürgerkarten-Umgebung* enthält. `sl:DataObjectInfo` ist mit einem Attribut `Structure` ausgestattet, das angibt, ob das im Behälter spezifizierte Datenobjekt in die Signaturstruktur eingebunden werden soll (Wert `"enveloping"`), oder ob die Signatur dieses Datenobjekt nur referenzieren soll (Wert `"detached"`).

Datenobjekt

Ein solcher Behälter besteht zunächst aus Informationen zu jenem Datenobjekt, das gegebenenfalls transformiert und nachfolgend signiert wird (`sl:DataObject`). In Abhängigkeit des oben besprochenen Attributs `Structure` sind für Inhalt von `sl:DataObject` sowie sein Attribut `Reference` folgende Kombinationen zulässig (alle anderen Kombinationen sind ungültig):

Struktur	Möglichkeit	Beschreibung
	$A^{(1,2,3)}$	Das Attribut `Reference` wird nicht verwendet, der Inhalt von `sl:DataObject` repräsentiert das Datenobjekt. Ist das Datenobjekt XML-kodiert (Verwendung des Elements `sl:XMLContent` in `sl:DataObject`), muss es als geparstes XML in die Signaturstruktur eingebunden werden.
		Das Attribut `Reference` enthält eine URI, die von der *Bürgerkarten-Umgebung* aufgelöst werden MUSS, um das Datenobjekt zu erhalten. Der

"enveloping"	B(2,4)	Inhalt von sl:DataObject bleibt leer. Handelt es sich bei dem so referenzierten Datenobjekt um Text oder XML-Daten, ist es sinnvoll, das Datenobjekt als geparstes XML in die Signaturstruktur einzubinden. Die *Bürgerkarten-Umgebung* SOLL daher bei der Auflösung Informationen des Transportprotokolls auswerten, um das eventuelle Vorliegen von Text oder XML-Daten herauszufinden. Für eine URI, die als Protokoll http oder https aufweist, MUSS dazu die Information im HTTP-Header (Content-Type) ausgewertet werden.
"detached"	C	Das Attribut Reference enthält eine URI, die von der *Bürgerkarten-Umgebung* aufgelöst werden muss, um das Datenobjekt zu erhalten. Darüber hinaus wird diese URI auch für die Kodierung der Referenz auf das Datenobjekt als Bestandteil der XML-Signatur verwendet (Attribut URI im Element dsig:Reference). Der Inhalt von sl:DataObject bleibt leer.
	D(1,5)	Das Attribut Reference enthält eine URI, die von der *Bürgerkarten-Umgebung* für die Kodierung der Referenz auf das Datenobjekt als Bestandteil der XML-Signatur verwendet wird (Attribut URI im Element dsig:Reference). Der Inhalt von sl:DataObject repräsentiert das Datenobjekt.

(1) Soll der Inhalt von sl:Dataobject zur expliziten Angabe des Datenobjekts verwendet werden, stehen drei unterschiedliche Arten der Kodierung zur Verfügung:

- sl:Base64Content: Enthält das Datenobjekt in base64-kodierter Form.
- sl:XMLContent: Enthält das Datenobjekt als XML kodiert. Wenn es aus Sicht der *Applikation* wichtig ist, dass die *Bürgerkarten-Umgebung* Whitespace innerhalb des übergebenen XML nicht verändert, kann für sl:XMLContent das Attribut xml:space mit dem Wert preserve versehen werden. Das Inhaltsmodell von sl:XMLContent ist so definiert, dass es eine beliebige Mischung aus Text und XML-Markup erlaubt. Das schließt ausdrücklich auch reinen Text mit ein. Eine gültige Instanz von sl:XMLContent ist also beispielsweise auch <sl:XMLContent>Text</sl:XMLContent>.
- sl:LocRefContent: Enthält eine Referenz auf das Datenobjekt, die von der *Bürgerkarten-Umgebung* aufzulösen ist, um das Datenobjekt zu erhalten.

(2) Die Einbindung eines im Fall A oder B übergebenen Datenobjekts in die XML-Signatur bzw. die Referenzierung auf dieses in die Signatur eingebundene Datenobjekt aus dem zugehörigen dsig:Reference Element der XML-Signatur muss so erfolgen, dass ausschließlich die im Request in sl:DataObject übergebenen Daten signiert werden.Wird beispielsweise das Datenobjekt als Inhalt eines dsig:Object Elements in die XML-Signatur eingebunden, darf dieses dsig:Object Containerelement nicht mitsigniert werden, sondern lediglich sein Inhalt.

(3) Wenn das Datenobjekt base64-kodiert vorliegt (Verwendung des Elements sl:Base64Content in sl:DataObject), MÜSSEN jedenfalls die Base64-dekodierten Daten signiert werden. Kann die *Bürgerkarten-Umgebung* die Base64-dekodierten Daten nicht direkt in das dsig:Object integrieren (weil darin Zeichen enthalten sind, die nicht als XML-Text darstellbar sind), MUSS sie stattdessen die base64-kodierten Daten integrieren und eine Base64 Transformation als erste Transformation im zugehörigen dsig:Reference verwenden.

(4) Handelt es sich beim referenzierten Datenobjekt weder um Text noch um XML-Daten, oder wird das Datenobjekt entgegen der Empfehlung nicht auf Vorliegen von Text oder XML-Daten geprüft, muss es in base64-kodierter Form in die Signaturstruktur eingebunden werden. Signiert werden muss jedoch das ursprüngliche Datenobjekt; es ist daher in der auf das eingebundene Datenobjekt verweisenden dsig:Reference eine Base64 Transformation zu verwenden.

(5) Handelt es sich bei der in Reference übergebenen URI um eine interne URI gemäß [URI], Abschnitt 4.2 (*Same-document Reference*), so ist sie auf jenes XML-Dokument zu beziehen, in das die Signatur eingebettet werden soll (und welches in sl:SignatureEnvironment übergeben wird - vergleiche Abschnitt 2.2.1.3, „Informationen zum Signaturdokument“).

Transformationswege

Weiters enthält der Behälter sl:DataObjectInfo einen oder mehrere Transformationswege für das Datenobjekt (sl:TransformsInfo).

Ein Transformationsweg beschreibt dabei eine Kette von auszuführenden Transformationen (dsig:Transforms), um vom Datenobjekt zu jenen Daten zu gelangen, die in die Hashberechnung und in weiterer Folge in die Signaturberechnung einfließen (nachfolgend als Hash-Eingangsdaten bezeichnet).

Die Hash-Eingangsdaten sind gleichzeitig jene Daten, die gegebenenfalls in der *Bürgerkarten-Umgebung* dem Benutzer angezeigt werden müssen. Damit die *Bürgerkarten-Umgebung* weiß, welcher Natur die Hash-Eingangsdaten sind, enthält der Transformationsweg andererseits Metainformationen darüber (`sl:FinalDataMetaInfo`). Jedenfalls ist der Mime-Type (siehe [MIME]) anzugeben (`sl:MimeType`), zusätzlich DARF eine eine verbale Beschreibung dieser Daten angegeben werden (`sl:Description`).

> **Anmerkung**
>
> Soll das zu signierende Datenobjekt von der *Bürgerkarten-Umgebung* als Text interpretiert werden, MUSS die *Applikation* den Mime-Type `text/plain` verwenden; soll es im Sinne des Standard-Anzeigeformat des *Security-Layer* interpretiert werden, MUSS die *Applikation* den Mime-Type `application/xhtml+xml` verwenden. Siehe dazu auch Minimale Umsetzung des Security-Layers.

Soll das Datenobjekt direkt unterzeichnet werden, wird ebenfalls ein Transformationsweg spezifiziert, jedoch unterbleibt die Angabe der Kette von Transformationen. Bei Angabe mehrerer Transformationswege wählt die *Bürgerkarten-Umgebung* einen Weg frei aus.

Bei der Angabe eines Transformationsweges muss die *Applikation* sicherstellen, dass alle Namenräume, die innerhalb der Struktur der spezifizierten Transformationen (`dsig:Transforms`) verwendet werden, innerhalb dieser Struktur explizit deklariert werden. Die *Bürgerkarten-Umgebung* DARF NICHT Namenraum-Deklarationen innerhalb von `dsig:Transforms` hinzufügen, wenn Sie die Struktur in die zu erzeugende Signatur übernimmt.

Ergänzungsobjekte

Optional werden schließlich im Behälter `sl:DataObjectInfo` Ergänzungsobjekte (`sl:Supplements`) angegeben. Solche können übergeben werden, damit Daten, die im Rahmen des Transformationsprozesses für das Datenobjekt oder für die Anzeige benötigt werden, nicht von der *Bürgerkarten-Umgebung* aufgelöst werden müssen. Das Datenobjekt selbst darf jedoch nicht als Ergänzungsobjekt übergeben werden.

Als Beispiel sei hier eine Stylesheet-Transformation angeführt, die sich verschachtelter Stylesheets bedient: Nur der Basis-Stylesheet ist tatsächlich im Transformationsobjekt (`dsig:Transform`) als Parameter angeführt; im Basis-Stylesheet referenzierte weitere Stylesheets müssten von der *Bürgerkarten-Umgebung* selbst aufgelöst werden. Dies kann vermieden werden, indem solche referenzierte Stylesheets als Ergänzungsobjekte übergeben werden.

Ein Ergänzungsobjekt besteht dabei einerseits aus OPTIONALEN Metainformationen (vergleiche `sl:FinalDataMetaInfo`), andererseits aus dem eigentlichen Daten (`sl:Content`): Das verpflichtend zu verwendende Attribut `Reference` enthält dabei als URI die Referenz auf die Ergänzungsdaten, und zwar so, wie sie von der *Bürgerkarten-Umgebung* zur Auflösung verwendet werden würde. Der Inhalt von `sl:Content` repräsentiert die Ergänzungsdaten (siehe auch Hinweis Content in „Datenobjekt“).

Stößt die *Bürgerkarten-Umgebung* während der Berechnung des Transformationsprozesses auf aufzulösende Daten, muss Sie folgenden Ablauf für die Auflösung einhalten:

1. Prüfung, ob die aufzulösende Referenz in einem `sl:Supplement` innerhalb jenes `sl:DataObjectInfo` vorkommt, das den gerade berechneten Transformationsprozess spezifiziert. Ist diese Prüfung erfolgreich, sind die in diesem `sl:Supplement` angegebenen Daten als Ergebnis der Auflösung zu verwenden. Ansonsten ist mit Schritt 2 fortzufahren.
2. Prüfung, ob die aufzulösende Referenz in einem `sl:Supplement` eines anderen `sl:DataObjectInfo` vorkommt, das im Befehl zur Erzeugung der XML-Signatur spezifiziert wurde. Die einzelnen `sl:DataObjectInfo`-Elemente sind dabei in jener Reihenfolge zu untersuchen, in der Sie im Befehl angegeben wurden. Die im ersten so gefundenen `sl:Supplement` angegebenen Daten sind als Ergebnis der Auflösung zu verwenden. Wird kein `sl:Supplement` gefunden, ist mit Schritt 3 fortzufahren.
3. Auflösung der Referenz.

2.2.1.3. Informationen zum Signaturdokument

Soll die zu erstellende Signatur in ein bestehendes XML-Dokument eingebettet werden, findet sich in der Signaturanfrage schließlich der Behälter `sl:SignatureInfo`. Bei Fehlen dieses Behälters ist die Signatur nirgends einzubetten, sondern direkt als Ergebnis der Anfrage (siehe Abschnitt 2.2.2, „Antwort“) zurückzuliefern.

Das Signaturdokument

`sl:SignatureInfo` enthält zunächst Angaben zum Signaturdokument, in das die Signatur eingebettet werden soll (`sl:SignatureEnvironment`). Entweder enthält das Attribut `Reference` einen Verweis auf

dieses Dokument, der von der *Bürgerkarten-Umgebung* aufzulösen ist, oder das Dokument selbst ist als Inhalt von `sl:SignatureEnvironment` angegeben.

Die direkte Einbettung kann auf zwei Arten erfolgen: Entweder wird das XML-Dokument base64 kodiert und als Text des Kindelements `sl:Base64Content` integriert, oder aber das Wurzelelement des XML-Dokuments wird direkt als einziges Kind des Kindelements `sl:XMLContent` angegeben. Siehe dazu auch den Hinweis Content in „Datenobjekt“.

Anmerkung

Wird ein XML-Dokument als Signaturdokument verwendet, das eine Document Type Declaration (siehe [XML]) verwendet, SOLL die erste Variante der direkten Einbettung (base64 Kodierung) verwendet werden, damit die dort enthaltenen Informationen vom XML-Parser der *Bürgerkarten-Umgebung* ausgewertet werden können. Bei der Einbettung des Wurzelelements in `sl:XMLContent` können diese Informationen nicht angegeben werden.

Anmerkung

Für das Parsen des Signaturdokuments wird folgende Vorgangsweise empfohlen: In einem ersten Schritt versucht die Bürgerkarten-Umgebung das Dokument validierend zu parsen. Informationen über die Grammatik des Dokuments liefern dazu eine ggf. im Dokument vorhandene Document Type Declaration (siehe [XML]), oder XML-Schemata, die mit den in [XML-Schema], Abschnitt 2.6.3 angegebenen Mechanismen referenziert werden. Sollte dieser erste Versuch scheitern, parst die Bürgerkarten-Umgebung das Dokument in einem zweiten Anlauf nichtvalidierend, d. h. ohne Auswertung von Grammatikinformationen.

Position der Signatur

Das darauffolgende Element `sl:SignatureLocation` enthält Informationen, an welcher Position im Signaturdokument die zu erstelltende Signatur von der *Bürgerkarten-Umgebung* eingefügt werden soll.

Der Textinhalt des Elements enthält einen Ausdruck nach [XPath], mit dem das Eltern-Element der einzufügenden Signatur ausgewählt wird. Als Kontext-Knoten für die Auswertung des XPath-Ausdrucks ist der Dokument-Knoten des in `sl:SignatureEnvironment` spezifizierten XML-Dokuments zu verwenden. Werden im XPath-Ausdruck Namespace-Prefixes verwendet, MÜSSEN die entsprechenden Namespace-Deklarationen im Kontext des Elements `sl:SignatureLocation` bekannt sein.

Das Attribut `Index` selektiert die Position der Signatur innerhalb des Elternelements. Hat `Index` den Wert 0, wird die Signatur als erster Kindknoten des Elternelements eingefügt, hat `Index` den Wert n, wird die Signatur unmittelbar nach dem n-ten Kindknoten des Elternelements eingefügt.

Ergänzungsobjekte

Zusätzlich KÖNNEN schließlich Ergänzungsobjekte (`sl:Supplements`) spezifiziert werden, die in Zusammenhang mit dem Signaturdokument stehen.

Beispielsweise könnte im Signaturdokument, das in `sl:SignatureEnvironment` spezifiziert wurde, ein Verweis auf die Dokumenttypdefinition enthalten sein. Das auf diese Weise referenzierte Dokument kann von der *Applikation* als Ergänzungsobjekt übergeben werden, wenn die *Bürgerkarten-Umgebung* den Verweis nicht selbst auflösen soll, oder wenn die *Bürgerkarten-Umgebung* den Verweis gar nicht auflösen kann, weil es sich etwa um einen relativen Veweis, bezogen auf das Signaturdokument, handelt.

Ein weiteres Beispiel für die Verwendung eines Ergänzungsobjekts könnte ein XML-Schema sein, das im mittels `sl:SignatureEnvironment` spezifizierten Signaturdokument unter Verwendung der in [XML-Schema] vorgeschlagenen Mechanismen (siehe Anmerkung im vorangegangenen „Das Signaturdokument“) referenziert wird.

Ein Ergänzungsobjekt besteht dabei einerseits aus OPTIONALEN Metainformationen (vergleiche `sl:FinalDataMetaInfo` in „Transformationswege“), andererseits aus dem eigentlichen Daten (`sl:Content`): Das verpflichtend zu verwendende Attribut `Reference` enthält dabei als URI die Referenz auf die Ergänzungsdaten, und zwar so, wie sie von der *Bürgerkarten-Umgebung* zur Auflösung verwendet werden würde. Der Inhalt von `sl:Content` repräsentiert die Ergänzungsdaten (siehe auch Hinweis Content in „Datenobjekt“).

2.2.2. Antwort

Die Antwort enthält die nach [XMLDSIG] kodierte elektronische Signatur. Wurde in der Anfrage ein `sl:SignatureInfo` Element angegeben, enthält die Antwort als einziges Kind das in `sl:SignatureInfo`

angegebene Dokument mit der darin integrierten Signatur. Ansonsten enthält die Antwort direkt die erzeugte Signatur.

2.2.2.1. Signierte Daten

Für jedes in der Anfrage mittels Behälter (`sl:DataObjectInfo`) übergebene Datenobjekt enthält die XML-Signatur ein `dsig:Reference` Element. In dessen `dsig:Transforms` Element ist jene Transformationskette anzugeben, die in der *Bürgerkarten-Umgebung* durchlaufen wurde, um aus dem übergebenen Datenobjekt jene Daten zu erhalten, die für die Berechnung des Hash-Wertes sowie gegebenenfalls für die Anzeige im Secure Viewer verwendet worden sind.

2.2.2.2. Implizite Transformationsparameter

Um den korrekten Zusammenhang zwischen den *Referenz-Eingangsdaten* sowie den *Hash-Eingangsdaten* später jederzeit überprüfen zu können, MÜSSEN die impliziten Transformationsparameter aller in die Signatur aufzunehmenden Datenobjekte in ein einziges Signaturmanifest aufgenommen werden. Liegen keine impliziten Transformationsparameter vor, darf das Signaturmanifest nicht erstellt werden.

Ein impliziter Transformationsparameter ist in diesem Zusammenhang ein Datum, das von der *Bürgerkarten-Umgebung* zur Berechnung der Transformationen für ein zu signierendes Datenobjekt verwendet wurde, jedoch nicht explizit als Parameter im entsprechenden Transformationsobjekt (`dsig:Transform`) aufscheint.

Als (derzeit einzig bekanntes) Beispiel soll hier wiederum die bereits in „Transformationswege“ erwähnte Stylesheet-Transformation erwähnt werden, die sich verschachtelter Stylesheets bedient. Die im Basis-Stylesheet referenzierten weiteren Stylesheets sind implizite Transformationsparameter im obigen Sinne.

Das Signaturmanifest (`dsig:Manifest`) enthält für jeden impliziten Transformationsparameter ein eigenes Referenzobjekt (`dsig:Reference`), das einen Hash-Wert für den impliziten Transformationsparameter inkludiert. Transformationen in den Referenzobjekten des Signaturmanifests DÜRFEN NICHT verwendet werden. Das Attribut URI eines Referenzobjekts enthält dabei die Referenz auf den impliziten Transformationsparameter, und zwar in exakt gleicher Weise, wie sie von der *Bürgerkarten-Umgebung* im Falle der Signaturprüfung zur Auflösung verwendet werden würde.

