CCES-Signature-API

Version 1.0 vom 21.01.2005

Zusammenfassung

Dieses Papier spezifiziert die Anforderungen an eine herstellerunabhängige Schnittstelle zwischen Dokumentenmanagement- und Signatursystemen, erörtert die Eignung existierender Standard-APIs und spezifiziert schließlich mit der CCES-Signature-API eine offene Programmierschnittstelle, mittels der Funktionen für langfristig überprüfbare elektronische Signaturen leicht in Anwendungssysteme integriert werden können.

Inhaltsverzeichnis

Ansprechpartner

Competence Center Elektronische Signatur im VOI e.V. (cces.info@voi.de)

Dr. Detlef Hühnlein (Tel. 09571-896479)

Rolf Schmoldt (Tel. 069-58700635)

Änderungshistorie

Version	Datum	Änderung
0.80	30.07.2004	Erste Gesamtfassung zur CCES-internen Abstimmung
0.90	07.09.2004	Einarbeitung erster CCES-interner Kommentare: • Bereitstellung einer Funktion „Hash“ zur Erzeugung von Hash-Werten (angeregt von Herrn Schmoldt); siehe Kapitel 4.1.2, 4.2.1.2.2 und 4.2.4 sowie die Erweiterung des Aufrufes von CreateEvidenceRecord in Kapitel 4.2.10. • Bereitstellung einer separaten Funktion „ValidateCertificate“ zur Validierung von Zertifikaten (angeregt von den Herren Schmoldt und Hartwich); siehe Kapitel 4.1.5 und 4.2.7. • Präzisierung der Aufrufbeschreibung der „Decrypt„-Funktion in Kapitel 4.2.14 (angeregt von Herrn Berndt) • Klarstellende Umbenennung der Verify-Options in Kapitel 4.2.1.2.3 (angeregt durch Herrn Janhoff) • Einführung einer Funktion „CountSignatures“ zur Unterstützung von Mehrfachsignaturen (angeregt durch Herrn Janhoff); siehe Kapitel 4.1.3, 4.2.1.1.6 und 4.2.5. • Erweiterung der „Verify“-Funktion zur selektiven Prüfung einzelner Signaturen und zur Unterstützung von Mehrfachsignaturen (angeregt durch Herrn Janhoff); siehe Kapitel 4.1.4, 4.2.1.3.1 und 4.2.6 sowie die Einführung des zusätzlichen Fehlercodes CCES_NotExistingSignature in Kapitel 4.2.1.1.1. • Ergänzung der möglichen CCES_RevocationStates (nunmehr CCES_CertStatus) in Kapitel 4.2.1.3.6 um die Sperrgründe aus [ISIS-MTT] (Part 1, Table 38) bzw. [RFC3280] (angeregt durch Herrn Belke)

		• Update der referenzierten ISIS-MTT-Version (angeregt durch Herrn Belke)
0.98	19.11.2004	Ausbesserung von Tippfehlern und Einarbeitung weiterer CCES-interner Kommentare: • Löschen der expliziten Definition von CCES_ErrorLanguage. Statt dessen wird in der Funktion GetErrorMessage der Sprachcode gemäß [ISO639-1] übergeben (angeregt von Dr. Kampffmeyer) • Einfügen einer Erweiterungsmöglichkeit bzgl. Signatur- und Datenformaten (angeregt von Dr. Kampffmeyer). Dies wird dadurch realisiert, dass ein abstrakter ASN.1-codierter Datentyp e_AbstractASN1Type eingeführt wird. Außerdem wird bei den CCES_SignOptions (Abschnitt 4.2.1.2.1), den CCES_HashOptions (Abschnitt 4.2.1.2.2) und CCES_EncryptOptions (Abschnitt 4.2.1.2.6) jeweils zwischen Formaten auf einer höheren Schicht und Algorithmen auf einer unteren Schicht unterschieden. Die Optionen setzen sich aus einem explizit definierten High-Level-Format und einem optionalen, mittels Object Identifier referenzierten, kryptographischen Algorithmus zusammen. Fehlt die Angabe der OID, so werden jeweils die Standardalgorithmen verwendet. Dadurch können neue, heute gänzlich unbekannte, Algorithmen über die CCES-Signature-API angesprochen werden. Durch die Kombination des abstrakten Datentyps e_AbstractASN1Type mit der Möglichkeit den Signaturalgorithmus explizit zu spezifizieren, können auch nicht-PKCS#1-basierte Signaturverfahren verwendet werden. • Komplette Überarbeitung der Datenstrukturen für Prüfergebnisse in Abschnitt 4.2.1.3 (teilweise angeregt durch Herrn Pichler). In diesem Zusammenhang wurden auch die CCES_VerifyOptions in Abschnitt 4.2.1.2.3 entsprechend angepasst. • Bei der Funktion ValidateCertificate kann die vorgesehene Schlüsselnutzung übergeben werden, so dass diese in die Prüfung eines Zertifikatspfades einfließen kann. Hierfür wurde in Abschnitt 4.2.1.1.7, in Anlehnung an [RFC3280], die Datenstruktur CCES_KeyUsage zur Spezifikation der Schlüsselnutzung eingeführt. Im Gegenzug wurde die Datenstruktur CCES_CertificateType gelöscht.

		• Einführung der Funktion CCES_GetContentType in Abschnitt 4.2.2.3.5 zur Ermittlung des Datentyps eines übergebenen Datencontainers (angeregt durch Herrn Dickmann). In diesem Zusammenhang wurden auch die Aufrufoptionen in Abschnitt 4.2.1.2.7 eingeführt. • Um die in einem PSE (z.B. Chipkarte) gespeicherten Zertifikate auslesen zu können, wurden die beiden Funktionen ReadCertListFromPSE (Abschnitt 4.2.2.2.3) und GetCertFromPSE (Abschnitt 4.2.2.2.4) eingeführt. In diesem Zusammenhang wurde in Abschnitt 4.2.1.1.1 auch der Fehlercode CCES_CertificateNotFound ergänzt.
0.99	19.12.2004	Einfügen einer Funktion GetLibraryInfo (vgl. Abschnitt 4.2.1.1.8 und 4.2.2.3.6), mit der grundlegende Informationen über die eingesetzte Bibliothek angefordert werden können (angeregt durch Herrn Dickmann). Einfügen einer Klarstellung, dass die Signaturparameter bei einer XML-Signatur im Rahmen der zu signierenden XML-Datenstruktur übergeben werden (vgl. Abschnitt 4.2.3). Löschen der Signaturoption zur Integration von Signaturen in TIFF-Dokumente, da eine entsprechende Standardisierung derzeit noch aussteht (vgl. Abschnitt 4.2.3).
1.00	21.01.2005	Erste Freigegebene Version

1 Einleitung

Durch das Fehlen geeigneter herstellerunabhängiger Programmier- oder gar „Plug-and-Play“-Schnittstellen ist mit der Integration der elektronischen Signatur in Anwendungssysteme, z.B. im Umfeld des Dokumentenmanagements, oft erheblicher individueller Entwicklungsaufwand verbunden. Ist das Anforderungsprofil für zu integrierende Signaturanwendungskomponenten zudem unklar oder starken Änderungen unterworfen, so schwindet die Investitionssicherheit weiter und der Business Case für die Signaturintegration ist für den Applikationsanbieter zunehmend schwieriger darstellbar.

Um dieses Hindernis, das der Nutzung der elektronischen Signatur in vielen Anwendungsszenarien entgegensteht, zu beseitigen, soll eine offene, herstellerunabhängige Schnittstelle definiert werden, durch die die Integration der elektronischen Signatur in Anwendungen erleichtert wird.

Dieses Dokument ist wie folgt gegliedert. In Abschnitt 2 werden die wichtigsten Anforderungen an eine Schnittstelle zur Integration der elektronischen Signatur im Umfeld des Dokumentenmanagements zusammengetragen. Abschnitt 3 liefert einen Überblick über existierende öffentlich zugängliche Standard-APIs im Umfeld der elektronischen Signatur und diskutiert deren Eignung im Hinblick auf die vorher definierten Anforderungen. In Abschnitt 4 findet sich schließlich die Spezifikation der CCES-Signature-API.

2 Anforderungen an eine DMS-Signatur-Schnittstelle

In diesem Abschnitt sind die Anforderungen an eine Signatur-Schnittstelle im DMS-Umfeld zusammengetragen.

2.1 Geringer Integrationsaufwand

Die Integration von Signaturanwendungskomponenten soll möglichst geringen (einmaligen und anwendungskontextspezifischen) Integrationsaufwand verursachen. Dies impliziert insbesondere, dass es sich um eine „High-Level“-API handeln muss, die von vielen technischen Details abstrahiert.

2.2 Verfügbare Plattformen

Die Schnittstelle sollte idealerweise für verschiedene gängige Entwicklungsplattformen (C, C++, C#, Java, etc.) und Betriebssystemplattformen (MS Windows, Linux, Unix, etc.) zur Verfügung stehen.

2.3 Funktionalität

Über die Schnittstelle soll die im Folgenden beschriebenen Funktionalität, oder aber bei Bedarf eine entsprechende Teilmenge daraus, zur Verfügung gestellt werden können.

2.3.1 Erstellung und Verifikation elektronischer Signaturen

Bei der Erstellung und Verifikation elektronischer Signaturen sollen unterschiedliche Einsatzszenarien und verschiedene Signaturformate unterstützt werden.

2.3.1.1 Unterstützung unterschiedlicher Einsatzszenarien

Die Signatur-API soll die Erstellung und die Verifikation der elektronischen Signaturen in verschiedenen Einsatzszenarien optimal unterstützen. Deshalb sollte die API unterschiedliche

- Mengengerüste (Einzelsignatur oder Massensignatur)
- Qualitätsstufen (einfache el.S., fortgeschrittene el.S., qualifizierte el.S., qualifizierte el.S. mit Anbieterakkreditierung)
- Personal Security Environments (PSEs) (handschriftlich, Software-PSE, Hardwaretoken)
- Gültigkeitsmodelle (Kettenmodell, Schalenmodell)
- Systemarchitekturen (zentrale / dezentrale Signaturerzeugung)

unterstützen können.

Bei der serverbasierten Signaturerzeugung und –verifikation sollten verschiedene Einsatzszenarien (Stapelverarbeitung, Request-Response-Mechanismus etc.) unterstützt werden.

Außerdem muss die API die Erzeugung und Verifikation von Mehrfachsignaturen (parallel und sequentiell) unterstützen.

2.3.1.2 Unterstützung verschiedener Signaturformate

Während die CCES-Signature-API grundsätzlich beliebige Signaturformate unterstützen können sollte, so müssen insbesondere die folgenden Formate berücksichtigt werden:

- Cryptographic Message Syntax
- XML Digital Signature
- E-Mail-Signaturformate, wie z.B. S/MIME, OpenPGP und PGP/MIME
- Dokumentenformate mit eingebetteter Signatur

2.3.1.2.1 Cryptographic Message Syntax (CMS)

Das heute vielleicht am weitesten verbreitete Basisformat für kryptographisch behandelte Nachrichten ist in [RFC3369] definiert. Dieser Standard basiert[1], wie bereits seine Vorgängerversion [RFC2630], auf [PKCS#7] bzw. [RFC2315]. PKCS#7 ist wiederum – unter gewissen Umständen[2] – sogar mit den heute schon fast antik anmutenden PEM-Format [RFC1421] kompatibel.

Die Verwendung von CMS wurde in [ISIS-MTT] (Part 3 – Message Formats) näher profiliert.

2.3.1.2.2 XML Digital Signature

Auf Grund der in vielen Einsatz-Szenarien wachsenden Bedeutung von XML wird die Unterstützung der XML-Signatur [RFC3275] zunehmend wichtiger.

Die Verwendung der XML-Signatur wurde in [ISIS-MTT] (Part 8 – XML Signature and Encryption Profile) näher profiliert.

2.3.1.2.3 E-Mail-Signaturformate

Für die elektronische Signatur von E-Mail-Nachrichten und MIME-Anhängen sind die folgenden Formate besonders weit verbreitet:

- S/MIME

  Das mittlerweile im kommerziellen Umfeld wohl am weitesten verbreitete Format zur Signatur (und Verschlüsselung) von E-Mails und MIME-Anhängen ist in [RFC2633] spezifiziert.

  Die Verwendung von CMS wurde in [ISIS-MTT] (Part 3 – Message Formats) näher profiliert.

- OpenPGP und PGP/MIME

  Neben S/MIME wird zur Signatur (und Verschlüsselung) von E-Mails und MIME-Anhängen, insbesondere zur Ad-hoc-Kommunikation und in kleineren Benutzergruppen, auch das in [RFC2440] und [RFC3156] spezifizierte OpenPGP bzw. PGP/MIME eingesetzt.

2.3.1.2.4 Weitere Dokumentenformate mit eingebetteter Signatur

Neben den oben genannten Signaturformaten spielen im Umfeld des Dokumentenmanagements auch Formate eine Rolle, bei denen elektronische Signaturen in populäre Dokumentenformate eingebettet werden. Auch wenn in diesem Bereich noch genereller Standardisierungsbedarf zu existieren scheint, so sollten hier insbesondere folgende Formate berücksichtigt werden:

- PDF (siehe [AI-PDF])

[1] Inkompatibilitäten zwischen CMS und PKCS#7 ergeben sich lediglich, falls es sich bei den Nutzdaten nicht um den Typ `Data:: = OCTET STRING`, sondern um strukturierte Daten, wie z.B. ein `TSTInfo` Zeitstempel-Token aus [RFC3161], handelt (vgl. [RFC3369] Abschnitt 5.2.1.).
[2] Siehe [PKCS#7] Abschnitt 9.5.