Die Eingangsdaten für die Berechnung des Hash-Wertes über einen implizten Transformationsparameter ergeben sich wie folgt:

- Wird der implizite Transformationsparameter als Referenz angegeben, die von der *Bürgerkarten-Umgebung* selbst aufzulösten ist, ist zwischen einer externen und einer internen Referenz zu unterscheiden:
  - Die Auflösung einer externen Referenz MUSS einen Byte-Stream liefern. Dieser Byte-Stream bildet die Eingangsdaten für die Hash-Berechnung.
  - Die Auflösung einer internen Referenz MUSS eine XPath-Knotenmenge liefern (vgl. [XMLDSIG], Abschnitt 4.3.3.3]). Diese Knotenmenge ist nach [C14N] zu kanonisieren, um einen eindeutigen Byte-Stream zu erhalten. Dieser Byte-Stream bildet dann die Eingangsdaten für die Hash-Berechnung.
- Wird der implizite Transformationsparameter als Referenz angegeben, die von der *Bürgerkarten-Umgebung* nicht selbst aufzulösen ist, weil in der Anfrage ein entsprechendes Ergänzungsobjekt angegeben wurde, ist wiederum zwischen einer externen und einer internen Referenz zu unterscheiden:
  - Enthält das Ergänzungsobjekt Daten für eine externe Referenz, MÜSSEN diese Daten als Base64 (in `sl:Supplement/sl:Content/sl:Base64Content`) vorliegen. Die Base64-dekodierten Daten bilden dann die Eingangsdaten für die Hash-Berechnung.
  - Enthält das Ergänzungsobjekt Daten für eine interne Referenz, MÜSSEN diese Daten als XML (in `sl:Supplement/sl:Content/sl:XMLContent`) vorliegen. Aus diesen Daten ist entsprechend [XMLDSIG], Abschnitt 4.3.3.3] eine XPath-Knotenmenge zu erzeugen. Diese Knotenmenge ist nach [C14N] zu kanonisieren, um einen eindeutigen Byte-Stream zu erhalten. Dieser Byte-Stream bildet dann die Eingangsdaten für die Hash-Berechnung.

Die Einbindung des Signaturmanifests in die Signatur erfolgt durch ein eigenes Referenzobjekt der Signatur (`dsig:Reference` in `dsig:SignedInfo`). Dabei ist die Verwendung des Attributs `Type` im Referenzobjekt zur Kennzeichnung des referenzierten Datums als Signaturmanifest ERFORDERLICH. Das Attribut MUSS folgenden Wert aufweisen:

```
http://www.buergerkarte.at/specifications/securitylayer/20020225#SignatureManifest
```

2.2.2.3. Signaturattribute

Die nachfolgend angegebenen Informationen müssen in die Signatur unter Verwendung von Signaturattributen nach [ETSIXML] aufgenommen werden. Die Einbindung der Signaturattribute in die Signatur MUSS als als Direct Incorporation entsprechend den Hinweisen in den Abschnitten 6.3 und insbesondere 6.3.1 von [ETSIXML] erfolgen. Das in Abschnitt 6.3.1 erwähnte Attribut `Type` MUSS verwendet werden.

Signierte Metainformationen

Für jedes in der Anfrage spezifizierte zu signierende Datenobjekt (`sl:DataObject`), istMUSS ein Signaturattribut aufgenommen werden, das die dazu spezifizierten Metainformationen (`sl:FinalDataMetaInfo`) enthält.

Dazu ist das signierte Signaturattribut `etsi:DataObjectFormat` nach [ETSIXML] zu verwenden. Das Element `sl:MimeType` aus `sl:FinalDataMetaInfo` entspricht dabei dem Element `etsi:MimeType` aus `etsi:DataObjectFormat`, das optional vorhandene Element `sl:Description` aus `sl:FinalDataMetaInfo` dem Element `etsi:Description` aus `etsi:DataObjectFormat`.

Signierte Zertifikatsreferenz

Weiters MUSS ein Signaturattribut mitaufgenommen werden, mit dem das für die Verifikation der Signatur zu verwendende Signatorzertifikat eindeutig identifiziert wird. Dazu ist das signierte Signaturattribut `etsi:SigningCertificate` nach [ETSIXML] zu verwenden.

Zeitpunkt der Signaturerstellung

Schließlich MUSS ein Signaturattribut mitaufgenommen werden, das den vom Signator behaupteten Zeitpunkt der Signaturerstellung enthält. Dazu ist das signierte Signaturattribut `etsi:SigningTime` nach [ETSIXML] zu verwenden.

> **Anmerkung**
>
> Für Beispiele zu diesem Befehl siehe Tutorium; für Anforderungen an die Benutzerschnittstelle siehe Abschnitt 3.1, „Signaturerstellung“ in Die österreichische Bürgerkarte - Anforderungen an die Benutzer-Schnittstelle.

3. Signaturprüfung

3.1. Signatur nach CMS

3.1.1. Anfrage

Mit dieser Anfrage wird eine beliebige Signatur nach [CMS], die nicht notwendigerweise das Profil *dieser* Spezifikation einhält, an die *Bürgerkarten-Umgebung* zur Überprüfung übergeben.

3.1.1.1. Signatoren

Im Attribut `Signatories` enthält die Anfrage zunächst Angaben über jene in der CMs-Signatur angegebenen Signatoren, deren Unterschriften überprüft werden sollen. Der Wert dieses Attributs enthält eine Aufzählung von positiven Ganzzahlen, wobei eine Ganzzahl jeweils die Position des Signators enthält, so wie er in der CMS-Signatur (Strukturelement `SignerInfo`) vorkommt. Wird dieses Attribut nicht spezifiziert, so ist es mit dem Wert `1` anzunehmen, d. h. es ist nur die Signatur des ersten in `SignerInfo` spezifizierten Signators zu überprüfen.

3.1.1.2. Prüfzeitpunkt

Als nächstes enthält die Anfrage zur Signaturüberprüfung optional die Angabe jenes Zeitpunktes, der zur Feststellung der Gültigkeit der zu überprüfenden Signatur verwendet werden soll (`sl:DateTime`). Fehlt diese Angabe, MUSS die *Bürgerkarten-Umgebung* wie folgt zur Bestimmung des Prüfzeitpunkts vorgehen:

- Enthält die Signatur ein Signaturattribut `SigningTime` nach [ETSICMS], das den vom Signator behaupteten Zeitpunkt der Signaturerstellung enthält, ist die darin genannte Zeit als Prüfzeitpunkt zu verwenden.
- Existiert kein solches Signaturattribut, ist als Prüfzeitpunkt der Zeitpunkt des Einlangens der Überprüfungsanfrage bei der *Bürgerkarten-Umgebung* zu verwenden.

3.1.1.3. Signatur

Nachfolgend enthält die Anfrage die zu überprüfende Signatur nach [CMS] (`sl:CMSSignature`) in base64-kodierter Form.

3.1.1.4. Datenobjekt

Falls das signierte Datenobjekt in der zu prüfenden [CMS]-Signatur nicht mitkodiert ist, muss es in der Anfrage im Element `sl:DataObject` mitübergeben werden.

In `sl:DataObject` können zunächst Metainformationen (`sl:MetaInfo`) angegeben werden (Mime-Type sowie eine Referenz auf eine Beschreibung des signierten Datenobjekts). Danach folgt die Angabe der Daten auf eine von zwei Arten: Entweder enthält das Element `sl:Content` die base64-kodierten Daten, oder das Element `sl:Content` ist leer, hat aber sein Attribut `Reference` gesetzt. Die *Bürgerkarten-Umgebung* versucht im letztenFall, die in diesem Attribut angegebene URI aufzulösen, um so das signierte Datenobjekt zu erhalten.

3.1.2. Antwort

Für jede in der Anfrage spezifizierte zu prüfende Signatur werden in der Antwort folgende Informationen zurückgeliefert:

Zunächst werden in in `sl:SignerInfo` Informationen zum X.509-Zertifikat des Signators angegeben. `sl:SignerInfo` MUSS aus genau einem Element `dsig:X509Data` bestehen, das mindestens folgende zwei Elemente enthalten MUSS:

- `dsig:SubjectName` enthält den Namen des Signators aus dem Signatorzertifikat. Das Element ist wie in [XMLDSIG], Kapitel 4.4.4 angegeben zu kodieren. In `dsig:X509Data` MUSS genau ein solches Element vorkommen.
- `dsig:IssuerSerial` enthält den Namen des Austellers und die Seriennummer des Signatorzertifikats. Das Element ist wie in [XMLDSIG], Kapitel 4.4.4 angegeben zu kodieren. In `dsig:X509Data` muss genau ein solches Element vorkommen.

Weiters muss dieses `dsig:X509Data` das leere Element `sl:QualifiedCertificate` enthalten, wenn das Signatorzertifikat als qualifiziert anzusehen ist. Dazu müssen folgende Bedingungen erfüllt sein:

1. Es muss die Zertifikatserweiterung `qcStatements` (siehe [QCert], 3.2.5) vorhanden sein.
2. Innerhalb dieses Zertifikatserweiterung muss das Statement `esi4-qcStatement-1` (siehe [ETSIQCert], 4.2.1) vorhanden sein.

Ob weitere Elemente - wie beispielsweise das Signatorzertifikat selbst - zurückgeliefert werden, bleibt der *Bürgerkarten-Umgebung* überlassen. Konnte bei der Überprüfung kein Signatorzertifikat nach X.509 identifiziert werden, bleibt es ebenfalls der *Bürgerkarten-Umgebung* überlassen, welche Informationen über den öffentlichen Schlüssel sie zurückliefert.

Weiters enthalten sind getrennte Ergebnisse für die kryptographische Überprüfung der Signatur (`sl:SignatureCheck`) sowie für die Prüfung der Signaturprüfdaten (`sl:CertificateCheck`).

Beide Ergebnisse besitzen die gleiche Struktur: Das Element `sl:Code` enthält das Ergebnis der Prüfung in maschinenlesbarer Form, das optionale Element `sl:Info` enthält genauere, nicht näher spezifizierte Zusatzinfomationen. Vorstellbar ist etwa eine vom Menschen lesbare Klartext-Interpretation des Prüfungsergebnises.

3.1.2.1. Prüfung der Gültigkeit der Signatur

Wurde die in der Anfrage spezifizierte Signatur konform zu dieser Spezifikation erstellt, so enthält sie ein Signaturattribut `ContentHints` nach [ESS-S/MIME], Abschnitt 2.9, das Metainformationen zum signierten Datenobjekt enthält. Dieses Signaturattribut kann von der *Bürgerkarten-Umgebung* bei Bedarf ausgewertet werden.

Folgende Werte sind für den Inhalt des Elements `sl:Code` in `sl:SignatureCheck` definiert:

`sl:Code`	**Bedeutung**
0	Die Überprüfung des Werts der Signatur konnte erfolgreich durchgeführt werden.
1	Bei der Überprüfung des Werts der Signatur ist ein Fehler aufgetreten.

3.1.2.2. Prüfung der Signaturprüfdaten

Diese umfaßt die Konstruktion der Zertifikatskette vom Signatorzertifikat bis zu einem vertrauenswürdigen Wurzelzertifikat, sowie die Statusprüfung für jedes Zertifikat der konstruierten Zertifikatskette. Die Prüfung der Signaturprüfdaten darf unterbleiben, wenn bei der Prüfung der Gültigkeit der Signatur ein Fehler aufgetreten ist.

Wurde die in der Anfrage spezifizierte Signatur konform zu dieser Spezifikation erstellt, so enthält sie ein Signaturattribut `OtherSigningCertificate` nach [ETSICMS], das Informationen zum Signatorzertifikat enthält. Diese Informationen MÜSSEN von der *Bürgerkarten-Umgebung* in einem solchen Fall als Ausgangspunkt zur Konstruktion der Zertifikatskette verwendet werden.

Bei der Überprüfung älterer Signaturen kann es vorkommen, dass vom Zertifizierungsdiensteanbieter, der das Signatorzertifikat ausgestellt hat, keinerlei Online-Informationen mehr angeboten werden, die die Bildung der Zertifikatskette bzw. die Statusprüfung der Zertifikate in dieser Kette ermöglichen. Damit solche Signaturen trotzdem überprüft werden können, wird EMPFOHLEN, dass die *Bürgerkarten-Umgebung* in solchen Fällen die ggf. vorhandenen unsignierten Signaturattribute `CompleteCertificateRefs`, `CompleteRevocationRefs`, `CertificateValues` und `RevocationValues` nach [ETSICMS] auswertet.

Folgende Werte sind für den Inhalt des Elements `sl:Code` in `sl:CertificateCheck` definiert:

`sl:Code`	**Bedeutung**
0	Eine formal korrekte Zertifikatskette vom Signatorzertifikat zu einem vertrauenswürdigen Wurzelzertifikat konnte konstruiert werden. Jedes Zertifikat dieser Kette ist zum in der Anfrage angegebenen Prüfzeitpunkt gültig.
1	Es konnte keine formal korrekte Zertifikatskette vom Signatorzertifikat zu einem vertrauenswürdigen Wurzelzertifikat konstruiert werden.
2	Eine formal korrekte Zertifikatskette vom Signatorzertifikat zu einem vertrauenswürdigen Wurzelzertifikat konnte konstruiert werden. Für zumindest ein Zertifikat dieser Kette fällt der Prüfzeitpunkt nicht in das Gültigkeitsintervall.
3	Eine formal korrekte Zertifikatskette vom Signatorzertifikat zu einem vertrauenswürdigen Wurzelzertifikat konnte konstruiert werden. Für alle Zertifikate dieser Kette fällt der Prüfzeitpunkt in das jeweilige Gültigkeitsintervall. Für zumindest ein Zertifikat konnte der Zertifikatstatus nicht festgestellt werden.
4	Eine formal korrekte Zertifikatskette vom Signatorzertifikat zu einem vertrauenswürdigen Wurzelzertifikat konnte konstruiert werden. Für alle Zertifikate dieser Kette fällt der Prüfzeitpunkt in das jeweilige Gültigkeitsintervall. Zumindest ein Zertifikat ist zum Prüfzeitpunkt widerrufen.
5	Eine formal korrekte Zertifikatskette vom Signatorzertifikat zu einem vertrauenswürdigen Wurzelzertifikat konnte konstruiert werden. Für alle Zertifikate dieser Kette fällt der Prüfzeitpunkt in das jeweilige Gültigkeitsintervall. Kein Zertifikat dieser Kette ist zum Prüfzeitpunkt widerrufen. Zumindest ein Zertifikat ist zum Prüfzeitpunkt gesperrt.
99	Die Prüfung der Signaturprüfdaten wurde nicht durchgeführt, da bei der Prüfung der Gültigkeit der Signatur ein Fehler aufgetreten ist.

Anmerkung

Für Beispiele zu diesem Befehl siehe Tutorium; für Anforderungen an die Benutzerschnittstelle siehe Abschnitt 3.2, „Signaturprüfung“ in Die österreichische Bürgerkarte - Anforderungen an die Benutzer-Schnittstelle.

3.2. Signatur nach XMLDSIG

3.2.1. Anfrage

Mit dieser Anfrage wird eine beliebige Signatur nach [XMLDSIG], die nicht notwendigerweise das Profil dieser Spezifikation einhält, an die *Bürgerkarten-Umgebung* zur Überprüfung übergeben.

3.2.1.1. Prüfzeitpunkt

Die Anfrage zur Signaturüberprüfung enthält zunächst optional die Angabe jenes Zeitpunktes, der zur Feststellung der Gültigkeit der zu überprüfenden Signatur verwendet werden soll (`sl:DateTime`). Fehlt diese Angabe, MUSS die *Bürgerkarten-Umgebung* wie folgt zur Bestimmung des Prüfzeitpunkts vorgehen:

- Enthält die Signatur ein Signaturattribut `etsi:SigningTime` nach [ETSIXML], das den vom Signator behaupteten Zeitpunkt der Signaturerstellung enthält, ist die darin genannte Zeit als Prüfzeitpunkt zu verwenden.

- Existiert kein solches Signaturattribut, ist als Prüfzeitpunkt der Zeitpunkt des Einlangens der Überprüfungsanfrage bei der *Bürgerkarten-Umgebung* zu verwenden.

3.2.1.2. Signatur

Nachfolgend enthält die Anfrage Angaben zur zu überprüfenden Signatur nach [XMLDSIG] (`sl:SignatureInfo`).

Zunächst enhält `sl:SignatureInfo` Angaben zum XML-Dokument, das die zu überprüfende Signatur inkludiert (`sl:SignatureEnvironment`). Entweder enthält das Attribut `Reference` einen Verweis auf dieses Dokument, der von der *Bürgerkarten-Umgebung* aufzulösen ist, oder das Dokument selbst ist als Inhalt von `sl:SignatureEnvironment` angegeben.

Die direkte Einbettung kann auf zwei Arten erfolgen: Entweder wird das XML-Dokument base64 kodiert und als Text des Kindelements `sl:Base64Content` integriert, oder aber das Wurzelelement des XML-Dokuments wird direkt als einziges Kind des Kindelements `sl:XMLContent` angegeben. Siehe dazu auch den Hinweis Content in „Transformationswege“, Datenobjekt.

> **Anmerkung**
>
> Wird ein XML-Dokument angegeben, das eine Document Type Declaration (siehe [XML]) verwendet, SOLL die erste Variante der direkten Einbettung (base64 Kodierung) verwendet werden, damit die dort enthaltenen Informationen vom XML-Parser der *Bürgerkarten-Umgebung* ausgewertet werden können. Bei der Einbettung des Wurzelelements in `sl:XMLContent` können diese Informationen nicht angegeben werden.

> **Anmerkung**
>
> Für das Parsen des XML-Dokuments, das die zu überprüfende Signatur beinhaltet, siehe Anmerkung in Abschnitt 2.2.1.3, „Informationen zum Signaturdokument“.

Weiters enthält `sl:SignatureInfo` mit `sl:SignatureLocation` einen Ausdruck nach [XPath], mit dem anzugeben ist, wo sich die zu überprüfende Signatur innerhalb des mittels `sl:SignatureEnvironment` spezifizierten XML-Dokuments befindet. Als Kontext-Knoten für die Auswertung des XPath-Ausdrucks ist der Dokument-Knoten des in `sl:SignatureEnvironment` spezifizierten XML-Dokuments zu verwenden. Werden im XPath-Ausdruck Namespace-Prefixes verwendet, MÜSSEN die entsprechenden Namespace-Deklarationen im Kontext des Elements `sl:SignatureLocation` bekannt sein.

3.2.1.3. Ergänzungsobjekte

Optional können in der Anfrage schließlich Ergänzungsobjekte übergeben werden. Das sind Datenobjekte, auf die innerhalb der Signaturstruktur verwiesen wird. Beispiele dafür sind:

- ein Datenobjekt, das im URI Attribut eines `dsig:Reference` Elements der Signatur referenziert wird;
- referenzierte Parameter für die Transformationen in einem `dsig:Reference` Elements der Signatur (z. B. ein Stylesheet, der im Basis-Stylesheet einer XSLT-Transformation referenziert wird);
- referenzierte Informationen für die Validierung des Dokuments, das die Signatur beinhaltet (Verweis in der Document Type Declaration, Verweise auf XML-Schemata mittels der in [XML-Schema], Abschnitt 2.6.3 angegebenen Mechanismen);
- referenzierte Informationen für die Aufindung des für die Signaturprüfung zu verwendenden Zertifikats (vergleiche [XMLDSIG], Abschnitt 4.4.3).

Für ein Ergänzungsobjekt (`sl:Supplement`) DÜRFEN zunächst Metainformationen (`sl:MetaInfo`) angegeben werden (Mime-Type sowie eine Referenz auf eine Beschreibung des Ergänzungsobjekts). Danach folgen die verpflichtenden Angaben zum Inhalt des Objekts (`sl:Content`): Das Attribut `Reference` enthält dabei als URI die Referenz auf das Ergänzungsobjekt, und zwar in exakt gleicher Weise, wie sie von der *Bürgerkarten-Umgebung* zur Auflösung verwendet werden würde. Der Inhalt von `sl:Content` repräsentiert das Ergänzungsobjekt (siehe auch Hinweis Content in „Datenobjekt“).

Werden in der Anfrage Ergänzungsobjekte übergeben, MUSS die *Bürgerkarten-Umgebung* diese verwenden, anstatt die entsprechenden Referenzen selbst aufzulösen.

> **Anmerkung**
>
> Notwendig ist die Angabe von Ergänzungsobjekten, wenn in der Signaturstruktur Verweise relativ zum Signaturdokument vorhanden sind.