- TIFF (siehe [AI-TIFF])

2.3.2 Anforderung und Verifikation von Zeitstempeln

Zur sicheren Verknüpfung von Zeitattributen mit Dokumenten spielen vertrauenswürdige Zeitstempel im DMS-Umfeld eine wichtige Rolle. Dies ist umso wichtiger, da bei der Prüfung einer qualifizierten elektronischen Signatur die Gültigkeit des Zertifikates zum Erstellungszeitpunkt der Signatur heranzuziehen ist.

Für das Ausstellen von Zeitstempeln wird zumeist das in [RFC3161] spezifizierte Time Stamp Protocol (TSP) verwendet. Beispielsweise haben sich alle Anbieter[3] von qualifizierten Zeitstempeln auf die Unterstützung dieses Protokolls verständigt.

Außerdem wird in der ISIS-MTT-Spezifikation [ISIS-MTT] (Part 4 – Operational Protocols) ohne nennenswerte Profilierungen auf das in [RFC3161] spezifizierte TSP verwiesen.

2.3.3 Erweiterte Funktionalität zur langfristigen Archivierung

Neben der naiven Methode zur langfristigen Sicherung des Beweiswertes (qualifizierter) elektronischer Signaturen durch Übersignatur bzw. Zeitstempelung einzelner Dokumente und Signaturen sollten über die API auch Mechanismen zur Verfügung gestellt werden, die eine effiziente Langzeitarchivierung gemäß § 17 SigV ermöglichen. Hierzu bietet sich die Unterstützung der in [BGPT03] spezifizierten Archive-Time-Stamp Syntax (ATS), bzw. der darauf aufbauend in [BrHu04] spezifizierten Evidence Record Syntax (ERS) an.

Die API sollte also die folgende Funktionalität bereitstellen:

- Erstellung und Prüfung von Archive Time Stamps
- Erstellung und Prüfung von Evidence Records
- Funktionen zur Realisierung des
    - Time Stamp Renewal (Übersignatur nur über Hashwerte)
    - Hash Tree Renewal (komplette Rekonstruktion des Hashbaumes)

2.3.4 Weitere kryptographische Operationen

Auch wenn die CCES-Signature-API primär die Aspekte der elektronischen Signatur berücksichtigen soll, so sollte die eingesetzte Technologie auch für weitere kryptographische Operationen genutzt werden können. Insbesondere sollte hier die Verschlüsselung von Daten in den Formaten CMS (vgl. Abschnitt 2.3.1.2.1), S/MIME und PGP (vgl. Abschnitt 2.3.1.2.3) unterstützt werden.

[3] Derzeit bieten Authentidate International AG, Datev eG, Deutsche Post Signtrust GmbH, D-Trust GmbH und TC Trustcenter AG qualifizierte Zeitstempel gemäß §2 Nr. 14 SigG an.

3 Existierende Standard-APIs und Ihre Eignung

In diesem Abschnitt werden existierende Standard-APIs betrachtet und im Hinblick auf Ihre Eignung, als Basis für die Spezifikation der CCES-Signature-API zu fungieren, bewertet. Neben der kurzen Diskussion der verschiedenen APIs hier verweisen wir auch auf [BaSch00] für weitere Informationen zu kryptographischen Standards und zugehörigen APIs.

3.1 Existierende Standard-APIs

3.1.1 Acrobat Digital Signature API

Bei der Acrobat Digital Signature API [AI-API-99] handelt es sich um eine Schnittstelle zwischen der Acrobat-Applikation und einem Acrobat-spezifischen Signatur-PlugIn. Der Fokus der vorgesehenen Funktionalität liegt im Integritätsschutz (ausgewählter Teile) einer PDF-Datei durch Hash- und Signaturmechanismen. Auch wenn bei der aktuellen Spezifikation dieser Schnittstelle [AI-API-03] mit dem „PubSec Layer“ eine auf den Einsatz von Public-Key-Mechanismen zugeschnittene Schnittstellenschicht existiert, so fehlen dennoch weite Teile der benötigten Funktionalität einer CCES-Signature-API. Insbesondere fehlt die Unterstützung kryptographisch gesicherter Zeitstempel und verschiedener Signatur- und Dokumentenformate.

3.1.2 Common Data Security Architecture / Common Security Services Manager (CDSA / CSSM)

Die CDSA [X/O-CDSA] wurde ursprünglich von Intel entwickelt und von der Open Group als umfassende Sicherheitsarchitektur spezifiziert. Neben den über die CSSM-API in einheitlicher Art und Weise nach außen zur Verfügung gestellten Sicherheitsdiensten, können Module mit zusätzlichen Diensten spezifiziert und angebunden werden. Die standardmäßig vorgesehene Funktionalität umfasst beispielsweise auch das Erstellen und Prüfen elektronischer Signaturen in Low-Level-Formaten wie PKCS#1 . Die Untersützung verschiedener Signaturformate müsste durch das (u.U. gleichzeitige) Laden von mehrereren Cryptographic Service Providers, oder aber über die Definition eines mächtigeren Erweiterungsmoduls, realisiert werden. Die Unterstützung von Zeitstempeln und weiteren Mechanismen zur langfristigen Archivierung müsste in einem noch zu spezifizierenden Erweiterungsmodul realisiert werden.

Während es durch die Erweiterbarkeit der CDSA und CSSM also generell möglich wäre, ein maßgeschneidertes Erweiterungsmodul und eine zugehörige API für die Integration der elektronischen Signatur in DMS-Anwendungen zu spezifizieren und als zusätzliches Modul in einer CDSA-Implementierung zu nutzen, so erscheint eine zwangsweise Kopplung der CCES-Signature-API mit CDSA/CSSM eher hinderlich. Dies gilt umso mehr, da CDSA/CSSM derzeit noch kaum[4] verbreitet zu sein scheint.

[4] Neben der mittlerweile als Open Source verfügbaren Referenz-Implementierung von Intel (http://sourceforge.net/projects/cdsa/) und der CDSA-Realisierung für Mac/OS von Apple

3.1.3 Generic Cryptographic Service API (GCS-API)

Die GCS-API [X/O-GCS] ist ein lediglich als vorläufige Spezifikation der Open Group verfügbarer Standardisierungsansatz, durch den auf verschiedene kryptographische Algorithmen über generische Aufrufe und Algorithmen-spezifische Templates zugegriffen werden kann. Da es sich hierbei eher um eine Low-Level-API handelt, die zudem wichtige Funktionen, wie z.B. die Validierung von Zertifikaten und die Erstellung von Zeitstempeln, vermissen lässt, ist die GCS-API als Basis für eine CCES-Signature-API wenig geeignet.