3.2.2. Antwort

Die Antwort auf die Anfrage zur Prüfung einer Signatur nach XML enthält zunächst Informationen zum öffentlichen Schlüssel des Signators (`sl:SignerInfo`). Konnte bei der Überprüfung ein dem öffentlichen Schlüssel entsprechendes Signatorzertifikat nach X.509 identifiziert werden, MUSS `sl:SignerInfo` zumindest folgende Informationen enthalten:

- Genau ein `dsig:X509Data` Element, das wiederum mindestens drei Elemente enthalten MUSS:
  - `dsig:SubjectName` enthält den Namen des Signators aus dem Signatorzertifikat. Das Element ist wie in [XMLDSIG], Kapitel 4.4.4 angegeben zu kodieren. In `dsig:X509Data` MUSS genau ein solches Element vorkommen.
  - `dsig:IssuerSerial` enthält den Namen des Austellers und die Seriennummer des Signatorzertifikats. Das Element ist wie in [XMLDSIG], Kapitel 4.4.4 angegeben zu kodieren. In `dsig:X509Data` MUSS genau ein solches Element vorkommen.
  - Weiters muss `dsig:X509Data` das leere Element `sl:QualifiedCertificate` enthalten, wenn das Signatorzertifikat als qualifiziert anzusehen ist. Dazu müssen folgende Bedingungen erfüllt sein:
    1. Es muss die Zertifikatserweiterung `qcStatements` (siehe [QCert], 3.2.5) vorhanden sein.
    2. Innerhalb dieses Zertifikatserweiterung muss das Statement `esi4-qcStatement-1` (siehe [ETSIQCert], 4.2.1) vorhanden sein.

Ob weitere Elemente - wie beispielsweise das Signatorzertifikat selbst - zurückgeliefert werden, bleibt der *Bürgerkarten-Umgebung* überlassen. Konnte bei der Überprüfung kein Signatorzertifikat nach X.509 identifiziert werden, bleibt es ebenfalls der *Bürgerkarten-Umgebung* überlassen, welche Informationen über den öffentlichen Schlüssel sie zurückliefert.

Weiters enthält die Antwort getrennte Ergebnisse für die kryptographische Überprüfung der Signatur (`sl:SignatureCheck`), für die Prüfung des *Signaturmanifests* (`sl:SignatureManifestCheck`), für die Überprüfung ggf. vorhandener weiterer Manifeste (`sl:XMLDSIGManifestCheck`), sowie für die Prüfung der Signaturprüfdaten (`sl:CertificateCheck`).

Alle vier Ergebnisse besitzen eine ähnliche Struktur: Das Element `sl:Code` enthält das Ergebnis der Prüfung in maschinenlesbarer Form, das Element `sl:Info` kann genauereZusatzinfomationen enthalten (siehe dazu die folgenden Unterabschnitte).

3.2.2.1. Prüfung der Gültigkeit der Signatur

Diese hat wie in [XMLDSIG] spezifiziert zu erfolgen. Das bedeutet, das sowohl der Hash-Wert jedes `dsig:Reference` Elements als auch der Wert der Signatur (`dsig:SignatureValue`) zu überprüfen sind. Nicht zu überprüfen sind die Hash-Werte der Referenzelemente (`dsig:Reference`) in gegebenenfalls vorhandenen Manifesten (`dsig:Manifest`).

Wurde die in der Anfrage spezifizierte Signatur konform zu dieser Spezifikation erstellt, so enthält sie für jedes signierte Datenobjekt ein Signaturattribut `etsi:DataObjectFormat` nach [ETSIXML], das Metainformationen zum jeweiligen Datenobjekt enthält. Diese Signaturattribute können von der *Bürgerkarten-Umgebung* bei Bedarf ausgewertet werden.

Folgende Werte sind für den Inhalt des Elements `sl:Code` in `sl:SignatureCheck` definiert:

`sl:Code`	**Bedeutung**
0	Die Überprüfung der Hash-Werte und des Werts der Signatur konnte erfolgreich durchgeführt werden.
1	Bei der Überprüfung des Hash-Werts zumindest einer `dsig:Reference` der Signatur ist ein Fehler aufgetreten. Der Wert der Signatur (`dsig:SignatureValue`) wurde nicht überprüft.
2	Die Überprüfung der Hash-Werte konnte erfolgreich durchgeführt werden. Beim Überprüfen des Werts der Signatur (`dsig:SignatureValue`) ist jedoch ein Fehler aufgetreten.

Das optionale Element `sl:Info` bietet zunächst Platz für nicht näher spezifizierte Zusatzinformationen, etwa eine vom Menschen lesebare Klartext-Interpretation des Prüfungsergebnisses.

Enthält das zuvor erwähnte Element `sl:Code` den Wert `1`, wird darüber hinaus EMPFOHLEN, ein `sl:FailedReference`-Elemente als Kind von `sl:Info` zur Kennzeichnung des ersten `dsig:Reference`-Elements anzugeben, bei dem die Überprüfung des Hash-Wertes einen Fehler ergeben hat. Der Inhalt von `sl:FailedReference`, eine positive Ganzzahl, gibt dabei die laufende Nummer der fehlerhaften `dsig:Reference` innerhalb von `dsig:SignedInfo` an. Zusätzliche `sl:FailedReference`-Elemente

DÜRFEN angegeben werden, um Fehler bei der Überprüfung weiterer `dsig:Reference`-Elemente anzuzeigen.

3.2.2.2. Prüfung des Signaturmanifests

Wurde die in der Anfrage spezifizierte Signatur konform zu dieser Spezifikation erstellt, muß eine `dsig:Reference` in `dsig:SignedInfo` auf das *Signaturmanifest* verweisen. In einem solchen Fall ist der Hash-Wert jeder `dsig:Reference` des Signaturmanifests zu überprüfen. Die Prüfung der Signaturprüfdaten darf unterbleiben, wenn bei der Prüfung der Gültigkeit der Signatur ein Fehler aufgetreten ist.

Folgende Werte sind für den Inhalt des Elements `sl:Code` in `sl:SignatureManifestCheck` definiert:

`sl:Code`	**Bedeutung**
0	Dieser Code hat eine der folgenden Bedeutungen: • Für diese Signatur ist kein Signaturmanifest notwendig. • Die Signatur enthält eine Referenz auf das notwendige Signaturmanifest. Das Signaturmanifest entspricht vom Umfang her den Anforderungen dieser Spezifikation. Für jede dsig:Reference des Signaturmanifests konnte der Hash-Wert erfolgreich überprüft werden.
1	Die Signatur enthält keine Referenz auf das notwendige Signaturmanifest.
2	Die Signatur enthält zwar eine Referenz auf das Signaturmanifest, dieses entspricht vom Umfang her jedoch nicht den Anforderungen dieser Spezifikation. Die Hash-Werte der im Signaturmanifest vorhandenen `dsig:Reference`-Elemente wurden nicht überprüft.
3	Die Signatur enthält eine Referenz auf das Signaturmanifest. Das Signaturmanifest entspricht vom Umfang her den Anforderungen dieser Spezifikation. Bei der Überprüfung des Hash-Werts zumindest einer dsig:Reference des Signaturmanifests ist jedoch ein Fehler aufgetreten.
99	Die Prüfung des Signaturmanifests wurde nicht durchgeführt, da bei der Prüfung der Gültigkeit der Signatur ein Fehler aufgetreten ist.

Das optionale Element `sl:Info` bietet zunächst Platz für nicht näher spezifizierte Zusatzinformationen, etwa eine vom Menschen lesebare Klartext-Interpretation des Prüfungsergebnisses.

Enthält das zuvor erwähnte Element `sl:Code` den Wert `3`, wird darüber hinaus empfohlen, ein bzw. mehrere `sl:FailedReference`-Elemente als Kinder von `sl:Info` zur Kennzeichnung all jener `dsig:Reference`-Elemente anzugeben, bei denen die Überprüfung des Hash-Wertes einen Fehler ergeben hat. Der Inhalt des jeweiligen `sl:FailedReference`-Elements, eine positive Ganzzahl, gibt dabei die laufende Nummer der fehlerhaften `dsig:Reference` innerhalb des Signaturmanifests an.

3.2.2.3. Prüfung weiterer Manifeste

Enthält die zu überprüfende Signatur in `dsig:SignedInfo` `dsig:Reference`-Elemente, die auf gewöhnliche Manifeste entsprechend [XMLDSIG] verweisen (d. h. das Attribut `Type` in `dsig:Reference` ist auf den Wert `http://www.w3.org/2000/09/xmldsig#Manifest` gesetzt), muss die Antwort für jedes auf diese Weise identifizierte Manifest ein Element `sl:XMLDSIGManifestCheck` mit dem Ergebnis der Überprüfung des Manifests enthalten. Die Prüfung weiterer Manifeste darf unterbleiben, wenn bei der Prüfung der Gültigkeit der Signatur ein Fehler aufgetreten ist.

Folgende Werte sind für den Inhalt des Elements `sl:Code` in `sl:XMLDSIGManifestCheck` definiert:

`sl:Code`	**Bedeutung**
0	Für jede `dsig:Reference` des mittels `sl:Info` näher gekennzeichneten Manifests konnte der Hash-Wert erfolgreich überprüft werden.
1	Für zuminest eine `dsig:Reference` des mittels `sl:Info` näher gekennzeichneten Manifests konnte der Hash-Wert nicht erfolgreich überprüft werden.
99	Die Prüfung weiterer Manifeste wurde nicht durchgeführt, da bei der Prüfung der Gültigkeit der Signatur ein Fehler aufgetreten ist.

Das verpflichtend anzugebende Element `sl:Info` bietet zunächst Platz für nicht näher spezifizierte Zusatzinformationen, etwa eine vom Menschen lesebare Klartext-Interpretation des Prüfungsergebnisses.

Enthält das zuvor erwähnte Element `sl:Code` den Wert `1`, MÜSSEN darüber hinaus ein bzw. mehrere `sl:FailedReference`-Elemente als Kinder von `sl:Info` zur Kennzeichnung all jener `dsig:Reference`-Elemente des Manifests angeben werden, bei denen die Überprüfung des Hash-Wertes einen Fehler ergeben hat. Der Inhalt des jeweiligen `sl:FailedReference`-Elements, eine positive Ganzzahl, gibt dabei die laufende Nummer der fehlerhaften `dsig:Reference` innerhalb des Manifests an.

Weiters MUSS in `sl:Info` mittels des Elements `sl:ReferringSignatureReference` angegeben werden, auf welches in der Signatur referenzierte Manifest sich das Prüfungsergebnis bezieht. Der Inhalt von `sl:ReferringSignatureReference`, eine positive Ganzzahl, gibt dabei die laufende Nummer jener `dsig:Reference` innerhalb von `dsig:SignedInfo` an, welche auf das Manifest verweist.

3.2.2.4. Prüfung der Signaturprüfdaten

Diese umfaßt die Konstruktion der Zertifikatskette vom Signatorzertifikat bis zu einem vertrauenswürdigen Wurzelzertifikat, sowie die Statusprüfung für jedes Zertifikat der konstruierten Zertifikatskette. Die Prüfung der Signaturprüfdaten DARF unterbleiben, wenn bei der Prüfung der Gültigkeit der Signatur ein Fehler aufgetreten ist.

Wurde die in der Anfrage spezifizierte Signatur konform zu dieser Spezifikation erstellt, so enthält sie ein Signaturattribut `etsi:SigningCertificate` nach [ETSIXML], das Informationen zum Signatorzertifikat enthält. Diese Informationen müssen von der *Bürgerkarten-Umgebung* in einem solchen Fall als Ausgangspunkt zur Konstruktion der Zertifikatskette verwendet werden.

Bei der Überprüfung älterer Signaturen kann es vorkommen, dass vom Zertifizierungsdiensteanbieter, der das Signatorzertifikat ausgestellt hat, keinerlei Online-Informationen mehr angeboten werden, die die Bildung der Zertifikatskette bzw. die Statusprüfung der Zertifikate in dieser Kette ermöglichen. Damit solche Signaturen trotzdem überprüft werden können, wird EMPFOHLEN, dass die *Bürgerkarten-Umgebung* in solchen Fällen die ggf. vorhandenen unsignierten Signaturattribute `etsi:CompleteCertificateRefs`, `etsi:CompleteRevocationRefs`, `etsi:CertificateValues` und `etsi:RevocationValues` nach [ETSIXML] auswertet.

Für die definierten Werte für den Inhalt des Elements `sl:Code` in `sl:CertificateCheck` siehe Kapitel Abschnitt 3.1.2.2, „Prüfung der Signaturprüfdaten“ im Abschnitt über die Prüfung einer [CMS]-Signatur.

> **Anmerkung**
>
> Für Beispiele zu diesem Befehl siehe Tutorium; für Anforderungen an die Benutzerschnittstelle siehe Abschnitt 3.2, „Signaturprüfung“ in Die österreichische Bürgerkarte - Anforderungen an die Benutzer-Schnittstelle.

4. Verschlüsselung

Die in diesem Abschnitt spezifizierten Befehle ermöglichen die Verschlüsselung eines der *Bürgerkarten-Umgebung* bekanntgegebenen Dokuments (oder auch mehrerer Dokumente, abhängig vom Nachrichtenformat). Die Verschlüsselung erfolgt je Empfänger mit einem von der *Applikation* angegebenen öffentlichen Schlüssel.

4.1. Verschlüsselung als CMS-Nachricht

Dieser Befehl ermöglicht die Verschlüsselung von Daten als CMS-Nachricht mit *EnvelopedData* Inhalt nach dem Standard [CMS].

4.1.1. Anfrage

Zunächst enthält die Anfrage einen öffentlichen Schlüssel (`sl:RecipientPublicKey`) mit pro Empfänger, für den die Daten verschlüsselt werden sollen. Der öffentliche Schlüssel ist in `sl:RecipientPublicKey` als base64-kodiertes X.509-Zertifikat (`sl:X509Certificate`) enthalten.

Weiters werden mit `sl:ToBeEncrypted` die zu verschlüsselnden Daten sowie Metainformationen zu diesen Daten bekanntgegeben.

Jedenfalls an Metainformationen spezifiziert werden MUSS der Mime-Type (siehe [MIME]) der zu verschlüsselnden Daten (`sl:ToBeEncrypted/sl:MetaInfo/sl:MimeType`); diese Information benötigt die *Bürgerkarten-Umgebung* im Rahmen der gegebenenfalls vorzunehmenden Anzeige der zu verschlüsselnden Daten. Weiters DARF eine verbale Beschreibung dieser Daten angegeben werden (`sl:ToBeEncrypted/sl:MetaInfo/sl:Description`).

Für die Angabe der eigentlichen zu verschlüsselnden Daten stehen zwei Möglichkeiten zur Verfügung:

- Die *Bürgerkarten-Umgebung* soll die zu verschlüsselnden Daten von einem angegebenen Ort auflösen: `sl:ToBeEncrypted/sl:Content/@Reference` enthält die entsprechende URL, der Inhalt von `sl:ToBeEncrypted` bleibt leer.
- Die zu verschlüsselnden Daten werden direkt an die *Bürgerkarten-Umgebung* übergeben:

`sl:ToBeEncrypted/sl:Content` enthält die Daten in base64-kodierter Form, `sl:ToBeEncrypted/sl:Content/@Reference` wird nicht verwendet.

Das Attribut `sl:EncryptCMSRequest/@ReturnBinaryResult` kann schlielich optional angegeben werden, um die Art der Antwort auf die Anfrage zu steuern (vgl. Abschnitt 4.1.2, „Antwort“).

4.1.2. Antwort

Die *Bürgerkarten-Umgebung* erstellt aus den Angaben der Anfrage eine CMS-Nachricht mit *EnvelopedData* Inhalt. Dabei MUSS sie folgende Vorgaben einhalten:

- Pro in der Anfrage angegebenen öffentlichem Schlüssel ist ein Feld vom Typ *KeyTransRecipientInfo* zu erzeugen. Es muss also immer *Key Transport* als Technik zum Schlüssel-Management verwendet werden.
- Die verschlüsselten Daten sind mittels des Feldes *encryptedContent* im Typ *EncryptedContentInfo* in die CMS-Nachricht zu integrieren. Eine Trennung von Nachricht und Daten ist nicht vorgesehen.
- Die verschlüsselten Daten sind mittels des Feldes *contentType* im Typ *EncryptedContentInfo* als *Data* (siehe [CMS], Abschnitt 4) zu kennzeichnen.

Die so erzeugte CMS-Nachricht wird von der *Bürgerkarten-Umgebung* in der Antwort zurückgeschickt. Die grundsätzliche Gestalt der Antwort ist abhängig vom Attribut `sl:EncryptCMSRequest/@ReturnBinaryResult` der Anfrage.

4.1.2.1. XML-Antwort

Ist das Attribut `sl:EncryptCMSRequest/@ReturnBinaryResult` entweder nicht oder auf den Wert `false` gesetzt, liefert die *Bürgerkarten-Umgebung* eine XML-Antwort zurück.

`sl:EncryptCMSResponse/sl:CMSMessage` enthält die erzeugte CMS-Nachricht in base64-kodierter Form.

4.1.2.2. Binäre Antwort

Ist das Attribut `sl:EncryptCMSRequest/@ReturnBinaryResult` hingegen auf den Wert `true` gesetzt, liefert die *Bürgerkarten-Umgebung* keine XML-Antwort, sondern direkt und ohne weitere Kodierung die CMS-Nachricht als Antwort auf die Anfrage zurück.

Bietet das Transportprotokoll die Möglichkeit, den *Mime-Type* zu spezifizieren, MUSS die *Bürgerkarten-Umgebung* den Mime-Type auf `application/pkcs7-mime; smime-type=enveloped-data` setzen.

> **Anmerkung**
>
> Für Beispiele zu diesem Befehl siehe Tutorium; für Anforderungen an die Benutzerschnittstelle siehe Abschnitt 3.3, „Verschlüsselung“ in Die österreichische Bürgerkarte - Anforderungen an die Benutzer-Schnittstelle.

4.2. Verschlüsselung als XML-Dokument

Dieser Befehl ermöglicht die Verschlüsselung von Daten als XML-Dokument nach dem Standard [XMLEnc].

4.2.1. Anfrage

Zunächst enthält die Anfrage einen öffentlichen Schlüssel (`sl:RecipientPublicKey`) pro Empfänger, für den die Daten verschlüsselt werden sollen. Der öffentliche Schlüssel ist in `sl:RecipientPublicKey` entweder direkt (`ds:KeyValue`) oder als base64-kodiertes X.509-Zertifikat (`sl:X509Certificate`) enthalten.

Weiters werden mit `sl:ToBeEncrypted` die zu verschlüsselnden Daten bekanntgegeben. Dieses Element muss zumindest einmal vorhanden sein, und enthält Informationen über die zu verschlüsselnden Daten auf eine von drei möglichen Arten:

- `sl:Element`: Ein Element eines bestehenden XML-Dokuments wird von der *Bürgerkarten-Umgebung* verschlüsselt und durch ein entsprechendes Element `xenc:EncryptedData` ersetzt. `sl:Element/@Selector` enthält einen Ausdruck nach [XPath], mit dem das zu verschlüsselnde Element innerhalb des bestehenden XML-Dokuments eindeutig ausgewählt wird. Der Kontextknoten für die Evaluierung des Ausdrucks ist dabei der Dokument-Knoten des XML-Dokuments. Namenraum-

Präfixe, die im Ausdruck verwendet werden, müssen im Kontext von `sl:Element/@Selector` bekannt sein. Wird `sl:Element/@EncDataReference` angegeben, DÜRFEN die verschlüsselten Daten von der *Bürgerkarten-Umgebung* NICHT base64-kodiert in das entsprechende Element `xenc:EncrpytedData` aufgenommen, sondern MÜSSEN stattdessen referenziert werden. Als Referenz MUSS dabei die in `sl:Element/@EncDataReference` angegebene URI verwendet werden.

Anmerkung

Wird `sl:Element` verwendet, muss ein bestehendes XML-Dokument angegeben werden (siehe unten).

- `sl:ElementContent`: Der Inhalt eines Elements eines bestehenden XML-Dokuments wird von der *Bürgerkarten-Umgebung* verschlüsselt und durch ein entsprechendes Element `xenc:EncryptedData` ersetzt. Die Bedeutung der Attribute `sl:ElementContent/@Selector` und `sl:ElementContent/@EncDataReference` ist die gleiche wie jene der entsprechenden Attribute in `sl:Element`.

  Anmerkung

  Wird diese `sl:ElementContent` verwendet, muss ein bestehendes XML-Dokument angegeben werden (siehe unten).

- `sl:New`: Beliebige Daten sollen verschlüsselt und als Element `xenc:EncryptedData` in ein bestehendes oder neues XML-Dokument eingefügt werden. Der Ort, an dem das erzeugte `xenc:EncryptedData` in das XML-Dokument eingefügt werden soll, wird durch die Attribute `sl:New/@ParentSelector` und `sl:New/@NodeCount` angegeben. `sl:New/@ParentSelector` selektiert das Eltern-Element des zukünftigen `xenc:EncryptedData` mittels eines Ausdrucks nach [XPath]; der Kontextknoten für die Evaluierung des Ausdrucks ist dabei der Dokument-Knoten des XML-Dokuments. Namenraum-Präfixe, die im Ausdruck verwendet werden, müssen im Kontext von `sl:New/@ParentSelector` bekannt sein. `sl:New/@NodeCount` enthält die Angabe, als wievielter Knoten `xenc:EncryptedData` ins Eltern-Element einzufügen ist, wobei mit Null zu zählen begonnen wird. Die zu verschlüsselnden Daten sowie Metainformationen dazu werden als Inhalt von `sl:New` angegeben. Jedenfalls an Metainformationen spezifiziert werden MUSS der Mime-Type (siehe [MIME]) der zu verschlüsselnden Daten (`sl:New/sl:MetaInfo/sl:MimeType`); diese Information benötigt die *Bürgerkarten-Umgebung* im Rahmen der gegebenenfalls vorzunehmenden Anzeige der zu verschlüsselnden Daten. Weiters DARF eine verbale Beschreibung dieser Daten angegeben werden (`sl:New/sl:MetaInfo/sl:Description`). Für die Angabe der eigentlichen zu verschlüsselnden Daten stehen drei Möglichkeiten zur Verfügung: `sl:New/sl:Content/sl:Base64Content` enthält die zu verschlüsselnden Daten in base64-kodierter Form; diese MÜSSEN von der *Bürgerkarten-Umgebung* dekodiert werden, bevor die Verschlüsselung durchgeführt wird. `sl:New/sl:Content/sl:XMLContent` enthält die zu verschlüsselnden Daten als XML-Fragment. `sl:new/sl:Content/sl:LocRefContent` enthält eine Referenz, die von der *Bürgerkarten-Umgebung* aufgelöst werden MUSS, um zu den zu verschlüsselnden Daten zu gelangen. Die Bedeutung des Attributs `sl:New/sl:Content/@EncDataReference` ist die gleiche wie jene des entsprechenden Attributs in `sl:Element`.

  Anmerkung

  Wird sl:New verwendet, jedoch kein bestehendes XML-Dokument angegeben (siehe unten),MUSS die Bürgerkarten-Umgebung ein neues XML-Dokument erzeugen. sl:New/@ParentSelector muss in diesem Fall den Dokument-Knoten auswählen (z.B. mit /), sl:New/@NodeCount den Wert 0 enthalten.

`sl:EncryptionInfo` kann verwendet werden, um Angaben zu einem bestehenden XML-Dokument zu machen, in das die erzeugten Elemente `xenc:EncryptedData` eingefügt werden sollen (siehe oben). Wird `sl:EncryptionInfo` nicht angegeben, darf die Anfrage nur ein einziges Element `sl:ToBeEncrypted` enthalten; der Inhalt dieses einzigen Elements muss `sl:New` sein.

Mit `sl:EncryptionInfo/sl:EncryptionEnvironment` wird das XML-Dokument spezifiziert. Dazu stehen drei Möglichkeiten zur Verfügung: Soll das XML-Dokument referenziert und diese Referenz durch die *Bürgerkarten-Umgebung* aufgelöst werden, ist das Attribut `sl:EncryptionEnvironment/@Reference` zu verwenden. Der Inhalt von `sl:EncryptionEnvironment` bleibt in diesem Fall leer. Soll das XML-Dokument explizit angegeben werden, ist als Inhalt von `sl:EncryptionEnvironment` entweder `sl:Base64Content` oder `sl:XMLContent` zu verwenden; das Attribut `sl:EncryptionEnvironment/@Reference` bleibt in diesem Fall leer. `sl:Base64Content` enthält das XML-Dokument in base64-kodierter Form. `sl:XMLContent` enthält das XML-Dokument direkt.

Wird das Element `sl:EncryptionInfo/sl:EncryptedKeyLocation` angegeben, MUSS die *Bürgerkarten-Umgebung* die Informationen zu allen in der Anfrage spezifizierten öffentlichen Schlüsseln der Empfänger (`sl:RecipientPublicKey`) in ein bzw. mehrere Elemente `xenc:EncryptedKey` zusammenfassen, die aus allen erzeugten Elementen `xenc:EncryptedData` referenziert werden MÜSSEN (für genauere Informationen zum Aufbau von `xenc:EncryptedKey` bzw. zur Referenzierung siehe Abschnitt 4.2.2, „Antwort“). `sl:EncryptedKeyLocation/@ParentSelector` selektiert das Eltern-Element des einzufügenden bzw. der einzufügenden `xenc:EncryptedKey` Elemente. `sl:EncryptedKeyLocation/@NodeCount` enthält die Angabe, als wievielter Knoten das erste `xenc:EncryptedKey` Element ins Eltern-Element einzgefügt werden MUSS, wobei mit Null zu zählen begonnen wird. Weitere `xenc:EncryptedKey` Elemente MÜSSEN ggf. unmittelbar nach dem ersten Element eingefügt werden. Wird das Element nicht angegeben, muss die *Bürgerkarten-Umgebung* die Informationen zu allen in der Anfrage spezifizierten öffentlichen Schlüsseln der Empfänger in jedes einzelne erzeugte Element `xenc:EncryptedData` direkt als `xenc:EncrpytedKey` kodieren (für genauere Informationen zum Aufbau von `xenc:EncryptedKey` siehe Abschnitt 4.2.2, „Antwort“).

Zusätzlich KÖNNEN schließlich Ergänzungsobjekte (`sl:EncryptedInfo/sl:Supplement`) spezifiziert werden, die in Zusammenhang mit dem bestehenden XML-Dokument stehen.

Beispielsweise könnte im XML-Dokument, das in `sl:EncryptionEnvironment` spezifiziert wurde, ein Verweis auf die Dokumenttypdefinition enthalten sein. Das auf diese Weise referenzierte Dokument kann von der *Applikation* als Ergänzungsobjekt übergeben werden, wenn die *Bürgerkarten-Umgebung* den Verweis nicht selbst auflösen soll, oder wenn die *Bürgerkarten-Umgebung* den Verweis gar nicht auflösen kann, weil es sich etwa um einen relativen Veweis, bezogen auf das XML-Dokument, handelt.

Ein weiteres Beispiel für die Verwendung eines Ergänzungsobjekts könnte ein XML-Schema sein, das im mittels `sl:EncryptionEnvironment` spezifizierten XML-Dokument unter Verwendung der in [XML-Schema] vorgeschlagenen Mechanismen referenziert wird.

Ein Ergänzungsobjekt besteht dabei einerseits aus OPTIONALEN Metainformationen (vergleiche `sl:FinalDataMetaInfo` in „Transformationswege“), andererseits aus dem eigentlichen Daten (`sl:Content`): Das verpflichtend zu verwendende Attribut `sl:Content/@Reference` enthält dabei als URI die Referenz auf die Ergänzungsdaten, und zwar so, wie sie von der *Bürgerkarten-Umgebung* zur Auflösung verwendet werden würde. Der Inhalt von `sl:Content` repräsentiert die Ergänzungsdaten (siehe auch Hinweis Content in „Datenobjekt“).

4.2.2. Antwort

Für jede in der Anfrage enthaltene Anweisung, Daten zu verschlüsseln (`sl:ToBeEncrypted`) MUSS die *Bürgerkarten-Umgebung* ein Element `xenc:EncryptedData` erzeugen. Folgende Bedingungen MÜSSEN eingehalten werden:

- Soll laut Anfrage ein Element eines bestehenden XML-Dokuments oder der Inhalt eines solchen Elements verschlüsselt werden (`sl:ToBeEncrypted/sl:Element` bzw. `sl:ToBeEncrypted/sl:ElementContent`), MUSS das Attribut `xenc:EncryptedData/@Type` auf den Wert `http://www.w3.org/2001/04/xmlenc#Element` bzw. `http://www.w3.org/2001/04/xmlenc#Content` gesetzt werden. Ansonsten DARF dieses Attribut NICHT verwendet werden.
- In `xenc:EncryptedData/xenc:EncryptionMethod` MUSS einer der in Abschnitt 7.1.1, „Algorithmen“ in Die österreichische Bürgerkarte - Minimale Umsetzung des Security-Layers gelisteten Algorithmen zur Datenverschlüsselung angegeben werden.
- Das Element `xenc:EncryptedData/ds:KeyInfo` MUSS von der *Bürgerkarten-Umgebung* verwendet werden und alle in der Anfrage angegebenen öffentlichen Schlüssel referenzieren (falls in der Anfrage das Element `sl:EncryptionInfo/sl:EncryptedKeyLocation` verwendet wurde) bzw. enthalten (sonst).

  Im Falle der direkten Beinhaltung MUSS die *Bürgerkarten-Umgebung* in `ds:KeyInfo` pro in der Anfrage angegebenen öffentlichen Schüssel ein `xenc:EncrypedKey` Element verwenden. Dabei müssen folgende Bedingungen eingehalten werden:

  - In `xenc:EncryptedKey/xenc:EncryptionMethod` MUSS einer der in Abschnitt 7.2.1.2, „Algorithmen für `xenc:EncryptedKey`“ in Die österreichische Bürgerkarte - Minimale Umsetzung des Security-Layers gelisteten Algorithmen angegeben werden.
  - Das Element `xenc:EncryptedData/ds:KeyInfo` MUSS von der *Bürgerkarten-Umgebung* wie folgt verwendet werden:
    - Enthält `xenc:EncryptedKey/xenc:EncryptionMethod` einen Algorithmus zur asymmetrischen Verschlüsselung (Schlüsseltransport), so MUSS

`xenc:EncryptedData/ds:KeyInfo` genau ein `ds:KeyValue` Element mit der Kodierung des öffentlichen Schlüssels enthalten.

- Enthält `xenc:EncryptedKey/xenc:EncryptionMethod` einen Algorithmus zur symmetrischen Verschlüsselung (Symetric Key Wrap), so MUSS `xenc:EncryptedData/ds:KeyInfo` genau ein `xenc:AgreementMethod` Element mit einem der in Abschnitt 7.2.1.3, „Algorithmen für `xenc:KeyAgreement`“ in Die österreichische Bürgerkarte - Minimale Umsetzung des Security-Layers gelisteten Algorithmen zur Schlüsselvereinbarung enthalten. Das Element `xenc:AgreementMethod/ds:DigestMethod` MUSS einen unter Abschnitt 5.1.1, „Digest-Algorithmen“ in Die österreichische Bürgerkarte - Minimale Umsetzung des Security-Layers angegebenen Algorithmus enthalten. Das Element `xenc:AgreementMethod/xenc:RecipientKeyInfo` MUSS genau ein Element `ds:KeyValue` mit der Kodierung des öffentlichen Schlüssels enthalten. Das Element `xenc:AgreementMethod/xenc:OriginatorKeyInfo` MUSS genau ein Element `ds:KeyValue` mit der Kodierung des öffentlichen Schlüssels enthalten.

- `xenc:EncryptedData/xenc:CipherData/xenc:CipherValue` enthält den verschlüsselten symmetrischen Schlüssel zur Datenverschlüsselung.

Im Falle der Referenzierung MUSS die *Bürgerkarten-Umgebung* in `ds:KeyInfo` pro in der Anfrage angegebenen öffentlichen Schüssel ein `ds:RetrievalMethod` Element verwenden. Dabei müssen folgende Bedingungen eingehalten werden:

- Das Attribut `ds:RetrievalMethod/@Type` muss auf den Wert `http://www.w3.org/2001/04/xmlenc#EncryptedKey` gesetzt werden.
- Das Attribut `ds:RetrievalMethod/@URI` muss auf das referenzierte `xenc:EncryptedKey` Element verweisen.
- Für das referenzierte `xenc:EncryptedKey` Element gelten die Bedingungen für die direkte Beinhaltung.

- Das Element `xenc:EncryptedData/xenc:CipherData` muss die mit dem symmetrischen Verschlüsselungsschlüssel verschlüsselten Daten referenzieren (wenn im entsprechenden `sl:ToBeEncrypted` Element der Anfrage wurde in `sl:Element` bzw. `sl:ElementContent` bzw. `sl:New` das Attribut `EncDataReference` gesetzt wurde) bzw. direkt beinhalten (sonst). Im Falle der Referenzierung muss die *Bürgerkarten-Umgebung* das Element `xenc:CipherData/xenc:CipherReference` verwenden.Dabei müssen folgende Bedingungen eingehalten werden:
  - Transformationen dürfen nicht verwendet werden.
  - `xenc:CipherReference/@URI` muss auf den in `sl:Element/@EncDataReference` bzw. `sl:ElementContent/@EncDataReference` bzw. `sl:New/@EncDataReference` angegebenen Wert gesetzt werden.
  - Die verschlüsselten Daten sind in der Antwort in einem `sl:EncryptedData` Element zurückzuliefern. `sl:EncryptedData/@EncDataReference` muss auf den in `sl:Element/@EncDataReference` bzw. `sl:ElementContent/@EncDataReference` bzw. `sl:New/@EncDataReference` angegebenen Wert gesetzt werden.

Wurde in der Anfrage ein XML-Dokument angegeben (`sl:EncryptionInfo/sl:EncryptionEnvironment`), MUSS die *Bürgerkarten-Umgebung* dieses XML-Dokument mit den folgenden Modifikationen in der Antwort als Inhalt von `sl:EncryptXMLResponse/sl:EncryptionEnvironment` zurückliefern:

- Für jedes in der Anfrage spezifizierte, zu verschlüsselnde XML-Element (`sl:ToBeEncrypted/sl:Element`) MUSS die *Bürgerkarten-Umgebung* das angegebene Element des XML-Dokuments durch das entsprechend erzeugte `xenc:EncryptedData` Element ersetzen.
- Für jeden in der Anfrage spezifzierten, zu verschlüsselnden Inhalt eines XML-Elements (`sl:ToBeEncrypted/sl:ElementContent`) MUSS die *Bürgerkarten-Umgebung* den Inhalt des angegebenen Elements des XML-Dokuments durch das entsprechend erzeugte `xenc:EncryptedData` Element ersetzen.
- Für jeden in der Anfrage spezifizierten, zu verschlüsselnden neuen Inhalt (`sl:ToBeEncrypted/sl:New`) MUSS die *Bürgerkarten-Umgebung* das entsprechend erzeugte `xenc:EncryptedData` Element an der spezifizierten Position (`sl:New/@ParentSelector`, `sl:New/@NodeCount`) in das XML-Dokument einfügen.
- Wurde in der Anfrage das Element `sl:EncryptionInfo/sl:EncryptedKeyLocation` verwendet,

MUSS die *Bürgerkarten-Umgebung* an der darin spezifzizierten Position `xenc:EncryptedKey` Elemente für allen in der Anfrage spezifizierten öffentlichen Schlüsseln der Empfänger einfügen.

Wurde in der Anfrage kein XML-Dokument angegeben, MUSS die *Bürgerkarten-Umgebung* als Inhalt von `sl:EncryptXMLResponse/sl:EncryptionEnvironment` das für den in der Anfrage spezifizierten, zu verschlüsselnden neuen Inhalt (`sl:ToBeEncrypted/sl:New`) erzeugte `xenc:EncryptedData` Element zurückliefern.

Kann die *Bürgerkarten-Umgebung* die Verschlüsselung für zumindest ein in der Anfrage spezifiziertes Datum (`sl:ToBeEcnrypted`) nicht durchführen, MUSS die *Bürgerkarten-Umgebung* eine Fehler-Antwort (`sl:ErrorResponse`) liefern.

Anmerkung

Für Beispiele zu diesem Befehl siehe Tutorium; für Anforderungen an die Benutzerschnittstelle siehe Abschnitt 3.3, „Verschlüsselung“ in Die österreichische Bürgerkarte - Anforderungen an die Benutzer-Schnittstelle.

5. Entschlüsselung

Die in diesem Abschnitt spezifizierten Befehle ermöglichen die Entschlüsselung eines an die *Bürgerkarten-Umgebung* übermittelten, in verschlüsselter Form vorliegenden Dokuments mittels eines in der *Bürgerkarten-Umgebung* verwalteten privaten Schlüssels.

5.1. Entschlüsselung einer CMS-Nachricht

Dieser Befehl ermöglicht die Entschlüsselung von CMS-Nachrichten mit *EnvelopedData* Inhalt, die nach dem Standard [CMS] vorliegen. Der Entschlüsselungs-Schlüssel muss dazu ein privater Schlüssel sein, der von der *Bürgerkarten-Umgebung* verwaltet wird.

5.1.1. Anfrage

Zunächst enthält die Anfrage `sl:DecryptCMSRequest` im Element `sl:CMSMessage` die CMS-Nachricht mit *Enveloped Data* Inhalt, welche entschlüsselt werden soll.

Optional können im Element `sl:EncryptedContent` Angaben zu den in *EnvelopedData* verschlüsselten Daten gemacht werden:

- In `sl:EncryptedContent/sl:MetaInfo/sl:MimeType` kann die *Applikation* den *Mime-Type* der verschlüsselten Daten angeben; diese Information MUSS von der *Bürgerkarten-Umgebung* im Falle einer Binäranwort zur Angabe des *Mime-Types* verwendet werden, falls dies im verwendeten Transportprotokoll vorgesehen ist.
- In `sl:EncryptedContent/sl:Content` MUSS die *Applikation* den Wert des Feldes *encryptedContent* (also die verschlüsselten Daten) angeben, wenn dieses in der zu entschlüsselnden CMS-Nachricht nicht enthalten ist. Sie kann dazu entweder `sl:Content/@Reference` oder `slContent/sl:Base64Content` verwenden. `sl:Content/@Reference` enthält eine URL, welche die *Bürgerkarten-Umgebung* zur Erlangung der verschlüsselten Daten auflösen muss. `slContent/sl:Base64Content` enthält die verschlüsselten Daten in base64-kodierter Form.

Was die Kennzeichnung des Datenverschlüsselungsschlüssels betrifft, darf die *Bürgerkarten-Umgebung* von folgenden Voraussetzungen ausgehen, die von der *Applikation* eingehalten werden MÜSSEN:

- Der öffentliche Schlüssel eines in der *Bürgerkarten-Umgebung* vorhandenen und zur Verschlüsselung geeigneten Schlüsselpaares wurde zur Verschlüsselung eines symmetrischen Schlüssels verwendet, mit dem dann die eigentliche Verschlüsselung durchgeführt wurde.
- Dieser öffentliche Schlüssel ist im Feld *KeyTransRecipientInfo* in der CMS-Nachricht bezeichnet, und zwar entweder mittels Kombination aus *Issuer* und *SerialNumber* oder mittels *SubjectKeyIdentifier* des zugehörigen Zertifikats.