3.1.4 Generic Security Services API (GSS-API)

Bei der in [RFC2078] spezifizierten GSS-API handelt es sich um eine generische Schnittstelle zur sicheren Client-Server-Kommunikation, die beispielsweise im SAP R/3-Umfeld zum Einsatz kommt. Da der Zweck der GSS-API der Schutz von Client-Server-Kommunikationsverbindungen ist, ist sie für Zwecke der elektronischen Signatur generell ungeeignet.

3.1.5 GSS-Independent Data Unit Protection GSS-IDUP

Der GSS-IDUP-Standard [RFC2479] ist das in der Praxis vergleichsweise wenig verbreitete, dokumenten-basierte Pendant zur GSS-API. Hierin sind generische Sicherheitsmechanismen für Dokumente definiert, mit denen auch elektronische Signaturen, beispielsweise im PEM-Format [RFC1421], realisiert werden könnten. Leider existieren, anders als bei der in der Praxis eingesetzten GSS-API, noch keinerlei C- oder Java-Bindings im Stile von [RFC2744] bzw. [RFC2853]. Da die in [RFC2479] spezifizierte Funktionalität selbst noch um viele Aspekte ergänzt werden müsste, scheint die Verwendung der GSS-IDUP als Basis für die Spezifikation der CCES-Signature-API wenig Vorteile zu bieten.

3.1.6 Java Cryptographic Architecture (JCA) und Extension (JCE)

Die Java Cryptographic Architecture [JCA] in Verbindung mit der Java Cryptography Extension [JCE], die konkrete kryptographische Mechanismen enthält und aus Gründen der Exporterleichterung von der JCA separiert ist, bildet den weithin akzeptierten Standard für die Unterstützung kryptographischer Primitive in Java. Allerdings handelt es sich hierbei um eine Low-Level-API, die nicht als möglicher Ersatz, sondern vielmehr als Basis für Realisierung eines Java-basierten Providers, für die CCES-Signature-API zu sehen ist.

3.1.7 Java XML Digital Signature API

Bei der Java XML Digital Signature API [JXS-API] handelt es sich um eine Java-basierte Schnittstelle zur Realisierung von XML-Signaturen gemäß [RFC3275]. Hierbei können die in [RFC3275] definierten Signatur-Objekte in einem XML-

(http://developer.apple.com/documentation/Security/Conceptual/Security_Overview/Architecture/chapter_2_section_3.html) ist keine weitere CDSA-Implementierung bekannt.

Dokument angelegt, näher spezifiziert und schließlich signiert und geprüft werden. Während der generelle Charakter dieser Schnittstelle dem der anvisierten CCES-Signature-API ähnelt, so deckt die Java-basierte Schnittstelle [JXS-API] unglücklicherweise nur einen Ausschnitt der benötigten Funktionalität ab. Eine Unterstützung von Zeitstempeln und verschiedenen Signaturformaten ist nicht vorgesehen.

3.1.8 Microsoft CryptoAPI

Die Microsoft CryptoAPI [MS-CAPI] stellt Windows-basierten Applikationen verschiedene kryptographische Mechanismen, wie z.B. die Erzeugung und Verifikation von Signaturen im PKCS#7-Format oder die Prüfung eines Zertifikatspfades nach dem Schalenmodell, bezüglich des Prüfungszeitpunktes[5], zur Verfügung. Die Implementierung der kryptographischen Mechanismen selbst erfolgt durch Cryptographic Service Provider. Auch hier entspricht der generelle Charakter dieser Schnittstelle dem der anvisierten CCES-Signature-API. Leider wird auch hier lediglich ein Ausschnitt der benötigten Funktionalität abgedeckt. Es fehlt die Unterstützung von Zeitstempeln und verschiedenen Signaturformaten.

3.1.9 Open Card Framework (OCF)

Das Open Card Framework [OCF] definiert eine Schnittstelle zur Java-nativen Ansteuerung von Chipkarten. Durch den Low-Level-Charakter ist diese Schnittstelle als Basis für die Spezifikation der CCES-Signature-API generell ungeeignet.

3.1.10 PC/SC und CT-API

Bei der PC/SC-Schnittstelle [PC/SC] handelt es sich um eine Schnittstelle zwischen, ursprünglich Windows-basierten, PCs und Chipkartenlesern. Ähnliches leistet die CT-API [CT-API] auf verschiedenen weiteren Plattformen. Deshalb sind diese Schnittstellen für die Spezifikation der CCES-Signature-API grundsätzlich nicht zu gebrauchen.

3.1.11 PKCS#11

Die Cryptographic Token Interface (CrypTokI) Schnittstelle [PKCS#11] definiert eine Low-Level-Schnittstelle zwischen Anwendungssystemen und kryptographischen Token, wie z.B. Chipkarten oder Hardware-Sicherheitsmodulen. Diese Schnittstelle im Rahmen von [ISIS-MTT] (Part 7 – Cryptographic Token Interface) näher profiliert und vergleichsweise weit verbreitet. Da der Funktionsumfang aber auf die Kommunikation mit kryptographischen Token fokusiert ist, ist die Verwendung dieser Schnittstelle zur Spezifikation der CCES-Signature-API eher ungeeignet.

[5] Da für die Gültigkeit einer qualifizierten elektronischen Signatur definitionsgemäß der Erstellungszeitpunkt entscheidend ist, führt die naive Verwendung der Microsoft CryptoAPI bei der Verifikation von qualifizierten elektronischen Signaturen zu Problemen.

3.1.12 SAP Secure Store and Forward (SSF) API

Mit der SSF-API von SAP [SAP-SSF] wurde eine High-Level-API spezifiziert, mit der Daten im PKCS#7-Format signiert und verschlüsselt werden können. Auch hier entspricht der generelle Charakter dieser Schnittstelle dem der anvisierten CCES-Signature-API. Leider wird auch hier lediglich ein Ausschnitt der benötigten Funktionalität abgedeckt. Es fehlt an der Unterstützung von Zeitstempeln und verschiedenen Signaturformaten, so dass in der Praxis oft über proprietäre Schnittstellen auf zusätzliche Funktionen zugegriffen werden muss.

3.1.13 Signaturbündnis-API

Im Rahmen des Signaturbündnisses wird derzeit eine Programmierschnittstelle [SigAll-API] entwickelt, die es erlaubt in einheitlicher Art und Weise auf Signaturkarten verschiedener Anbieter zuzugreifen. Somit besitzt diese Schnittstelle einen ähnlichen Charakter wie PKCS#11 und ist daher nicht als Ersatz sondern vielmehr als Ergänzung zur CCES-Signatur-API zu verstehen.

3.1.14 Simple Cryptographic Program Interface (Crypto API)

Mit dem Simple Cryptographic Program Interface [RFC2628] wurde eine einfache Programmierschnittstelle spezifiziert, die die notwendige kryptographische Basis-Funktionalität zum Aufbau von Virtuellen Privaten Netzen mittels IPSEC oder SSL/TLS bereitstellt. Deshalb sieht diese Schnittstelle derzeit insbesondere Funktionen zur Authentisierung, Ver- und Entschlüsselung sowie Komprimierung vor. Die Nutzung der Schnittstelle zur Signatur von Dokumenten wäre lediglich unter Verwendung von Low-Level-Formaten, wie z.B. PKCS#1 [RFC3447], möglich. Die notwendige High-Level-Funktionalität zur Behandlung von Zertifikaten und Zeitstempeln fehlt leider gänzlich.