Für detaillierte Informationen, welche Algorithmen und URL-Protokolle eine *Bürgerkarten-Umgebung* im Zusammenhang mit der Entschlüsselung einer CMS-Nachricht beherrschen muss, siehe *Abschnitt 6.2, „Entschlüsselung“ in Die österreichische Bürgerkarte - Minimale Umsetzung des Security-Layers* .

Das Attribut `sl:DecryptCMSRequest/@ReturnResult` kann schlie lich optional angegeben werden, um die Art der Antwort auf die Anfrage zu steuern (vgl. Abschnitt 5.1.2, „Antwort“).

5.1.2. Antwort

Die grundsätzliche Gestalt der Antwort ist abhängig vom Attribut `sl:DecryptCMSRequest/@ReturnResult` der Anfrage.

5.1.2.1. XML-Antwort

Ist das Attribut `sl:DecryptCMSRequest/@ReturnResult` entweder nicht oder auf den Wert `xml` gesetzt, liefert die *Bürgerkarten-Umgebung* eine XML-Antwort zurück.

`sl:DecryptCMSResponse/sl:DecryptedData` enthält die mit Hilfe des in der CMS-Nachricht bezeichneten und in der *Bürgerkarten-Umgebung* vorhandenen Schlüsselpaares entschlüsselten Daten in base64-kodierter Form.

5.1.2.2. Binäre Antwort

Ist das Attribut `sl:DecryptCMSRequest/@ReturnResult` hingegen auf den Wert `binary` gesetzt, liefert die *Bürgerkarten-Umgebung* keine XML-Antwort, sondern direkt und ohne weitere Kodierung die entschlüsselten Daten als Antwort auf die Anfrage zurück.

Bietet das Transportprotokoll die Möglichkeit, den *Mime-Type* zu spezifizieren, MUSS die *Bürgerkarten-Umgebung* die gegebenenfalls in der Anfrage gemachten Angaben (`sl:EncryptedContent/sl:MetaInfo/sl:MimeType`) verwenden.

5.1.2.3. Leere Antwort

Ist das Attribut `sl:DecryptCMSRequest/@ReturnResult` schließlich auf den Wert `none` gesetzt, liefert die *Bürgerkarten-Umgebung* eine leere Antwort (Element `sl:DecryptCMSResponse` ohne Inhalt) zurück. Mit dieser Option hat der *Bürger* die Möglichkeit, sich die entschlüsselten Daten über die *Benutzer-Schnittstelle* anzusehen und/oder zu speichern, ohne das sie im weiteren auch an die *Applikation* übermittelt werden.

> **Anmerkung**
>
> Für Beispiele zu diesem Befehl siehe Tutorium; für Anforderungen an die Benutzerschnittstelle siehe Abschnitt 3.4, „Entschlüsselung“ in Die österreichische Bürgerkarte - Anforderungen an die Benutzer-Schnittstelle.

5.2. Entschlüsselung eines XML-Dokuments

Dieser Befehl ermöglicht die Entschlüsselung von XML-Dokumenten, die entweder zur Gänze oder in Teilen nach dem Standard [XMLEnc] verschlüsselt vorliegen. Der Entschlüsselungs-Schlüssel muss dazu ein privater Schlüssel sein, der von der *Bürgerkarten-Umgebung* verwaltet wird.

5.2.1. Anfrage

Zunächst enthält die Anfrage `sl:DecryptXMLRequest` im Element `sl:EncryptedContent` Informationen zu dem nach [XMLEnc] verschlüsselten XML-Dokument, das entschlüsselt werden soll. Dieses Dokument kann entweder mittels Angabe einer URL referenziert, oder aber direkt in den Befehl eingebettet werden:

- Soll das zu entschlüsselnde XML-Dokument referenziert werden, ist für die Referenzierung das Attribut `sl:EncryptedContent/@Reference` zu verwenden. `Reference` muss eine von der *Bürgerkarten-Umgebung* auflösbare URL sein. Der Inhalt von `sl: EncryptedContent` bleibt leer.
- Soll das zu entschlüsselnde XML-Dokument hingegen eingebettet werden, geschieht dies direkt als Inhalt von `sl:EncryptedContent/sl:Base64Content` bzw. `sl:EncryptedContent/sl:XMLContent`. Wird die erste Variante verwendet, enthält `sl:Base64Content` das base64-kodierte XML-Dokument, in der zweiten Variante enthält `sl:XMLContent` als einzigen Knoten das Wurzel-Element des XML-Dokuments. `sl:EncryptedContent/@Reference` findet in beiden Fällen keine Verwendung.

Das zu entschlüsselnde XML-Dokument DARF prinzipiell beliebig viele Elemente mit verschlüsseltem Inhalt (`xenc:EncryptedData`) enthalten. Welche dieser verschlüsselten Elemente von der *Bürgerkarten-Umgebung* tatsächlich entschlüsselt werden sollen, ist mit dem Element `sl:EncrElemsSelector` anzugeben. Sein Inhalt spezifiziert einen Ausdruck nach [XPath], dessen Auswertung eine Knotenmenge mit beliebig vielen Elementen `xenc:EncryptedData` ergeben MUSS. Als Kontextknoten für die Auswertung

des Ausdrucks MUSS die *Bürgerkarten-Umgebung* den Dokument-Knoten des zu entschlüsselnden XML-Dokuments verwenden.

Was die Kennzeichnung des Datenverschlüsselungsschlüssels betrifft, darf die *Bürgerkarten-Umgebung* von einer folgenden Voraussetzungen ausgehen, die von der *Applikation* eingehalten werden MUSS:

- Als Datenverschlüsselungsschlüssel wurde direkt der öffentliche Schlüssel eines in der *Bürgerkarten-Umgebung* vorhandenen und zur Verschlüsselung geeigneten Schlüsselpaares verwendet (direkte Datenverschlüsselung).

  Der verwendete öffentliche Schlüssel muss in `xenc:EncryptedData/ds:KeyInfo` mittels `ds:KeyName`, `ds:KeyValue` oder `ds:X509Data` bezeichnet werden. `ds:KeyName` muss den Bezeichner eines in der *Bürgerkarten-Umgebung* verfügbaren Schlüssels enthalten. Bezeichner aller verfügbaren Schlüssel können mit Hilfe des Befehls Abschnitt 8.1, „Abfrage der Umgebungseigenschaften“ von der *Bürgerkarten-Umgebung* abgefragt werden. `ds:KeyValue` muss den zur Verschlüsselung verwendeten öffentlichen Schlüssel enthalten. `ds:X509Data` muss genau ein X.509-Zertifikat (`ds:X509Certificate`) enthalten, das den zur Verschlüsselung verwendeten öffentlichen Schlüssel zertifiziert.

- Ein in der *Bürgerkarten-Umgebung* vorhandener öffentlicher Schlüssel wurde zur Verschlüsselung eines symmetrischen Schlüssels verwendet, mit dem dann die eigentliche Verschlüsselung durchgeführt wurde (Schlüsseltransport / indirekte Datenverschlüsselung).

  `xenc:EncryptedData/ds:KeyInfo` muss entweder ein Element `xenc:EncryptedKey` enthalten, oder aber zumindest ein Element `ds:RetrievalMethod`, das auf ein Element `xenc:EncryptedKey` verweist. `xenc:EncryptedKey` muss die Angaben zum verwendeten symmetrischen Schlüssel enthalten und in `ds:KeyInfo` auf einem der oben erwähnten Wege den zur Verschlüsselung des symmetrischen Schlüssels verwendeten, in der *Bürgerkarten-Umgebung* verfügbaren Schlüssel bezeichnen.

- Ein in der *Bürgerkarten-Umgebung* vorhandener öffentlicher Schlüssel wurde zur Vereinbarung eines symmetrischen Schlüssels verwendet, mit dem dann die eigentliche Verschlüsselung durchgeführt wurde (Schlüsselvereinbarung / indirekte Datenverschlüsselung).

  `xenc:EncryptedData/ds:KeyInfo` muss ein Element `xenc:AgreementMethod` enthalten, oder aber zumindest ein Element `ds:RetrievalMethod`, das auf ein Element `xenc:AgreementMethod` verweist. `xenc:AgreementMethod/xenc:RecipientKeyInfo` muss auf einem der oben erwähnten Wege den zur Schlüsselvereinbarung verwendeten, in der *Bürgerkarten-Umgebung* verfügbaren Schlüssel bezeichnen.

- Ein in der *Bürgerkarten-Umgebung* vorhandener öffentlicher Schlüssel wurde zur Vereinbarung eines symmetrischen Schlüssels verwendet, mit dem der symmetrische Schlüssel, mit dem die eigentliche Verschlüsselung durchgeführt wurde, verschlüsselt ist (Schlüsselvereinbarung / Symmetric Key Wrap / indirekte Datenverschlüsselung).

  `xenc:EncryptedData/ds:KeyInfo` muss entweder ein Element `xenc:EncryptedKey` enthalten, oder aber zumindest ein Element `ds:RetrievalMethod`, das auf ein Element `xenc:EncryptedKey` verweist. `xenc:EncryptedKey` muss ein Element xenc:AgreementMethod enthalten, das im Element `xenc:AgreementMethod/xenc:RecipientKeyInfo`, auf einem der oben erwähnten Wege, den zur Schlüsselvereinbarung verwendeten, in der *Bürgerkarten-Umgebung* verfügbaren Schlüssel, bezeichnet.

Für detaillierte Informationen, welche Algorithmen, Schlüsselhinweise und URL-Protokolle eine *Bürgerkarten-Umgebung* im Zusammenhang mit der Entschlüsselung eines XML-Dokuments beherrschen muss, siehe Abschnitt 7.2, „Entschlüsselung“ in Die österreichische Bürgerkarte - Minimale Umsetzung des Security-Layers.

Optional KÖNNEN Ergänzungsobjekte (`sl:Supplements`) spezifiziert werden; das sind Datenobjekte, auf die im Verschlüsselungsdokument verwiesen wird. Beispiele dafür sind

- verschlüsselte Daten, auf die in einem Element `xenc:EncryptedData` verwiesen wird;
- ein verschlüsselter symmetrischer Schlüssel, auf den in einem Element `xenc:EncryptedKey` verwiesen wird;
- ein Element `xenc:EncryptedKey`, auf das in einem Element `ds:RetrievalMethod` verwiesen wird.

Für ein Ergänzungsobjekt (`sl:Supplement`) DÜRFEN zunächst Metainformationen (`sl:MetaInfo`) angegeben werden (Mime-Type sowie eine Referenz auf eine Beschreibung des Ergänzungsobjekts). Danach folgen die verpflichtenden Angaben zum Inhalt des Objekts (`sl:Content`): Das Attribut `Reference` enthält dabei als URI die Referenz auf das Ergänzungsobjekt, und zwar in exakt gleicher Weise, wie sie von der *Bürgerkarten-Umgebung* zur Auflösung verwendet werden würde. Der Inhalt von `sl:Content` repräsentiert

das Ergänzungsobjekt (siehe auch Hinweis Content in „Datenobjekt“). Werden in der Anfrage Ergänzungsobjekte übergeben, MUSS die *Bürgerkarten-Umgebung* diese verwenden, anstatt die entsprechenden Referenzen selbst aufzulösen.

Anmerkung

Notwendig ist die Angabe von Ergänzungsobjekten, wenn im Verschlüsselungsdokument relative Verweise vorhanden sind.

Das Attribut `sl:DecryptXMLRequest/@ReturnResult` kann schlielich optional angegeben werden, um die Art der Antwort auf die Anfrage zu steuern (vgl. Abschnitt 5.2.2, „Antwort“).

5.2.2. Antwort

Die grundsätzliche Gestalt der Antwort ist abhängig vom Attribut `sl:DecryptXMLRequest/@ReturnResult` der Anfrage.

5.2.2.1. XML-Antwort

Ist das Attribut `sl:DecryptXMLRequest/@ReturnResult` entweder nicht oder auf den Wert `xml` gesetzt, liefert die *Bürgerkarten-Umgebung* eine XML-Antwort zurück.

`sl:DecryptXMLResponse/sl:CandidateDocument` enthält dabei jenes XML-Dokument, das in der Anfrage entweder per Referenz oder direkt eingebettet an die *Bürgerkarten-Umgebung* übergeben wurde. Darin sind jedoch all jene verschlüsselten Elemente `xenc:EncryptedData` durch ihre entschlüsselten Inhalte ersetzt, für die alle nachfolgenden Bedingungen erfüllt sind:

- das verschlüsselte Element wurde mittels `sl:DecryptXMLRequest/sl:EncrElemsSelector` in der Anfrage zur Entschlüsselung ausgewählt;
- der im Attribut `xenc:EncryptedData/@Type` angegebene Typ des Verschlüsselungselements ist entweder `http://www.w3.org/2001/04/xmlenc#Element` oder `http://www.w3.org/2001/04/xmlenc#Content`.

Kaskadierte Verschlüsselungselemente werden von der *Bürgerkarten-Umgebung* nicht weiter behandelt, d. h. enthält der entschlüsselte Inhalt eines Verschlüsselungselements wiederum Verschlüsselungselemente, werden diese nicht weiter entschlüsselt.

Wählt `sl:DecryptXMLRequest/@EncrElemsSelector` auch Verschlüsselungselemente aus, deren Typ (`xenc:EncryptedData/@Type`) entweder nicht angegeben wurde, oder aber weder `http://www.w3.org/2001/04/xmlenc#Element` noch `http://www.w3.org/2001/04/xmlenc#Content` ist, ersetzen die entsprechenden entschlüsselten Inhalte nicht die Verschlüsselungselemente im XML-Dokument, sondern werden als eigene Elemente `sl:DecryptXMLResponse/sl:DecryptedBinaryData` retourniert:

- `sl:DecryptedBinaryData/@EncrElemSelector` enthält als Ausdruck nach [XPath] den Bezug zum korrespondierenden Verschlüsselungselement im XML-Dokument der Anfrage; das Verschlüsselungselement muss dabei von diesem Ausdruck eindeutig selektiert werden, d. h. er muss eine Knotenmenge liefern, die genau ein Element `xenc:EncryptedData` enthält. Der Kontextknoten für die Evaluierung des Ausdrucks ist dabei der Dokument-Knoten des XML-Dokuments der Anfrage.
- `sl:DecryptedBinaryData/@MimeType` enthält optional den Mime-Type der entschlüsselten Daten; und zwar genau dann, wenn dieser bereits im Verschlüsselungselement mittels `xenc:EncryptedData/@MimeType` angegeben wurde;
- `sl:DecryptedBinaryData/@Encoding` enthält optional die Transport-Kodierung der entschlüsselten Daten; und zwar genau dann, wenn dieser bereits im Verschlüsselungselement mittels `xenc:EncryptedData/@Encoding` angegeben wurde;
- den Inhalt von `sl:DecryptedBinaryData` schließlich bilden die entschlüsselten Daten, und zwar in base64-kodierter Form. Liegen die entschlüsselten Daten bereits von vorneherein base64-kodiert vor (d. h. weist das Attribut `xenc:EncryptedData/@Encoding` des Verschl sselungselements den Wert `http://www.w3.org/2000/09/xmldsig#base64` auf), DÜRFEN sie NICHT noch einmal base64-kodiert werden.

Kann auch nur ein einziges der mittels `sl:DecryptXMLRequest/sl:EncrElemsSelector` selektierten Verschlüsselungselemente nicht entschlüsselt werden, MUSS die *Bürgerkarten-Umgebung* eine Fehler-Antwort (`sl:ErrorResponse`) liefern. Liefert die Auswertung des XPaths in

`sl:DecryptXMLRequest/sl:EncrElemsSelector` eine leere Knotenmenge, MUSS die *Bürgerkarten-Umgebung* das unveränderte XML-Dokument der Anfrage retournieren.

5.2.2.2. Binäre Antwort

Ist das Attribut `sl:DecryptXMLRequest/@ReturnResult` hingegen auf den Wert `binary` gesetzt, liefert die *Bürgerkarten-Umgebung* keine XML-Antwort, sondern direkt Binärdaten. Zur Ermittlung dieser Binärdaten geht die *Bürgerkarten-Umgebung* wie folgt vor:

- Untersucht wird nur das erste Verschlüsselungselement, welches die Auswertung des XPath-Ausdrucks in `sl:DecryptXMLRequest/sl:EncrElemsSelector` liefert; es gilt dabei die Reihenfolge des Auftretens im XML-Dokument (*document-order*);
- das so ermittelte Verschlüsselungselement wird entschlüsselt; die sich ergebenden Binärdaten werden anstatt einer XML-Antwort direkt und ohne weitere Kodierung als Antwort auf die Anfrage zurückgeschickt.
- Bietet das Transportprotokoll die Möglichkeit, die Transport-Kodierung bzw. den *Mime-Type* zu spezifizieren, MUSS die *Bürgerkarten-Umgebung* die gegebenenfalls im Verschlüsselungselement gemachten Angaben (`xenc:EncryptedData/@MimeType` bzw. `xenc:EncryptedData/@Encoding`) verwenden.

Kann das zu untersuchende Verschlüsselungselement nicht entschlüsselt werden, oder liefert die Auswertung des XPaths in `sl:DecryptXMLRequest/@EncrElemsSelector` eine leere Knotenmenge, MUSS die *Bürgerkarten-Umgebung* eine Fehler-Antwort (`sl:ErrorResponse`) liefern.

5.2.2.3. Leere Antwort

Ist das Attribut `sl:DecryptXMLRequest/@ReturnResult` hingegen auf den Wert `none` gesetzt, liefert die *Bürgerkarten-Umgebung* eine leere Antwort (Element `sl:DecryptXMLResponse` ohne Inhalt) zurück. Mit dieser Option hat der *Bürger* die Möglichkeit, sich die entschlüsselten Daten über die *Benutzer-Schnittstelle* anzusehen und/oder zu speichern, ohne das sie im weiteren auch an die *Applikation* übermittelt werden.

> **Anmerkung**
>
> Für Beispiele zu diesem Befehl siehe Tutorium; für Anforderungen an die Benutzerschnittstelle siehe Abschnitt 3.4, „Entschlüsselung“ in Die österreichische Bürgerkarte - Anforderungen an die Benutzer-Schnittstelle.

6. Hashwerte

6.1. Hashwert-Berechnung

Dieser Befehl dient zur Berechnung von Hashwerten für ein oder mehrere in der Anfrage spezifizierte Datenobjekte.

6.1.1. Anfrage

Das Anfrage-Element `sl:CreateHashRequest` enthält pro Datenobjekt, für das die *Bürgerkarten-Umgebung* einen Hashwert berechnen soll, ein Element `sl:HashInfo`, bestehend aus den Elementen `sl:HashData`, `sl:HashAlgorithm` und (optional) `sl:FriendlyName`.

Mit `sl:HashData` spezifiziert die *Applikation* entweder eine Lokation, von der die *Bürgerkarten-Umgebung* die zu hashenden Daten abrufen soll (Verwendung des Attributs `sl:Content/@Reference`, Inhalt bleibt leer), oder sie spezifiziert direkt die zu hashenden Daten (keine Verwendung des Attributs `sl:Content/@Reference`, als Inhalt entweder `sl:Content/sl:Base64Content` oder `sl:Content/@sl:XMLContent`).

Jedenfalls an Metainformationen spezifiziert werden MUSS der Mime-Type (siehe [MIME]) der zu hashenden Daten (`sl:HashData/sl:MetaInfo/sl:MimeType`); diese Information benötigt die *Bürgerkarten-Umgebung* im Rahmen der gegebenenfalls vorzunehmenden Anzeige der zu hashenden Daten. Weiters DARF eine verbale Beschreibung dieser Daten angegeben werden (`sl:HashData/sl:MetaInfo/sl:Description`).

Wird `sl:XMLContent` zur direkten Spezifikation der zu hashenden Daten verwendet, MUSS die *Bürgerkarten-Umgebung* den XML-Inhalt unter Verwendung der Kanonisierung nach [C14N] in eine Folge von Bytes umwandeln, und diese dann der eigentlichen Hash-Berechnung zuführen. Vor der Kanonisierung

MUSS die *Bürgerkarten-Umgebung* den XML-Inhalt aus dem Kontext des Anfrage-Elements herauslösen.

Mit `sl:HashAlgorithm` wählt die *Applikation* den Hash-Algorithmus, nach dem die *Bürgerkarten-Umgebung* die Hashwertberechnung durchführen soll. `sl:FriendlyName` schließlich kann sie dazu verwenden, um den zu hashenden Daten einen für den Bürger verständlichen Namen zu geben. Dieser Name wird gegebenenfalls von der *Bürgerkarten-Umgebung* an der *Benutzer-Schnittstelle* zur Benennung der zu hashenden Daten verwendet.

Das Attribut `sl:HashInfo/@RespondHashData` gibt an, ob das Element `sl:HashData` im Antwort-Element zurückgeliefert wird.

Anmerkung

Die von einer *Bürgerkarten-Umgebung* zu unterstützenden Algorithmen zur Hashwertberechnung sind in Abschnitt 8, „Profil für Hashwerte“ in Die österreichische Bürgerkarte - Minimale Umsetzung des Security-Layers spezifiziert.

6.1.2. Antwort

Das Antwort-Element `sl:CreateHashResponse` enthält pro Datenobjekt, für das ein Hashwert zu bilden war, ein Element `sl:HashInfo`, bestehend aus den Elementen (gegebenenfalls) `sl:HashData`, `sl:HashAlgorithm`, (gegebenenfalls) `sl:FriendlyName` sowie `sl:HashValue`.

Die ersten drei Elemente entsprechen dabei exakt jenen aus dem korrespondierenden `sl:HashInfo` der Anfrage, wobei das Element `sl:HashData` nur dann aufscheint, wenn im korrespondierenden `sl:HashInfo` der Anfrage das Attribut `RespondHashData` den Wert `true` enthält. Das Element `sl:HashValue` beinhaltet den von der *Bürgerkarten-Umgebung* berechneten Hashwert über die per `sl:HashInfo` in der Anfrage angegebenen Daten, und zwar in base64-kodierter Form.

Anmerkung

Für Beispiele zu diesem Befehl siehe Tutorium; für Anforderungen an die Benutzerschnittstelle siehe Abschnitt 3.5, „Hashwert-Berechnung“ in Die österreichische Bürgerkarte - Anforderungen an die Benutzer-Schnittstelle.

6.2. Hashwert-Verifikation

Dieser Befehl dient zur Verifikation von Hashwerten über ein oder mehrere in der Anfrage spezifizierte Datenobjekte.

6.2.1. Anfrage

Das Anfrage-Element `sl:VerifyHashRequest` enthält pro Datenobjekt, für das die *Bürgerkarten-Umgebung* einen darüber gerechneten Hashwert verifizieren soll, ein Element `sl:HashInfo`, bestehend aus den Elementen `sl:HashData`, `sl:HashAlgorithm`, ggf. `sl:FriendlyName` und `sl:HashValue`.

Mit `sl:HashData` spezifiziert die *Applikation* entweder eine Lokation, von der die *Bürgerkarten-Umgebung* die Daten abrufen soll, über die der zu verifizierende Hashwert gerechnet worden ist (Verwendung des Attributs `sl:Content/@Reference`, Inhalt bleibt leer), oder sie spezifiziert diese Daten direkt (keine Verwendung des Attributs `sl:Content/@Reference`, als Inhalt entweder `sl:Content/sl:Base64Content` oder `sl:Content/sl:XMLContent`).

Jedenfalls an Metainformationen spezifiziert werden MUSS der Mime-Type (siehe [MIME]) der zu verifizierenden Hash-Daten (`sl:HashData/sl:MetaInfo/sl:MimeType`); diese Information benötigt die *Bürgerkarten-Umgebung* im Rahmen der gegebenenfalls vorzunehmenden Anzeige der zu verifizierenden Hash-Daten. Weiters DARF eine verbale Beschreibung dieser Daten angegeben werden (`sl:HashData/sl:MetaInfo/sl:Description`).

Wird `sl:XMLContent` zur direkten Spezifikation verwendet, MUSS die *Bürgerkarten-Umgebung* den XML-Inhalt unter Verwendung der Kanonisierung nach [C14N] in eine Folge von Bytes umwandeln, und diese dann der eigentlichen Hash-Verfikation zuführen. Vor der Kanonisierung MUSS die *Bürgerkarten-Umgebung* den XML-Inhalt aus dem Kontext des Anfrage-Elements herauslösen.

Mit `sl:HashAlgorithm` spezifiziert die *Applikation* den Hash-Algorithmus, nach dem der zu verifizierende Hashwert über die mittels `sl:HashData` angegebenen Daten gerechnet worden ist.

`sl:FriendlyName` kann sie dazu verwenden, um dem Datenobjekt, für das der Hashwert verifiziert werden soll, einen für den Bürger verständlichen Namen zu geben. Dieser Name wird gegebenenfalls von der *Bürgerkarten-Umgebung* an der *Benutzer-Schnittstelle* zur Benennung des Datenobjekts verwendet.

`sl:HashValue` schließlich enthält den zu verifizierenden Hashwert.

6.2.2. Antwort

Das Antwort-Element `sl:VerifyHashResponse` enthält je Hashwert, der im Anfrage-Element zur Verifikation eingereicht worden ist, ein korrespondierendes Element `sl:VerificationResult`. Dieses enthält ggf. das im korrespondierenden `sl:HashInfo` der Anfrage angegebene Element `sl:FriendlyName` sowie in `sl:Result` das Resultat der Hashwert-Verifikation. Konnte die *Bürgerkarten-Umgebung* den im korrespondierenden `sl:HashInfo` der Anfrage angegebenen Hashwert verifizieren (d. h. aus den angegebenen Daten nach dem angegebenen Hash-Algorithmus den identen Hashwert berechnen), enthält `sl:Result` den Wert `true`, ansonsten `false`.

Anmerkung

Für Beispiele zu diesem Befehl siehe Tutorium; für Anforderungen an die Benutzerschnittstelle siehe Abschnitt 3.6, „Hashwert-Verifikation“ in Die österreichische Bürgerkarte - Anforderungen an die Benutzer-Schnittstelle.

7. Zugriff auf Infoboxen

Die *Bürgerkarten-Umgebung* stellt der *Applikation* einen Datenspeicher zur Verfügung, der logisch in sogenannte Infoboxen gegliedert ist. Eine Infobox ist dabei als Datencontainer für eine Menge zusammengehöriger Daten zu sehen.

Physikalisch können sich solche Infoboxen entweder direkt auf dem Bürgerkarten-Token befinden, oder aber auch an einem anderen Platz, der unter Kontrolle der *Bürgerkarten-Umgebung* steht; beispielsweise auf der Festplatte des lokalen Hosts, auf dem die *Bürgerkarten-Umgebung* ausgeführt wird. Für die *Applikation* ist diese Unterteilung aber nicht sichtbar; ihr ist lediglich die logische Sicht der Infobox als Datencontainer in der *Bürgerkarten-Umgebung* bekannt.

Es obliegt der *Bürgerkarten-Umgebung*, den Zugriff auf Daten in einer Infobox gegebenenfalls durch geeignete Maßnahmen zu schützen. Für Daten, die auf dem Bürgerkarten-Token abgelegt werden, können etwa die dort vorhanden Schutzmechanismen (PIN, Schlüssel) verwendet werden. Ähnliche Mechanismen sind auch zum Schutz von sensiblen Daten denkbar, die von der *Bürgerkarten-Umgebung* auf der Festplatte verwaltet werden.

Die Schnittstelle *Security-Layer* bietet folgende Kommandos für den Zugriff auf Daten in Infoboxen:

- Abfrage von verfügbaren Infoboxen
- Lesen von Daten in einer Infobox
- Verändern von Daten in einer Infobox

Anmerkung

Für Beispiele zu diesem Befehl siehe Tutorium; für Anforderungen an die Benutzerschnittstelle siehe Abschnitt 3.7, „Infoboxen“ in Die österreichische Bürgerkarte - Anforderungen an die Benutzer-Schnittstelle.

7.1. Typen von Infoboxen

Nachfolgend sind jene zwei verschiedenen Typen definiert, die eine Infobox haben kann.

7.1.1. Binärdatei

Eine Binärdatei ist im Wesentlichen als eine Folge von Bytes anzusehen, die in ihrer Gesamtheit über den *Security-Layer* gelesen bzw. geschrieben werden kann. Lesen bzw. Veränderung einer Teilfolge von Bytes der Binärdatei ist nicht vorgesehen.

7.1.1.1. Leseparameter und Antwortdaten

Eine Binärdatei kann nur in ihrer Gesamtheit gelesen werden. Grundsätzlich werden daher keine Parameter in der Leseanfrage (siehe Abschnitt 7.5.1, „Anfrage“) übergeben, das Parameterelement `sl:BinaryFileParameters` bleibt leer. Die *Applikation* kann jedoch mit Hilfe des boolschen Attributs `sl:ContentIsXMLEntity` in `sl:BinaryFileParameters` der *Bürgerkarten-Umgebung* einen Hinweis geben, ob die in der Binärdatei gespeicherten Daten als XML interpretierbar sind.

In welcher Form die Daten in der Antwort auf die Leseanfrage (siehe Abschnitt 7.5.2, „Antwort“) geliefert werden, hängt vom Attribut `sl:ContentIsXMLEntity` in der Leseanfrage ab. Wurde dieses Attribut auf `true` gesetzt, enthält `sl:BinaryFileData` den Inhalt der Binärdatei als geparstes XML (als Kinder von `sl:XMLContent`), ansonsten in base64-kodierter Form (`sl:Base64Content`).

7.1.1.2. Update-Parameter und Antwortdaten

Da eine Binärdatei nur in ihrer Gesamtheit verändert werden kann, ist in der Update-Anfrage als einziger Parameter der gesamte neue Inhalt der Infobox anzugeben, und zwar entweder in base64-kodierter Form (`sl:BinaryFileParameters/sl:Base64Content`), oder als geparstes XML (`sl:BinaryFileParameters/sl:XMLContent`). Bei der Kodierung als XML kann für `sl:XMLContent` das Attribut `xml:space` mit dem Wert `preserve` versehen werden, wenn es aus Sicht der *Applikation* wichtig ist, dass die *Bürgerkarten-Umgebung* Whitespace innerhalb des übergebenen XML nicht verändert.

In der entsprechenden Antwort auf die Update-Anfrage (siehe Abschnitt 7.6.2, „Antwort“) werden keinerlei Daten zurückgeliefert.

7.1.2. Assoziatives Array

Ein assoziatives Array ist als eine Menge von Schlüsseln zu verstehen, wobei jedem Schlüssel ein entsprechender Wert zugeordnet ist. Als wichtige Einschränkung muss stets gelten, dass sämtliche Schlüssel paarweise verschieden sind. Ein Schlüssel ist stets ein XML-String, während für den zugeordneten Wert keine Einschränkungen gelten.

7.1.2.1. Leseparamter und Antwortdaten

Auf ein assoziatives Array sind drei verschiedene Lesezugriffe erlaubt:

Lesen von Schlüsseln

Der erste Lesezugriff (Element `sl:ReadKeys` in `sl:AssocArrayParameters`) liefert die Menge jener Schlüssel, die einem spezifizierten Suchstring (Attribut `SearchString` in `sl:Readkeys`) entsprechen. Mehrere Schlüssel sind als Ergebnis möglich, da im Suchstring die Angabe einer Wildcard `*` möglich ist. Diese Wildcard steht für eine beliebig lange Folge von Zeichen, mit Ausnahme des Zeichens `/`. Im gleichen Suchstring können auch mehrere Wildcards vorkommen, jedoch muss zwischen zwei Wildcards zumindest einmal das Zeichen `/` vorkommen. Für die Selektion aller verfügbaren Schlüssel ist der Wert `**` als Suchstring anzugeben.

Wird zusätzlich das Attribut `sl:ReadKeys/@UserMakesUnique` angegeben, so muss der Bürger über die *Benutzer-Schnittstelle* der *Bürgerkarten-Umgebung* genau einen Schlüssel aus der Menge der mit dem spezifizierten Suchstring gefundenen Schlüssel auswählen. Dieser vom Bürger selektierte Schlüssel ist dann als einziger Schlüssel in der Antwort zurückzusenden. Ist die Menge der mit dem spezifizierten Suchstring gefundenen Schlüssel leer, MUSS sich die *Bürgerkarten-Umgebung* so verhalten, als wäre das Attribut `sl:ReadKeys/@UserMakesUnique` gar nicht angegeben worden. Für genauere Informationen über die Vorgaben an die *Benutzer-Schnittstelle* siehe Anforderungen an die Benutzer-Schnittstelle.

> **Anmerkung**
>
> Dieses Modell für den Einsatz der Wildcard wurde gewählt, um die Nachbildung mehrdimensionaler Arrays durch das assoziative Array zu ermöglichen. So könnte etwa ein zweidimensionales Array durch Schlüssel der Form `<Zahl> '/' <Zahl>` nachgebildet werden. Mit Hilfe des Suchstrings `1/*` könnten dann etwa alle Werte der ersten Reihe ausgelesen werden.

Die Antwortdaten enthalten die gefundenen Schlüssel (Elemente `sl:Key` in `sl:AssocArrayData`).

Lesen von Schlüsseln und Werten

Der zweite Lesezugriff (Element `sl:ReadPairs` in `sl:AssocArrayParameters`) ist ähnlich dem ersten, jedoch werden nicht nur die dem Suchausdruck (Attribut `SearchString` in `sl:ReadPairs`) entsprechenden Schlüssel als Resultat geliefert, sondern zu jedem Schlüssel wird gleichzeitig auch der zugeordnete Wert retourniert.

Wird das Attribut `sl:ReadPairs/@UserMakesUnique` angegeben, so muss der Bürger über die *Benutzer-Schnittstelle* der *Bürgerkarten-Umgebung* genau einen Schlüssel aus der Menge der mit dem spezifizierten Suchstring gefundenen Schlüssel auswählen. Dieser vom Bürger selektierte Schlüssel ist dann als einziger Schlüssel gemeinsam mit dem zugehörigen Wert in der Antwort zurückzusenden. Ist die Menge der mit dem

spezifizierten Suchstring gefundenen Schlüssel leer, MUSS sich die *Bürgerkarten-Umgebung* so verhalten, als wäre das Attribut `sl:ReadPairs/@UserMakesUnique` gar nicht angegeben worden. Für genauere Informationen über die Vorgaben an die *Benutzer-Schnittstelle* siehe Anforderungen an die Benutzer-Schnittstelle.

Ähnlich wie bei den Leseparametern für eine Binärdatei (siehe Abschnitt 7.1.1.1, „Leseparameter und Antwortdaten“) kann die *Applikation* Hilfe des boolschen Attributs `ValuesAreXMLEntities` im Element `sl:ReadPairs` der *Bürgerkarten-Umgebung* einen Hinweis geben, ob die zu den gesuchten Schlüsseln zugeordneten Werte als XML interpretierbar sind. Sind nicht alle Werte als XML interpretierbar, DARF das Attribut NICHT auf `true` gesetzt werden.

Die Antwortdaten (`sl:AssocArrayData`) enthalten die gefundenen Schlüssel/Wert-Paare (Elemente `sl:Pair`). In `sl:Pair` befindet sich der Schlüssel als Attribut `Key`, sowie der zugehörige Wert als geparstes XML (`sl:XMLContent`) oder base64-kodiert (`sl:Base64Content`). Die Kodierung der Werte hängt vom oben besprochenen Attribut `ValuesAreXMLEntities` ab. Sie erfolgt entweder für alle Werte als geparstes XML oder für alle Werte base64.

Lesen des Werts zu einem Schlüssel

Der dritte Lesezugriff (Element `sl:ReadValue` in `sl:AssocArrayParameters`) erlaubt das Lesen jenes Wertes, der einem spezifizierten Schlüssel (Attribut `Key` in `sl:ReadValue`) zugeordnet ist. Wie beim Lesen von Schlüsseln und Werten kann die *Applikation* auch hier mit Hilfe des boolschen Attributs `ValueIsXMLEntity` im Element `sl:ReadValue` der *Bürgerkarten-Umgebung* einen Hinweis geben, ob der auszulesende Wert als XML interpretierbar ist.

Die Antwortdaten (`sl:AssocArrayData`) enthalten den angegebenen Schlüssel (Attribut Key in `sl:Pair`) sowie den dazugehörigen Wert (Inhalt von `sl:Pair`). Die Kodierung des Werts hängt vom oben besprochenen Attribut `ValueIsXMLEntity` ab. Sie erfolgt entweder als geparstes XML (`sl:XMLContent`) oder als base64 (`sl:Base64Content`).

7.1.2.2. Update-Parameter und Antwortdaten

Für ein Assoziatives Array sind drei unterschiedliche Update-Anfragen vorgesehen:

Änderung eines Schlüssels

Die erste Anfrage (Element `sl:UpdateKey` in `sl:AssocArrayParameters`) erlaubt die Änderung des Schlüssels eines im Assoziativen Array gespeicherten Schlüssel/Wert-Paares. In `sl:UpdateKey` sind dabei als Attribute der derzeitige (`Key`) sowie der neue Schlüssel (`NewKey`) anzugeben.

Änderung eines Wertes

Mittels der zweiten Anfrage (Element `sl:UpdateValue` in `sl:AssocArrayParameters`) kann der Wert eines im Assoziativen Array gespeicherten Schlüssel/Wert-Paares geändert werden. In `sl:UpdateValue` sind dabei der Schlüssel der zu ändernden Assoziation (Attribut `Key`) sowie der neue Wert (Inhalt von `sl:UpdateValue`) zu spezifizieren. Existiert zum angegebenen Schlüssel kein Eintrag im Assoziativen Array, wird ein neuer Eintrag mit dem angegebenen Schlüssel und dem angegebenen Wert hinzugefügt.

Hinsichtlich der Kodierung des Wertes gelten die in Abschnitt 7.1.1.2, „Update-Parameter und Antwortdaten“ gemachten Aussagen.

Löschen eines Schlüssel/Wert-Paares

Die dritte mögliche Anfrage (Element `sl:DeletePair` in `sl:AssocArrayParameters`) gestattet schließlich das Löschen eines Schlüssel/Wert-Paares aus dem Assoziativen Array. Dazu muß der Schlüssel des Paares als Attribut `Key` in `sl:DeletePair` angegeben werden.

In der entsprechenden Antwort auf alle drei möglichen Update-Anfragen (siehe Abschnitt 7.2.2, „Antwort“) werden keinerlei Daten zurückgeliefert.

7.2. Abfrage der verfügbaren Infoboxen

Mit diesem Kommando erhält die *Applikation* von der *Bürgerkarten-Umgebung* die Auskunft, welche Infoboxen für Zugriffe zur Verfügung stehen.

7.2.1. Anfrage

Die Anfrage ist ein leeres Kommando ohne Parameter.

7.2.2. Antwort

Die Antwort enthält eine Liste von Bezeichnern (Elemente `sl:InfoboxIdentifier`). Jeder Bezeichner entspricht einer Infoxbox, die der *Applikation* für Zugriffe zur Verfügung steht. Die Bezeichner werden in den nachfolgend beschriebenen Kommandos zum Lesen bzw. zum Verändern von Daten einer Infobox zur Identifikation der Infobox verwendet.

7.3. Anlegen einer Infobox

Dieser Befehl dient zum Anlegen einer Infobox durch die *Applikation*.