3.1.15 US Government Smart Card Interoperability Specification (GSC-IS)

Die vom NIST standardisierte Government Smart Card Interoperability Specification (GSC-IS) der US-Verwaltung [US-GSC-IS] definiert eine generische Schnittstelle für den Zugriff auf verschiedenartige Chipkarten. Neben Chipkarten mit ISO7816-basiertem Dateisystem werden beispielsweise auch Java-Karten über das einheitliche “Virtual Card Edge Interface” angesprochen. Die generelle Zielsetzung dieser Schnittstelle ist somit im Wesentlichen vergleichbar mit der von Chipkartenspezifikationen im Stile von [TTT-GOIC] und [HPC-Spec] und damit als Basis für die CCES-Signature-API wenig geeignet.

3.2 Zusammenfassende Bewertung der Eignung existierender APIs

Betrachtet man die Komponenten beim Signaturanwender, so ist im Wesentlichen zwischen der Signaturerstellungseinheit (vgl. §2 Nr. 10 SigG), der Signaturanwendungskomponente (vgl. §2 Nr. 11 SigG) und dem eigentlichen Anwendungssystem zu unterscheiden. Die Taxonomie der hier betrachteten Schnittstellen auf Basis dieses einfachen Modells liefert das folgende Bild:

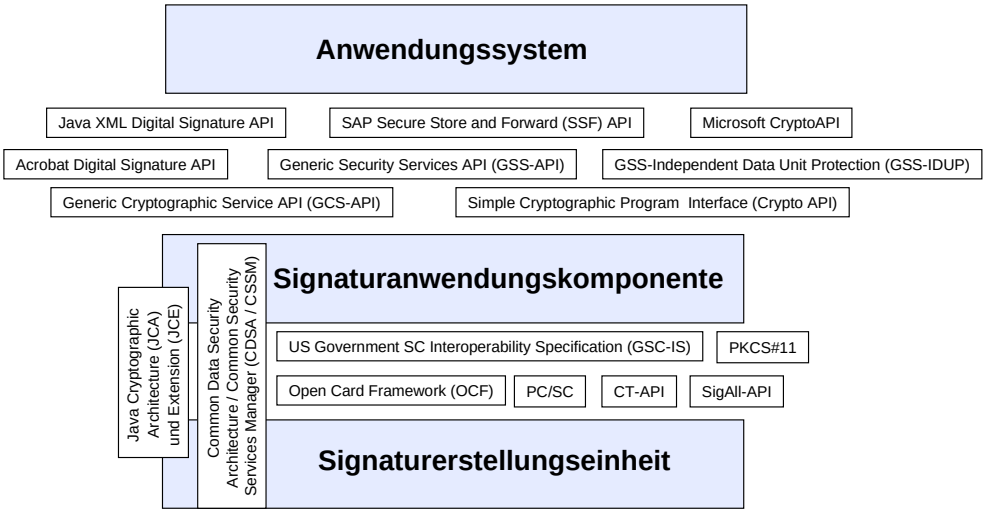

Abbildung 1: Existierende APIs im Überblick

Betrachtet man die oben aufgeführten APIs, dann fällt auf, dass sich einige Standard-Schnittstellen generell nicht als Basis für die CCES-Signature-API eignen, da sie auf einer zu tiefen, Chipkarten-nahen Schicht operieren und dem Anwendungsprogrammierer noch zu viele Aufgaben bei der Integration der elektronischen Signatur überlassen würden. Aus diesem Grund sind die Standards [OCF], [PC/SC], [CT-API], [PKCS#11], [SigAll-API] und [US-GSC-IS] generell wenig geeignet.

Auf der anderen Seite decken viele Standards, deren genereller Charakter dem der anvisierten CCES-Signature-API entspricht, lediglich eine mehr oder weniger große Untermenge der benötigten Funktionalität ab. Dies gilt beispielsweise für [AI-API-99], [AI-API-03], [X/O-GCS], [RFC2479], [JCA] / [JCE], [JXS-API], [MS-CAPI], [SAP-SSF] und [RFC2628]. Insbesondere fehlen hier bei all diesen Schnittstellen die für die DMS-Branche wichtigen Funktionen für Zeitstempel und zur langfristigen Archivierung (vgl. Abschnitte 2.3.2 und 2.3.3).

Deshalb wird in Abschnitt 4 die CCES-Signature-API als neue Schnittstelle spezifiziert, die die in Abschnitt 2 spezifizierten Anforderungen erfüllt.

4 Die CCES-Signature-API

In diesem Abschnitt findet sich die Spezifikation der CCES-Signature-API. Neben einer sprachunabhängigen Grobspezifikation der Schnittstelle in Abschnitt 4.1 finden sich in Abschnitt 4.2 eine präzise Spezifikation der Datenstrukturen und Aufrufe für C. Die Spezifikation einer Java-basierten CCES-Signature-API erfolgt in einer zukünftigen Version.

4.1 Sprachunabhängige Grobspezifikation der CCES-Signature-API

Neben administrativen Funktionen zur Verwaltung der Signatur-Provider für die CCES-Signature-API und der verwendeten PSEs[6], auf die an dieser Stelle nicht näher eingegangen werden soll, muss die CCES-Signature-API insbesondere die folgenden Funktionen vorsehen:

- Sign
- Hash
- CountSignatures
- Verify
- ValidateCertificate
- GetTimestamp
- VerifyTimestamp
- CreateEvidenceRecord
- RenewEvidenceRecord
- VerifyEvidenceRecord
- Encrypt
- Decrypt

4.1.1 Sign

Mit der `Sign`-Funktion wird die Erstellung elektronischer Signaturen ermöglicht. Um die in Abschnitt 2.3.1 zusammengetragenen Anforderungen zu ermöglichen, muss diese Funktion zumindest folgende Aufrufparameter und Optionen vorsehen:

- Informationen zu signierende Daten in verschiedenen Formen, wie z.B.
    - Daten selbst
    - Hashwert der zu signierenden Daten
    - Verweis auf zu signierende Daten oder Hashwert
- ggf. existierende Signaturen, die bei der Erstellung der Signatur berücksichtigt werden sollen
- ggf. zu verwendende Attribut-Zertifikate
- zu verwendendes Signaturformat, wie z.B.
        - CMS ([RFC3369])
        - XML-DSig ([RFC3275])
        - S/MIME ([RFC2633])

[6] „Personal Security Environment“ zur Aufbewahrung und Anwendung von (Signatur-) Schlüsseln (z.B. Chipkarte oder mit einem Passwort verschlüsselte Daten).