7.3.1. Anfrage

Das Anfrage-Element `sl:InfoboxCreateRequest` enthält zunächst die beiden Elemente `sl:InfoboxIdentifier` und `sl:InfoboxType`. `sl:InfoboxIdentifier` gibt den Bezeichner an, unter dem die anzulegende Infobox zukünftig ansprechbar sein soll, während `sl:InfoboxType` den Typ der anzulegenden Infobox spezifiziert (entweder `BinaryFile` für eine binäre Infobox oder `AssocArray` für ein assoziatives Feld).

Es folgen zwei weitere obligatorische Elemente: `sl:Creator` enthält als Freitext Informationen zum Betreiber der *Applikation*, welche diese Infobox anlegen möchte. `sl:Purpose` gibt ebenfalls als Freitext Informationen zum Zweck der Infobox an, den diese für die *Applikation* erfüllt. Beide Elemente werden von der *Bürgerkarten-Umgebung* an der *Benutzer-Schnittstelle* zur Information des Bürgers verwendet.

Mit den nächsten beiden optionalen Elementen `sl:ReadAccessAuthorization` bzw. `sl:UpdateAccessAuthorization` kann die *Applikation* Angaben darüber machen, wem das Lesen bzw. Verändern der Infobox erlaubt ist. Je nachdem, ob das Attribut `UserMayChange` auf den Wert `true` oder `false` gesetzt ist, MUSS die *Bürgerkarten-Umgebung* diese Angaben als Empfehlung (Bürger darf die vorgeschlagenen Rechte verändern) oder als Vorschrift (Bürger darf die vorgeschlagenen Rechte nicht verändern) interpretieren. Fehlt eines der beiden Elemente, bleibt es dem Bürger überlassen, die entsprechenden Zugriffsrechte zu vergeben. Genauere Informationen über die Vorgaben an die *Benutzer-Schnittstelle* enthält Anforderungen an die Benutzer-Schnittstelle. Jedes der beiden Elemente enthält ein ein oder mehrere Elemente `sl:RequesterID`. Der Inhalt von `sl:RequesterID` ist ein Pattern entweder für Domänennamen oder IP-Adressen zur Identifikation der zugriffsberechtigten *Applikation*. Für Domänennamen ist die einmalige Angabe einer Wildcard `*` für ein oder mehrere Domänenteile am Beginn des Patterns (z.B. `*` oder `*.gv.at` oder `*.cio.gv.at`, nicht aber `*io.gv.at`) erlaubt, für IP-Adressen die einmalige Angabe einer Wildcard `*` für ein oder mehrere Bytes der IP-Adresse am Ende des Patterns (z.B. `193.170.*` oder `193.170.251.*`, nicht aber `193.170.25*`). Das Attribut `AuthenticationClass` gibt an, wie streng die *Bürgerkarten-Umgebung* diese Identifikation prüfen MUSS. Mögliche Werte für dieses Attribut sind `anonym`, `pseudoanonym`, `certified` sowie `certifiedGovAgency`. Für genaue Informationen über die Bedeutung dieser Authenitiserungsklassen siehe *Abschnitt 2.1, „Authentisierungsklassen“ in Die österreichische Bürgerkarte - Zugriffsschutz* . Die Prüfung eines bestimmten Domänennamens oder einer bestimmten IP-Adresse gegen die spezifizierten Patterns MUSS von der *Bürgerkarten-Umgebung* entsprechend der Reihenfolge der spezifizierten Elemente `sl:RequesterID` erfolgen. Sind beispielsweise die Patterns `*.gv.at` und `*.cio.gv.at` in dieser Reihenfolge spezifiziert, matched der Domänenname `www.cio.gv.at` das Pattern `*.gv.at`, da dieses in der Reihenfolge vor `*.cio.gv.at` kommt.

Schließlich kann die *Applikation* mit den letzten beiden optionalen Elementen `sl:ReadUserConfirmation` bzw. `sl:UpdateUserConfirmation` Angaben darüber machen, auf welche Weise der Bürger einen Lese- oder Schreibzugriff auf die Infobox bestätigen muss. Je nachdem, ob das Attribut `UserMayChange` auf den Wert `true` oder `false` gesetzt ist, MUSS die *Bürgerkarten-Umgebung* diese Angaben als Empfehlung (Bürger darf die vorgeschlagenen Angaben verändern) oder als Vorschrift (Bürger darf die vorgeschlagenen Angaben nicht verändern) interpretieren. Fehlt eines der beiden Elemente, bleibt es dem Bürger überlassen, die entsprechenden Zugriffsrechte zu vergeben. Genauere Informationen über die Vorgaben an die *Benutzer-Schnittstelle* enthält Anforderungen an die Benutzer-Schnittstelle. Jedes der beiden Elemente kann als Textinhalt einen der vier Werte `none`, `info`, `confirm` oder `confirmWithSecret` haben. `none` bedeutet, dass der Zugriff auf die Infobox vom Bürger gar nicht bestätigt werden muss; `info` bedeutet, dass der Bürger über den erfolgen Zugriff über die *Benutzer-Schnittstelle* informiert werden muss (z.B. als Log-Eintrag); `confirm` bedeutet, dass der Bürger über die *Benutzer-Schnittstelle* die Erlaubnis für den Zugriff bestätigen muss; bei `confirmWithSecret` muss der Bürger die Erlaubnis für den Zugriff durch die Eingabe eines Passworts über die *Benutzer-Schnittstelle* erteilen.

7.3.2. Antwort

Das Antwort-Element `sl:InfoboxCreateResponse` ist leer und bestätigt das erfolgreiche Anlegen der Infobox.

7.4. Löschen einer Infobox

Dieser Befehl dient zum Löschen einer Infobox durch die *Applikation*.

7.4.1. Anfrage

Das Anfrage-Element `sl:InfoboxDeleteRequest` enthält nur das Element `sl:InfoboxIdentifier`; der Inhalt dieses Elements bezeichnet die zu löschende Infobox.

7.4.2. Antwort

Das Antwort-Element `sl:InfoboxDeleteResponse` ist leer und bestätigt das erfolgreiche Löschen der Infobox.

7.5. Lesen von Daten einer Infobox

Dieses Kommando erlaubt der *Applikation* das Lesen von Daten einer Infobox.

7.5.1. Anfrage

Die Anfrage `sl:InfoboxReadRequest` enthält zunächst den Bezeichner jener Infobox, deren Inhalt gelesen werden soll (Element `sl:InfoboxIdentifier`). Die *Applikation* kann die Bezeichner aller verfügbaren Infoboxen mit dem Befehl Abschnitt 7.2, „Abfrage der verfügbaren Infoboxen“ abfragen.

Weiters müssen - abhängig vom Typ der Infobox - boxtypspezifische Parameter für die Lese-Anfrage spezifiziert werden; für eine binäre Infobox mit Hilfe des Element `sl:BinaryFileParameters`, für ein assoziatives Array mit Hilfe des Elements `sl:AssocArrayParameters`. Für weitere Informationen zu den boxtypspezifischen Parametern siehe Abschnitt 7.1, „Typen von Infoboxen“.

Schließlich können optional noch boxspezifische Parameter angegeben werden; das sind Parameter, die nur im Kontext einer ganz bestimmten Infobox sinnvoll sind. Verwendung findet dazu das Element `sl:BoxSpecificParameters`; als Inhalt dieses Elements ist grundsätzlich eine beliebige Mischung aus Text und Elementen erlaubt.

> **Anmerkung**
>
> Die Festlegung der boxspezifischen Parameter für die im Konzept Bürgerkarte standardisierten Infoboxen befindet sich im Dokument Standardisierte Key- und Infoboxen

7.5.2. Antwort

Die Antwort besteht aus den abgefragten Daten der bezeichneten Infobox. Für weitere Informationen zu diesen vom Typ der Infobox sowie von der Art der Lese-Anfrage abhängigen Daten siehe Abschitt Abschnitt 7.1, „Typen von Infoboxen“.

7.6. Verändern von Daten einer Infobox

Dieses Kommando erlaubt der *Applikation* das Verändern von Daten einer Infobox.

7.6.1. Anfrage

Die Anfrage `sl:InfoboxUpdateRequest` enthält zunächst den Bezeichner jener Infobox, deren Inhalt verändert werden soll (Element `sl:InfoboxIdentifier`). Die *Applikation* kann die Bezeichner aller verfügbaren Infoboxen mit dem Befehl Abschnitt 7.2, „Abfrage der verfügbaren Infoboxen“ abfragen.

Weiters müssen - abhängig vom Typ der Infobox - boxtypspezifische Parameter für die Update-Anfrage spezifiziert werden; für eine binäre Infobox mit Hilfe des Element `sl:BinaryFileParameters`, für ein assoziatives Array mit Hilfe des Elements `sl:AssocArrayParameters`. Für weitere Informationen zu den boxtypspezifischen Parametern siehe Abschnitt 7.1, „Typen von Infoboxen“.

Schließlich können optional noch boxspezifische Parameter angegeben werden; das sind Parameter, die nur im Kontext einer ganz bestimmten Infobox sinnvoll sind. Verwendung findet dazu das Element

`sl:BoxSpecificParameters`; als Inhalt dieses Elements ist grundsätzlich eine beliebige Mischung aus Text und Elementen erlaubt.

> **Anmerkung**
>
> Die Festlegung der boxspezifischen Parameter für die im Konzept Bürgerkarte standardisierten Infoboxen befindet sich im Dokument Standardisierte Key- und Infoboxen

7.6.2. Antwort

Die Antwort ist ein leeres Kommando ohne Parameter.

8. Abfrage von Eigenschaften

Die Schnittstelle *Security-Layer* bietet die Möglichkeit, eine Reihe von Eigenschaften der *Bürgerkarten-Umgebung* sowie den Status des verwendeten Bürgerkarten-Tokens abzufragen.

> **Anmerkung**
>
> Für Beispiele zu diesem Befehl siehe Tutorium; für Anforderungen an die Benutzerschnittstelle siehe Abschnitt 3.8, „Abfrage von Eigenschaften“ in Die österreichische Bürgerkarte - Anforderungen an die Benutzer-Schnittstelle.

8.1. Abfrage der Umgebungseigenschaften

Die *Applikation* benötigt für bestimmte Aufgaben gegebenenfalls zusätzliche Informationen über die *Bürgerkarten-Umgebung*. Dieses Kommando dient zur Abfrage solcher Eigenschafen.

8.1.1. Anfrage

Die Anfrage besteht aus dem leeren Element `sl:GetPropertiesRequest`.

8.1.2. Antwort

Die Antwort besteht aus dem Element `sl:GetPropertiesResponse` und enthält als Kindelemente folgende Eigenschaften der *Bürgerkarten-Umgebung*:

- `sl:ViewerMediaType`: Unterstützte Formate der Anzeige: Bei Kenntnis der Fähigkeiten der Anzeige der *Bürgerkarten-Umgebung* kann die *Applikation* bei Anfragen zur Erstellung einer Signatur auf diese Fähigkeiten Rücksicht nehmen.

  > **Anmerkung**
  >
  > Für jedenfalls von einer Bürgerkarte zu unterstützende Anzeigeformate siehe das Spezifikationsdokument Minimale Umsetzung des Security-Layers.

- `sl:XMLSignatureTransform`: Unterstützte Transformationen für Erstellung/Überprüfung einer XML-Signatur: Bei Kenntnis der Fähigkeiten der Transformationsunterstützung der *Bürgerkarten-Umgebung* kann die *Applikation* bei Anfragen zur Erstellung einer Signatur auf diese Fähigkeiten Rücksicht nehmen.

  > **Anmerkung**
  >
  > Für jedenfalls von einer Bürgerkarte zu unterstützende Transformationen siehe das Spezifikationsdokument Minimale Umsetzung des Security-Layers.

- `sl:KeyboxIdentifier`: Bezeichner aller in der *Bürgerkarten-Umgebung* zum Zeitpunkt der Anfrage vorhandenen Schlüsselpaare: Diese Bezeichner werden für all jene Kommandos benötigt, deren Ausführung der Verwendung eines bestimmten Schlüsselpaares durch die *Bürgerkarten-Umgebung* bedarf. Für jedes Schlüsselpaar wird angegeben, ob es zur Verwendung im Kontext Signatur (Attribut `Signature`) bzw. Verschlüsselung (Attribut `Encryption`) geeignet ist.

  > **Anmerkung**
  >
  > Für jedenfalls von einer Bürgerkarte zu unterstützende Schlüsselpaarbezeichner siehe das

Spezifikationsdokument Standardisierte Key- und Infoboxen.

- `sl:Binding`: Unterstützte Bindungen an Transportprotokolle: Damit kann die *Applikation* in Erfahrung bringen, über welche Transportprotokolle die *Bürgerkarten-Umgebung* angesprochen werden kann. Für jede implementierte Bindung wird ein Bezeichner sowie optional eine Menge von bindungsspezifischen Parametern zurückgeliefert.

  Anmerkung

  Für Bezeichner und mögliche Parameter siehe Dokument Transportprotokolle Security-Layer. Für jedenfalls von einer Bürgerkarte zu unterstützende Transportprotokolle siehe das Spezifikationsdokument Minimale Umsetzung des Security-Layers.

- `sl:ProtocolVersion`: Unterstützte Protokollversionen des *Security-Layer*: Damit kann die *Applikation* ersehen, welche Protokollversionen des *Security-Layer* die *Bürgerkarten-Umgebung* unterstützt. Die Protokollversion, die durch dieses Spezifikationsdokument beschrieben wird, ist `1.2`.

8.2. Abfrage des Tokenstatus

Der *Security-Layer* sieht die Möglichkeit vor, den Status des verwendeten Bürgerkarten-Tokens abzufragen. Mögliche Stati sind dabei `ready` (Token vorhanden und initialisiert) oder `removed` (kein Token vorhanden).

8.2.1. Anfrage

Enthält der Befehl, bestehend aus dem Element `sl:GetStatusRequest`, keine Parameter, liefert die *Bürgerkarten-Umgebung* in der Antwort sofort den aktuellen Status des verwendeten Bürgerkarten-Tokens zurück.

Optional kann die Anfrage jedoch die beiden Elemente `sl:TokenStatus` und `sl:MaxDelay` enthalten. In einem solchen Fall gibt `sl:TokenStatus` den von der *Applikation* gewünschten Status des Bürgerkarten-Tokens an, und `sl:MaxDelay` die längste Zeitspanne in Sekunden, um die die *Bürgerkarten-Umgebung* die Antwort auf diese Anfrage bis zum Eintritt des gewünschten Zustandes verzögern soll.

8.2.2. Antwort

Die Antwort besteht aus dem Element `sl:GetStatusResponse` und enthält als Text des Kindelements `sl:TokenStatus` den aktuellen Status des Tokens (entweder `ready` oder `removed`).

9. Null-Operation

Die Null-Operation ist ein Kommando, das die *Bürgerkarten-Umgebung* beantwortet, ohne dass dazu irgendwelche Berechnungen notwendig wären. Auf eine parameterlose Anfrage wird eine parameterlose Antwort gesendet.

Anmerkung

Eine sinnvolle Anwendung dieses Kommandos existiert in Zusammenhang mit der Authenifizierung gegenüber der *Bürgerkarten-Umgebung* bei Verwendung der HTTP- bzw. HTTPS-Bindung des *Security-Layer*s: Die Authentifizierung funktioniert sinnvoll nur bei Verwendung der Data-URL; um jedoch mit der Data-URL operieren zu können, muss zumindest ein erster Befehl vom Browser des Bürgers an die *Bürgerkarten-Umgebung* gesendet werden, für den die Authentifizierung nicht sinnvoll möglich ist. Für diesen ersten Befehl kann die Null-Operation eingesetzt werden (für genauere Informationen zur Authentifizierung der *Applikation* gegenüber der *Bürgerkarten-Umgebung* vergleicheZugriffschutz).

9.1. Anfrage

Die *Applikation* sendet die parameterlose Anfrage bestehend aus dem Element `sl:NullOperationRequest` an die *Bürgerkarten-Umgebung*.

9.2. Antwort

Die *Bürgerkarten-Umgebung* empfängt die parameterlose Anfrage und reagiert darauf mit der ebenfalls parameterlosen Antwort `sl:NullOperationResponse`.

Anmerkung

Für Beispiele zu diesem Befehl siehe Tutorium; für Anforderungen an die Benutzerschnittstelle siehe Abschnitt 3.9, „Null-Operation“ in Die österreichische Bürgerkarte - Anforderungen an die Benutzer-Schnittstelle.

10. Fehlerbehandlung

Sollte innerhalb der *Bürgerkarten-Umgebung* ein Fehler auftreten, der die korrekte Beantwortung einer Anfrage verhindert, wird anstatt der zur Anfrage gehörenden Antwort eine Fehler-Antwort an die *Applikation* zurückgeschickt.

Die Datenstruktur der Fehler-Antwort besteht aus einem maschinenlesbaren Fehlercode (`sl:ErrorCode`), sowie optional aus weiterführender Information (`sl:Info`), die an dieser Stelle nicht näher spezifiziert wird. Vorstellbar wäre etwa eine klartextliche Erläuterung des Fehlers, oder eine XML-Struktur mit nähren Informationen zum aufgetretenen Fehler.

Anmerkung

Für Beispiele zur Fehlerbehandlung siehe Tutorium.

10.1. Fehlercodes

Siehe eigenes Dokument Fehlercodes Security-Layer.

Glossar

Glossar

Applikation

Jenes Programm, das Anfragen an die *Bürgerkarten-Umgebung* über den *Security-Layer* richtet und die entsprechenden Antworten entgegennimmt und auswertet.

Benutzer-Schnittstelle

Jene Schnittstelle, über die der *Bürger* mit der *Bürgerkarten-Umgebung* kommuniziert. Über diese Schnittstelle wird einerseits die Benutzerinteraktion abgewickelt, die gegebenenfalls zur Abwicklung eines Befehls des *Security-Layers* notwendig ist (z.B. die Anzeige eines zu signierenden Dokuments beim Befehl zur Erzeugung einer XML-Signatur); andererseits kann der *Bürger* über diese Schnittstelle seine *Bürgerkarten-Umgebung* nach seinen persönlichen Bedürfnissen konfigurieren (z.B. kann er Einstellungen zum Zugriffsschutz auf seine Infoboxen verändern). Die Vorgaben an die *Benutzer-Schnittstelle* sind in Minimale Umsetzung des Security-Layers geregelt.

Bürger

Jene Person, die die Funktionen der *Bürgerkarten-Umgebung* für die sichere Abwicklung von E-Government oder E-Commerce verwenden möchte. Die Ansteuerung der *Bürgerkarten-Umgebung* erfolgt in der Regel nicht durch den *Bürger* selbst, sondern durch die *Applikation*, welche die E-Government oder E-Commerce Anwendung repräsentiert.

Bürgerkarte

Laut [E-GovG], §10 ZI 10 ist die *Bürgerkarte* „die unabhängig von der Umsetzung auf unterschiedlichen technischen Komponenten gebildete logische Einheit, die eine elektronische Signatur mit einer Personenbindung (§ 4 Abs. 2) und den zugehörigen Sicherheitsdaten und -funktionen sowie mit allenfalls vorhandenen Vollmachtsdaten verbindet“. Im Sinne der in den Spezifikationen zur österreichischen Bürgerkarte gebrauchten Terminologie ist die *Bürgerkarten-Umgebung* die Implementierung der logischen Einheit *Bürgerkarte*.

Bürgerkarten-Umgebung

Jenes Programm bzw. jener Dienst, der die Funktionalität der *Bürgerkarte* zur Verfügung stellt. Grundsätzlich vorstellbar ist die Ausführung als Programm, das lokal am Rechner des *Bürgers* läuft (*lokale Bürgerkarten-Umgebung*), oder als serverbasierter Dienst, der über das Internet angesprochen wird (*serverbasierte Bürgerkarten-Umgebung*). Die Interaktion mit diesem Programm bzw. Dienst wird über zwei Schnittstellen abgewickelt: Über die *Benutzer-Schnittstelle* sowie über den *Security-Layer*.