- PGP ([RFC2440])
- Integrierte Daten- und Signaturformate, wie z.B. Signatur in [AI-PDF]
- etc.

mit weiteren formatspezifischen Optionen.

Der Rückgabewert der Funktion `Sign` ist die erzeugte elektronische Signatur.

4.1.2 Hash

Mit der `Hash`-Funktion wird die Erstellung von Hashwerten ermöglicht, die in den Funktionen `Sign`, `GetTimestamp` oder `CreateEvidenceRecord` verwendet werden können. Die Funktion muss zumindest folgende Aufrufparameter und Optionen vorsehen:

- Daten oder Verweis auf Daten,
- Zu verwendende Hash-Funktion, wobei zumindest folgende berücksichtigt werden sollten:
  - SHA-1
  - RIPEMD-160
  - SHA-256

4.1.3 CountSignatures

Mit der `CountSignatures`-Funktion kann die Anzahl der in einem Datencontainer vorhandenen Signaturen ermittelt werden. Diese Funktion ist insbesondere für die Unterstützung von Mehrfachsignaturen bedeutsam.

4.1.4 Verify

Die `Verify`-Funktion muss in der Lage sein, die mit `Sign` erzeugten Signaturen zu verifizieren.

Hierzu müssen zumindest die folgenden Parameter und Optionen vorgesehen werden:

- Signatur der Daten
- Signierte Daten
- Prüfoptionen, wie z.B. keine Prüfung, lokale Prüfung oder Online-Prüfung der Zertifikatsstatus etc.

Existieren mehrere Signaturen in einem Signaturcontainer dann soll ausgewählt werden können, ob alle existierenden Signaturen oder nur eine bestimmte Signatur geprüft werden soll.

Der Rückgabewert der Funktion `Verify` ist das Ergebnis der Signaturprüfung. Das für die Prüfung zu verwendende Gültigkeitsmodell (Kette, Schale etc.) wird nicht als Parameter übergeben, sondern sollte anhand des Zertifikatstyps und der dadurch implizierten Policy von einer CCES-Signature-Library selbstständig ermittelt werden können.

4.1.5 ValidateCertificate

Mittels der `ValidateCertificate`-Funktion kann der Status eines Zertifikates ermittelt werden. Somit kann die ggf. implizit in `Verify` aufgerufene Funktionalität zur Prüfung eines Zertifikates separat angesprochen werden.

Hierzu müssen zumindest die folgenden Parameter und Optionen vorgesehen werden:

- Zertifikatskette
- Referenzzeitpunkt
- Angestrebte Schlüsselverwendung

Das für die Prüfung zu verwendende Gültigkeitsmodell (Kette, Schale etc.) wird nicht als Parameter übergeben, sondern sollte anhand des Zertifikatstyps von einer CCES-Signature-Library selbstständig ermittelt werden können.

4.1.6 GetTimestamp

Mit der `GetTimestamp`-Funktion wird das Anfordern von Zeitstempeln unterstützt. Hier sollten zumindest folgende Aufrufparameter und Optionen vorgesehen werden:

- Informationen zu den mit einem Zeitstempel zu versehenden Daten in verschiedenen Formen, wie z.B.
    - Daten selbst
    - Hashwert der Daten
    - Verweis auf Daten oder Hashwert
- Zeitstempeloptionen, die beispielsweise bei einer Vielzahl übergebener Daten aussagen ob
    - mehrere Archive Time Stamps gemäß [BrHu04],
    - ein einziger Archive Time Stamp gemäß [BrHu04] oder
    - mehrere einfache TSP-Zeitstempel gemäß [RFC3161]

  erzeugt werden sollen.

Der Rückgabewert der Funktion `GetTimestamp` ist der – typischerweise von einem entfernt residierenden, vertrauenswürdigen Zeitstempeldienst – erzeugte Zeitstempel.

4.1.7 VerifyTimestamp

Mit `VerifyTimestamp` müssen die mit `GetTimestamp` erzeugten Zeitstempel verifiziert werden können. Hierzu müssen zumindest folgende Aufrufparameter und Optionen vorgesehen werden:

- Zeitstempel
- Daten (oder Hashwert oder Verweis auf Daten oder Hashwert)
- Prüfpolicy (keine, lokale oder Online-Prüfung der Zertifikate)

Der Rückgabewert der Funktion `VerifyTimestamp` ist das Ergebnis der Zeitstempelprüfung.

4.1.8 CreateEvidenceRecord

Mit `CreateEvidenceRecord` kann ein in [BrHu04] spezifizierter EvidenceRecord erzeugt werden. Hierzu müssen zumindest die Daten übergeben werden, aus denen der EvidenceRecord erzeugt werden soll. Außerdem sollten folgende Optionen für die Übergabe vorgesehen werden:

- Unverschlüsselten Daten selbst
- Verschlüsselte Daten
- Hashwert der Daten
- Verweis auf (verschlüsselte oder unverschlüsselte) Daten oder Hashwert

Der Rückgabewert der Funktion `CreateEvidenceRecord` ist ein EvidenceRecord.

4.1.9 RenewEvidenceRecord

Mit dem `RenewEvidenceRecord` kann ein bereits existierender EvidenceRecord erneuert werden. Hierbei müssen zumindest folgende Übergabeparameter und Optionen vorgesehen werden:

- Zu erneuernder EvidenceRecord
- Information ob lediglich ein Time Stamp Renewal (Übersignatur nur über Hashwerte) oder ein Hash Tree Renewal (Rekostruktion des Hash-Baumes) durchgeführt werden soll
- ggf. Daten wie bei `CreateEvidenceRecord` (für Hash Tree Renewal)

Der Rückgabewert der Funktion `RenewEvidenceRecord` ist der erneuerte EvidenceRecord.

4.1.10 VerifyEvidenceRecord

Mit `VerifyEvidenceRecord` kann ein EvidenceRecord auf seine Gültigkeit hin überprüft werden. Hierzu sind zumindest folgende Übergabeparameter zu berücksichtigen:

- EvidenceRecord
- ggf. Daten wie bei `CreateEvidenceRecord`
- Prüfpolicy (keine, lokale oder Online-Prüfung der Zertifikate)

Der Rückgabewert der Funktion `VerifyEvidenceRecord` ist das Ergebnis der Prüfung des EvidenceRecord.

4.1.11 Encrypt

Mit der `Encrypt`-Funktion wird die Verschlüsselung von Daten ermöglicht. Hierfür sind zumindest folgende Aufrufparameter und Optionen vorzusehen:

- Zu verschlüsselnde Daten
- (Zertifikate der) Empfänger für die verschlüsselt werden soll
- Verschlüsselungsoptionen, wie z.B.
    - Prüfpolicy (keine, lokale oder Online-Prüfung der Verschlüsselungszertifikate)
    - zu verwendendes Verschlüsselungsformat, wie z.B.
        - CMS ([RFC3369])
        - S/MIME ([RFC2633])
        - PGP ([RFC2440])

Der Rückgabewert der `Encrypt`-Funktion sind die verschlüsselten Daten.

4.1.12 Decrypt

Mit der `Decrypt`-Funktion wird die Entschlüsselung von Daten ermöglicht. Hierzu müssen lediglich die verschlüsselten Daten übergeben werden.

Der Rückgabewert der `Decrypt`-Funktion sind die entschlüsselten Daten.

4.2 C-Bindings der CCES-Signature-API

In diesem Kapitel finden sich die C-Bindings der CCES-Signature-API. Neben der Definition der notwendigen Datenstrukturen in Abschnitt 4.2.1 und der Spezifikation administrativer Funktionen in Abschnitt 4.2.2 finden sich in den folgenden Abschnitten auch präzise Spezifikationen der oben skizzierten Aufrufe.

4.2.1 Definition der Datenstrukturen

4.2.1.1 Allgemeine Datenstrukturen

Im Rahmen der CCES-Signature-API werden folgende allgemeine Datenstrukturen definiert:

- Datenstrukturen zur Fehlerbehandlung
- CCES_InstanceHandle
- CCES_QueryPSEHandle
- CCES_DataContainer und CCES_DataType
- CCES_Time
- CCES_Number
- CCES_KeyUsage

4.2.1.1.1 Datenstrukturen zur Fehlerbehandlung

typedef unsigned long CCES_ErrorCode

typedef enum {

CCES_NoError,

CCES_UnknownError,

CCES_UnsupportedLanguage,

CCES_UserCancel,

CCES_OutOfMemory,

CCES_InvalidParam,

CCES_FunctionNotSupported,

CCES_FunctionBlockedByLicence,

CCES_FeatureNotSupported,

CCES_FeatureBlockedByLicence,

CCES_InvalidDataContainer,

CCES_InvalidDataType,

CCES_InvalidPSEHandle,

CCES_InvalidLibHandle,

CCES_NoMorePSEs,

CCES_PSEInitFailed,

CCES_PSENotAvailable,

```
        CCES_CertificateChainError,
        CCES_NotExistingSignature,
        CCES_OCSPConnectError,
        CCES_OCSPRespSigVerifyError,
        CCES_OCSPDecodeResponseError,
        CCES_TSPConnectError,
        CCES_InvalidCertificate,
        CCES_CertificateNotFound,
        CCES_CannotDecryptData,
        CCES_BadPin,
        CCES_ChangePinFailed
};
```

Diese Fehler haben die folgende Bedeutung:

- CCES_NoError
  Es trat kein Fehler auf.
- CCES_UnknownError
  Der Fehlercode ist nicht bekannt.
- CCES_UnsupportedLanguage
  Die Sprache wird nicht unterstützt.
- CCES_UserCancel
  Der Benutzer hat die Aktion abgebrochen.
- CCES_OutOfMemory
  Es steht nicht genügend Speicherplatz zur Verfügung.
- CCES_InvalidParam
  Es wurde ein ungültiger Parameter übergeben.
- CCES_FunctionNotSupported
  Die aufgerufene Funktion wird nicht unterstützt.
- CCES_FunctionBlockedByLicence
  Die aufgerufene Funktion wird technisch unterstützt, ist aber auf Grund von Lizenzbestimmungen gesperrt.
- CCES_FeatureNotSupported
  Das gewünschte Feature (z.B. gewähltes Signaturformat) wird nicht unterstützt.

- CCES_FeatureBlockedByLicence

  Das gewünschte Feature wird technisch unterstützt, ist aber auf Grund von Lizenzbestimmungen gesperrt.
- CCES_InvalidDataContainer

  Der übergebene Datencontainer ist ungültig.
- CCES_InvalidDataType

  Der übergebene Datentyp ist ungültig.
- CCES_InvalidPSEHandle

  Das übergebene PSEHandle ist ungültig.
- CCES_InvalidLibHandle

  Das übergebene LibHandle ist ungültig.
- CCES_NoMorePSEs

  Es gibt keine weiteren PSEs – die Aufzählfunktion ist am Ende angelangt.
- CCES_PSEInitFailed

  Die Initialisierung der gewählten PSE schlug fehl.
- CCES_PSENotAvailable

  Die gewählte PSE ist nicht verfügbar.
- CCES_CertificateChainError

  Beim Aufbau des Zertifikatspfades ist ein Fehler aufgetreten.
- CCES_NotExistingSignature,

  Es existiert keine Signatur mit dem angegebenen Signatur-Index.
- CCES_OCSPConnectError

  Der OCSP-Responder ist nicht erreichbar.
- CCES_OCSPRespSigVerifyError

  Die Signatur an der OCSP-Antwort ist nicht gültig.
- CCES_OCSPDecodeResponseError

  Beim Decodieren der OCSP-Antwort ist ein Fehler aufgetreten.
- CCES_TSPConnectError

  Der Zeitstempeldienst ist nicht erreichbar.
- CCES_InvalidCertificate

  Das übergebene Zertifikat ist ungültig.
- CCES_CertificateNotFound

  Das angegebene Zertifikat kann nicht gefunden werden.
- CCES_CannotDecryptData

Die Daten können nicht entschlüsselt werden.

- CCES_BadPin
  Die angegebene PIN ist ungültig.
- CCES_ChangePinFailed
  Beim Ändern der PIN ist ein Fehler aufgetreten.

4.2.1.1.2 CCES_InstanceHandle

typedef unsigned long CCES_InstanceHandle

4.2.1.1.3 CCES_QueryPSEHandle

typedef unsigned long CCES_QueryPSEHandle

4.2.1.1.4 CCES_DataContainer und CCES_DataType

```
typedef unsigned char BYTE;
typedef BYTE* PBYTE;

typedef struct {
        PBYTE           pcDataBuffer;
        Int             iDataBufferSize;
        CCES_DataType   eDataType;}
CCES_DataContainer;

typedef enum {
        e_FileType,
        e_HashType,
        e_HashTreeType,
        e_OctetStringType,
        e_MIMEType,
        e_CMSType,
        e_SMIMEType,
        e_PGPType,
        e_XMLType,
        e_PDFType,
        e_TSPTokenType,
```

```
        e_ATSTokenType,
        e_EvidenceRecordType,
        e_ASCIIType,
        e_CertificateType,
        e_CertificateListType,
        e_AbstractASN1Type}
CCES_DataType;
```

Die interne Struktur der Datencontainer ist abhängig vom Typ folgendermaßen gegeben:

- e_FileType

  Der Datencontainer enthält eine Liste mit einem oder mehreren URLs, oder lokalen Dateinamen, die mit \0 voneinander getrennt sind.