Hash-Eingangsdaten

Jene Daten, die für die Berechnung des Hash-Wertes für eine `dsig:Reference` verwendet werden. Sind für die `dsig:Reference` Transformationen angegeben, entsprechen diese Daten dem Ergebnis der letzten Transformation. Sind keine Transformationen spezifiziert, gleichen die Hash-Eingangsdaten den *Referenz-Eingangsdaten*.

Impliziter Transformationsparameter

Siehe Abschnitt 2.2.2.2, „Implizite Transformationsparameter“

Referenz-Eingangsdaten

Jene Daten, die sich aus der Auflösung der im Attribut `URI` der `dsig:Reference` angegebenen URI ergeben. Sind für die `dsig:Reference` Transformationen angegeben, werden diese Daten als Eingangsdaten zur Berechnung der ersten Transformation verwendet. Sind keine Transformationen spezifiziert, gleichen die Referenz-Eingangsdaten den *Hash-Eingangsdaten*.

Security-Layer

Jene Schnittstelle, über die die *Applikation* mit der *Bürgerkarten-Umgebung* kommuniziert. Das genaue Protokoll, das über diese Schnittstelle gesprochen werden kann, wird in Applikationsschnittstelle Security-Layer spezifiziert. Die möglichen Bindungen dieses Protokolls an Transportschichten wie HTTP oder TCP wird in Transportprotokolle Security-Layer geregelt.

Signaturmanifest

Siehe Abschnitt 2.2.2.2, „Implizite Transformationsparameter“ .

Referenzen

[CMS] BHously, R.: *RFC 3369: Cryptographic Message Syntax (CMS)* , IETF Request For Comment, August 2002

[CMS-AES] chaad, J.: *RFC 3565: Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)* . IETF Request For Comment, Juli 2003.

[CMS-Alg] Hously, R.: *RFC 3370: Cryptographic Message Syntax (CMS) Algorithms* . IETF Request For Comment, August 2002.

[CMS-RSAES-OAEP] Hously, R.: *RFC 3560: Use of the RSAES-OAEP Key Transport Algorithm in the Cryptographic Message Syntax (CMS)* . IETF Request For Comment, Juli 2003.

[CSS 2] Bert Bos, Håkon Wium Lie, Chris Lilley und Ian Jacobs: *Cascading Style Sheets, level 2* . W3C Recommendation, Mai 1998.

[EC14N] Boyer, John, Eastlake, Donald und Reagle, Joseph: *Exclusive XML Canonicalization. W3C Recommendation, Juli 2002* .

[ECDSA-CMS] Blake-Wilson, S., Brown, D., Lampert, D.: *RFC 3278: Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)* . IETF Request For Comment, April 2002.

[ECDSA-XML] Blake-Wilson, S., Karlinger, G. und Wang, Y.: *ECDSA with XML-Signature Syntax* . Internet-Draft, Jänner 2004.

[E-GovG] *BGBl. I Nr. 10/2004.*

[ESS-S/MIME] Hoffman, P.: *RFC 2634: Enhanced Security Services for S/MIME* , IETF Request For Comment, Juni 1999

[ETSICMS] European Telecommunications Standards Institute: *ETSI TS 101733: Electronic Signature Formats, v1.5.1* , Technical Specification, Dezember 2003

[ETSIQCert] European Telecommunications Standards Institute: *ETSI TS 101 862: Qualified certificate profile, v1.2.1* , Technical Specification, Juni 2001

[ETSIXML] European Telecommunications Standards Institute: *ETSI TS 101903: XML Advanced Electronic Signatures (XAdES), v1.2.2* , Technical Specification, April 2004

[GIF] *Graphics Interchange Format, Version 89a* . CompuServe Incorporated, Juli 1990.

[HTML4] Dave Ragget, Arnaud Le Hors und Ian Jacobs: *HTML 4.01 Specification* . W3C Recommendation, Dezember 1999.

[HTTP1.1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leech und T. Berners-Lee: *Hypertext Transfer Protocol -- HTTP/1.1*. IETF Request For Comment, Juni 1999.

[HTTPS] E. Rescorla *HTTP over TLS*. IETF Request For Comment, Mai 2000

[ISO-8859-1] *ISO/IEC 8859-1*:1998: Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1.

[ISO-8859-10] *ISO/IEC 8859-10*:1998: Information technology -- 8-bit single-byte coded graphic character sets -- Part 10: Latin alphabet No. 6.

[ISO-8859-15] *ISO/IEC 8859-15*:1999: Information technology -- 8-bit single-byte coded graphic character sets -- Part 15: Latin alphabet No. 9.

[ISO-8859-2] *ISO/IEC 8859-2*:1999: Information technology -- 8-bit single-byte coded graphic character sets -- Part 2: Latin alphabet No. 2.

[ISO-8859-3] *ISO/IEC 8859-3*:1999: Information technology -- 8-bit single-byte coded graphic character sets -- Part 3: Latin alphabet No. 3.

[ISO-8859-9] *ISO/IEC 8859-9*:1999: Information technology -- 8-bit single-byte coded graphic character sets -- Part 9: Latin alphabet No. 5.

[JPEG] Eric Hamilton: *JPEG File Interchange Format, Version 1.02* . C-Cube Microsystems, September 1992.

[KEYWORDS] Bradner, S.: *RFC 2119: Key words for use in RFCs to Indicate Requirement Levels* , IETF Request For Comment, März 1997

[MIME] Freed, N. und Borenstein, N.: *RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types* , IETF Request For Comment, November 1996

[PersBin] Hollosi, Arno und Karlinger, Gregor: *XML-Definition der Personenbindung* . Konvention zum E-Government Austria erarbeitet von der Stabsstelle IKT-Strategie des Bundes, Technik und Standards. Öffentlicher Entwurf, Version 1.2.2, 14. Februar 2005.

[PersonData] Naber, Larissa: *PersonData Struktur - XML Spezifikation* . Konvention zum E-Government Austria erarbeitet von der Arbeitsgruppe Kommunikationsarchitekturen. Öffentlicher Entwurf, Version 2.0.0, 14. Oktober 2004.

[PKCS#12] RSA Laboratories: *PKCS#12 v1.0: Personal Information Exchange Syntax* , Juni 1999.

[port-numbers] Internet Assigned Numbers Authority: *Port Numbers*

[QCert] Santesson, S. und Nystrom M.: *RFC 3739: Internet X.509 Public Key Infrastructure: Qualified Certificates Profile* , IETF Request For Comment, März 2004

[SigG] *BGBl I Nr. 190/1999* idF BGBl I Nr. 152/2001.

[SigV] *BGBl II Nr. 30/2000* idF BGBl II Nr. 527/2004.

[Stammzahl] Hollosi, Arno und Hörbe, Rainer: *Bildung von Stammzahl und bereichsspezifischem Personenkennzeichen (bPK)* . Konvention zum E-Government Austria erarbeitet von der Stabsstelle IKT-Strategie des Bundes, Technik und Standards sowie vom Bundesminsterium für Inneres. Öffentlicher Entwurf, Version 1.0, 2. Februar 2004.

[TLS] T. Dierks und C. Allen: *The TLS Protocol Version 1.0* . IETF Request For Comment, Januar 1999.

[Unicode] The Unicode Consortium. *The Unicode Standard, Version 4.0.0* , defined by: The Unicode Standard, Version 4.0 (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1).

[URI] Berners-Lee, T. , Fielding, R. und Masinter, L.: *RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax* , IETF Request For Comment, August 1998

[VerwEig] Hollosi, Arno: *X.509 Zertifikatserweiterungen für die Verwaltung* . Konvention zum E-Government Austria erarbeitet von der Stabsstelle IKT-Strategie des Bundes, Technik und Standards. Öffentlicher Entwurf, Version 1.0.3, 21. Februar 2005.

[X509] Polk, W., Ford, W., Solo, D.: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* . IETF Request For Comment, April 2002.

[XHTML 1.1] Murray Altheim, Frank Boumphrey, Sam Dooley, Shane McCarron, Sebastian Schnitzenbaumer und Ted Wugofski: *Modularization of XHTML* . W3C Recommendation, April 2001.

[XHTML MOD] Daniel Austin, Subramanian Peruvemba, Shane McCarron, Masayasu Ishikawa: *Modularization of XHTML in XML Schema* . W3C Working Draft, Oktober 2003.

[XML] Bray, Tim, Paoli, Jean, Sperberg-McQueen, C.M. und Maler, Eve: *Extensible Markup Language (XML) 1.0 (Second Edition)* , W3C Recommendation, Oktober 2000.

[XMLDecTF] Hughes, Merlin, Imamura, Takeshi und Maruyama, Hiroshi: *Decryption Transform for XML Signature* . W3C Recommendation, Dezember 2002.

[XMLDSIG] Eastlake, Donald, Reagle, Joseph und Solo, David: *XML-Signature Syntax and Processing* , W3C Recommendation, Februar 2002

[XMLDSIG-URI] Eastlake, Donald: *RFC 4051: Additional XML Security Uniform Resource Identifiers (URIs)* , IETF Request For Comments, April 2005

[XMLEnc] Eastlake, Donald und Reagle, Joseph: *XML Encryption Syntax and Processing* , W3C Recommendation, Dezember 2002

[XML-Schema] Thompson, Henry S., Beech, David, Maloney, Murray und Mendelson, Noah: *XML Schema Part 1: Structures* , W3C Recommendation, Mai 2001

[XMLTYPE] Murata, M., St.Laurent, S., und Kohn, D.: *RFC 3023: XML Media Types* , IETF Request For Comment, Jänner 2001.

[XPath] Clark, James und DeRose, Steven: *XML Path Language* , W3C Recommendation, November 1999

[XPF2] Boyer, John, Hughes, Merlin und Reagle, Joseph: *XML-Signature XPath Filter 2.0* . W3C Candidate Recommendation, Juli 2002.

[XPointer] Grosso, Paul, Maler, Eve, Marsh, Jonathan und Walsh, Norman: *XPointer Framework* . W3C Recommendation, März 2003.

[XSS-FAQ] Cgisecurity.com: *The Cross Site Scripting FAQ* .

A. Historie

Datum	Version	Änderungen
20.02.2008	1.2.3	• Abschnitt 4.2, „Verschlüsselung als XML-Dokument“ und Abschnitt 5.2, „Entschlüsselung eines XML-Dokuments“ wurde um die Möglichkeit zur Schlüsselvereinbarung (key agreement) erweitert.
01.03.2005	1.2.2	• Errata Erratum 26 in Die österreichische Bürgerkarte - Errata, Erratum 27 in Die österreichische Bürgerkarte - Errata, Erratum 29 in Die österreichische Bürgerkarte - Errata und Erratum 32 in Die österreichische Bürgerkarte - Errata ausgebessert.
29.06.2004	1.2.1	• Erratum 22 laut Errata ausgebessert.

14.05.2004		• Errata 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 20 laut Errata ausgebessert.
		• Begriff *TrustedViewer* durch *Anzeige* ersetzt.
		• Einheitlichen neuen Namenraum für alle Protokoll-Elemente geschaffen.
		• `sl:CreateCMSSignatureRequest`:
		  ○ Hinweis zum Mime-Type der zu signierenden Datenobjekte eingefügt.
		  ○ ETSI-Property `SigningTime` als MUSS hinzugefügt.
		• `sl:CreateXMLSignatureRequest`:
		  ○ Hinweis eingefügt, wie interne URIs auf zu signierende Daten zu interpretieren sind.
		  ○ Datenobjekte explizit nun auch als URL-Zuordnung (neben Base64Content und XMLContent) übergebbar.
		  ○ Supplements explizit nun auch als URL-Zuordnung (neben Base64Content und XMLContent) übergebbar.
		  ○ Hinweis zum Mime-Type der zu signierenden Datenobjekte eingefügt.
		  ○ ETSI-Property `etsi:SigningTime` als MUSS hinzugefügt.
		• `sl:VerifyXMLSignatureRequest`:
		  ○ Supplements explizit nun auch als URL-Zuordnung (neben Base64Content und XMLContent) übergebbar.
		• `sl:InfoboxReadRequest`:
		  ○ Optionale boxspezifische Parameter eingeführt.
		  ○ Assoziatives Array: Optionales Attribut UserMakesUnique bei Auswahl von Schlüsseln bzw. Schlüssel/Wert-Paaren hinzugefügt.
		• `sl:InfoboxUpdateRequest`:
		  ○ Optionale boxspezifische Parameter eingeführt.

		○ Assoziatives Array: Update eines nicht vorhandenen Schlüssel/Wert-Paares bedeutet Hinzufügen des Schlüssel/Wert-Paares. • `sl:GetPropertiesResponse`: ○ `sl:KeyBoxIdentifier` um zwei Attribute `Signature` und `Encryption` erweitert, die über die Verwendbarkeit des Schlüsselpaares für Signatur bzw. Verschlüsselung Auskunft geben. • Befehl *CreateSymmetricSecret* entfernt. • Befehl *NullOperation* eingeführt. • Befehle *CreateHash* und *VerifyHash* eingeführt. • Befehle *InfoboxCreate* und *InfoboxDelete* eingeführt. • Entschlüsselungsbefehle (*DecryptCMS*, *DecryptXML*) eingeführt.
31.08.2002		• `sl:CreateCMSSignatureResponse`: ○ Empfehlung für die Kodierung der Zertifikatskette auf `CertificateSet` eingeschränkt. • `sl:VerifyCMSSignatureRequest`: ○ Attribut `Signatories` zur Spezifizierung der zu prüfenden Signaturen bei mehreren in der CMS-Signatur angegebene Signatoren hinzugefügt. • `sl:VerifyCMSSignatureResponse`: ○ Erweiterung der Kardinalität der Sequenz `(SignerInfo, SignatureCheck, CertificateCheck)` auf `1..oo`, um CMS-Signaturen mit mehreren Signatoren zu unterstützen. ○ Hinzufügen der Empfehlung, dass die Bürgerkarten-Umgebung zur Überprüfung älterer Signaturen die ggf. vorhandenen Signaturattribute `CompleteCertificateRefs`, `CompleteRevocationRefs`, `CertificateValues`, `RevocationsValues` nach [ETSICMS] auswerten können soll. ○ Beim Ergebnis der Prüfung der Signaturprüfdaten wird nun zwischen der Sperre und dem Widerruf eines Zertifkats der Zertifikatskette unterschieden (Fehlercodes 4 und 5). ○ Bei den Informationen zum Signator wird nun auch zurückgeliefert, ob das Signatorzertifikat qualifiziert ist. ○ Empfehlung für die Kodierung der Zertifikatskette auf die Mechanismen von XMLDSIG geändert (`X509Data`, `RetrievalMethod`). • `sl:CreateXMLSignatureRequest`: ○ Element `sl:SignatureInfo` hinzugefügt, um das Erstellen von *Enveloped Signatures* zu ermöglichen. ○ Beispiel entsprechend korrigiert. • `sl:CreateXMLSignatureResponse`: ○ Beispiel entsprechend des geänderten Anfragebeispiels abgeändert. • `sl:VerifyXMLSignatureRequest`: ○ Grammatik für Element `sl:SignatureEnvironment` verallgemeinert, um auch XML-Dokumente mit DTDs angeben zu können. ○ Beschreibung der Ergänzungsobjekte verallgemeinert. ○ Beispiel entsprechend den Änderungen in `sl:SignatureEnvironment` korrigiert. • `sl:VerifyXMLSignatureResponse`: ○ Erweiterung des Resultats für die Signaturprüfung um die Angabe, welche Referenzen der Signatur gegebenenfalls nicht validiert werden konnten. ○ Erweiterung des Resultats für die Prüfung des Signaturmanifests, welche Referenzen des Manifests gegebenenfalls nicht validiert werden konnten. ○ Einführung der Prüfung von gewöhnlichen Manifesten, die in der zu

		prüfenden Signatur referenziert werden. ○ Hinzufügen der Empfehlung, dass die Bürgerkarten-Umgebung zur Überprüfung älterer Signaturen die ggf. vorhandenen Signaturattribute `etsi:CompleteCertificateRefs`, `etsi:CompleteRevocationRefs`, `etsi:CertificateValues`, `etsi:RevocationsValues` nach [ETSIXML] auswerten können soll. ○ Beim Ergebnis der Prüfung der Signaturprüfdaten wird nun zwischen der Sperre und dem Widerruf eines Zertifkats der Zertifikatskette unterschieden (Fehlercodes 4 und 5). ○ Bei den Informationen zum Signator wird nun auch zurückgeliefert, ob das Signatorzertifikat qualifiziert ist. • `sl:CreateSessionKeyRequest`, `sl:CreateSessionkeyResponse`: ○ Befehle entfernt. • `sl:GetPropertiesResponse`: ○ Kardinalitäten der einzelnen Umgebungsparameter im Schema auf `1..oo` gelockert, um bei zukünftigen Änderungen nicht den Namespace des Befehls ändern zu müssen. ○ Any-Platzhalter für zukünftige Umgebungsparameter aufgenommen, um bei zukünftigen Änderungen nicht den Namespace des Befehls ändern zu müssen. ○ Neuen Umgebungsparamter `sl:ProtocolVersion` aufgenommen.
		• Schlüsselwörter entsprechend RFC 2119 verwendet. • Elemente in allen Beschreibungen und Beispielen mit NS-Präfix `sl` versehen. • Referenzen hinzugefügt: ○ XML ○ XML Namespaces ○ XML Schema ○ RFC Keyword ○ Enhanced Security Services for S/MIME • Begriffliche Konsistenz: Bürgerkarten-Umgebung statt Security-Kapsel, Bürgerkarten-Token statt Bürgerkarte • `sl:CreateCMSSignatureRequest`: ○ Referenz für Signaturattribut `ContentHints` von [ETSICMS] auf [ESS-S/MIME] geändert, da dieses Attribut in einer Revision von [ETSICMS] entfernt wurde. • `sl:VerifyCMSSignatureResponse`: ○ Vorgehensweise bei Vorkommen mehrerer Signatoren in der zu prüfenden CMS-Signatur präzisiert. ○ Referenz für Signaturattribut `ContentHints` von [ETSICMS] auf [ESS-S/MIME] geändert, da dieses Attribut in einer Revision von [ETSICMS] entfernt wurde. • `sl:CreateXMLSignatureRequest`: ○ Vorgangsweise bei der Integration von zu signierenden Datenobjekten im Fall einer "`enveloped`" Struktur präzisiert. ○ Explizite Angabe von Namenraumdeklarationen innerhalb einer im Befehl übergebenen Transformationskette (`dsig:Transforms`) durch die Applikation vorgeschrieben. ○ Vorgangsweise bei der Auflösung von Daten während des Durchlaufens des Transformationsprozesses durch die Bürgerkarten-Umgebung präzisiert. • `sl:VerifyXMLSignatureResponse`: ○ Bedeutung des Fehlercodes `2` bei der Überprüfung des Signaturmanifests

		präzisiert. • `sl:InfoboxUpdateRequest`: o Tippfehler in Schema und Beispiel korrigiert (nun `sl:InfoboxIdentifier` statt `sl:InfoboxIdentifer`). • `sl:CreateSymmetricSecretRequest`, `sl:CreateSymmetricSecretResponse`: o Copy/Paste-Fehler in der Beschreibung der Anfrage korrigiert. o Beispiel der Anfrage korrigiert. o Beschreibung der Berechnung präzisiert
24.04.2002		• Hinweis auf Spezifikation "Anforderungen Bürgerkarten-Umgebung" in Einleitung aufgenommen. • Hinweis auf Dokument "Minimalanforderungen Security-Layer" in Einleitung aufgenommen. • Referenz "AnfBKU" aufgenommen. • Referenz "ETSICMS" auf Dokument in der Version 1.3.1 aktualisiert.