- e_HashType

  Der Datencontainer enthält ein ASN.1 DER-codiertes Datenelement, das folgendermaßen definiert ist:

  ```
  DigestInfo ::= SEQUENCE {
      digestAlgorithm AlgorithmIdentifier,
      digest OCTET STRING}
  ```

- e_HashTreeType

  Der Datencontainer enthält ein ASN.1 DER-codiertes Datenelement, das folgendermaßen definiert ist:

  ```
  HashTree ::= SEQUENCE {
      digestAlgorithm AlgorithmIdentifier,
      reducedHashtree [0] SEQUENCE OF SEQUENCE
                  OF OCTET STRING }
  ```

- e_OctetStringType

  Der Datencontainer enthält einen OctetString.

- e_MIMEType

  Der Datencontainer enthält eine E-Mail-Nachricht im MIME-Format ([RFC2045]).

- e_CMSType

  Der Datencontainer enthält einen CMS-Datencontainer gemäß [RFC3369].

- e_SMIMEType

  Der Datencontainer enthält eine S/MIME-Nachricht gemäß [RFC2633].

- e_PGPType

  Der Datencontainer enthält eine OpenPGP-Nachricht gemäß [RFC2440].

- e_XMLType

  Der Datencontainer enthält eine XML-Datei gemäß [XML].

- e_PDFType

  Der Datencontainer enthält eine PDF-Datei gemäß [AI-PDF].

- e_TSPTokenType

  Der Datencontainer enthält ein TimeStamp-Token gemäß [RFC3161].

- e_ATSTokenType

  Der Datencontainer enthält ein ArchiveTimeStamp-Token gemäß [BrHu04].

- e_EvidenceRecordType

  Der Datencontainer enthält einen EvidenceRecord gemäß [BrHu04].

- e_ASCIIType

  Der Datencontainer enthält eine Folge von ASCII-Zeichen.

- e_CertificateType

  Der Datencontainer enthält ein x.509-Zertifikat.

- e_CertificateListType

  Der Datencontainer enthält eine Liste von Verweisen auf Zertifikate im Format "SerialNumber1;IssuerName1\0 SerialNumber2;IssuerName2\0…".

- e_AbstractASN1Type

  Während die vorliegende Spezifikation der CCES-Siganture-API bereits sehr viele Datentypen und Signaturformate unterstützt, so kann nicht ausgeschlossen werden, dass zukünftige andere Datenstrukturen und Signaturformate genutzt werden müssen.

  Um die Nutzung von heute noch nicht spezifizierten Formaten nutzen zu können, wird der e_AbstractASN1Type als ASN.1-Objekt vom Typ TYPE-IDENTIFIER definiert. In [ISO8824-2] ist dieser Typ folgendermaßen definiert:

  ```
  TYPE-IDENTIFIER ::= CLASS {
        &id OBJECT IDENTIFIER UNIQUE,
        &Type }
  WITH SYNTAX {&Type IDENTIFIED BY &id}
  ```

4.2.1.1.5 CCES_Time

typedef long CCES_Time;

4.2.1.1.6 CCES_Number

typedef long CCES_Number;

4.2.1.1.7 CCES_KeyUsage

Die Datenstruktur CCES_KeyUsage entspricht dem in [RFC3280] definierten Bitstring. Diese Datenstruktur wird zur Spezifikation der vorgesehenen Schlüsselnutzung bei der in Abschnitt 4.2.7 definierten Routine ValidateCertificate verwendet.

```
typedef enum
{
        e_digitalSignature      = 0x00000001,
        e_nonRepudiation        = 0x00000002,
        e_keyEncipherment       = 0x00000004,
        e_dataEncipherment      = 0x00000008,
        e_keyAgreement          = 0x00000010,
        e_keyCertSign           = 0x00000020,
        e_cRLSign               = 0x00000040,
        e_encipherOnly          = 0x00000080,
        e_decipherOnly          = 0x00000100
} CCES_KeyUsage;
```

Die Key Usage Bits sind durch eine logische Oderverknüpfung auf Bitebene miteinander kombinierbar. So bedeutet die übergebene Key Usage 0x00000003, dass der Schlüssel zur Erstellung von Signaturen (e_digitalSignature) und für Zwecke der Verbindlichkeit (e_nonRepudiation) eingesetzt werden kann.

4.2.1.1.8 CCES_LibraryInfo

Alle CCES_DataContainer in der folgenden Datenstruktur sind vom Typ e_ASCIIType.

```
typedef struct{
        CCES_DataContainer      APIVersion;
        CCES_DataContainer      ManufacturerID;
        CCES_DataContainer      LibraryDescription;
        CCES_DataContainer      LibraryVersion;
}CCES_LibraryInfo;
```

4.2.1.2 Datenstrukturen zur Spezifikation von Aufrufoptionen

Im Rahmen der CCES Signature API werden folgende Datenstrukturen für Aufrufoptionen definiert:

- CCES_SignOptions
- CCES_HashOptions
- CCES_VerifyOptions
- CCES_TimeStampOptions
- CCES_ERRenewalOptions
- CCES_EncryptOptions

4.2.1.2.1 CCES_SignOptions

Die CCES_SignOptions bestehen aus zwei unabhängigen Teilen. Neben dem „High-Level-Signatur-Format“ kann ein optionaler Object Identifier (OID) als mit „\0“ terminierender String übergeben werden, der einen „Low-Level-Signatur-Algorithmus“ referenziert. Fehlt die explizite Angabe der OID für den Signatur-Algorithmus, so wird der in PKCS#1 definierte Standardwert sha-1WithRSAEncryption (OID= 1.2.840.113549.1.1.5) verwendet.

OIDs für alternative Signaturalgorithmen finden sich beispielsweise auch in [ISIS-MTT] (Part 6 – Cryptographic Algorithms, Table 2 – Signature Algorithms).

```
typedef struct {
        CCES_HighLevelSignatureFormat    pHLSFormat;
        PBYTE                            OID;
} CCES_SignOptions;

typedef enum
{
        e_NoHighLevelFormat,
        e_CMSEnveloping,
        e_CMSDetached,
        e_SMIMEClearSigned,
        e_SMIMEOpaqueSigned,
        e_PGP,
        e_XML,
        e_PDF
} CCES_HighLevelSignatureFormat;
```

4.2.1.2.2 CCES_HashOptions

Die CCES_HashOptions bestehen aus zwei unabhängigen Teilen. Neben der gewünschten Ausgabeform kann ein optionaler Object Identifier (OID) als mit „\0“ terminierender String übergeben werden, der den zu verwendenden Hash-Algorithmus referenziert. Fehlt die explizite Angabe der OID für den Hash-Algorithmus, so wird standardmäßig SHA-1 (OID= 1.3.14.3.2.26) verwendet. OIDs für alternative Hashalgorithmen finden sich beispielsweise auch in [ISIS-MTT] (Part 6 – Cryptographic Algorithms, Table 1 – One-Way-Hash-Functions).

```
typedef struct {
        CCES_HashOutputForm     pHOForm;
        PBYTE                   oid;
} CCES_HashOptions;

typedef enum
{
        e_manyHashes,
        e_singleHashtree
} CCES_HashOutputForm;
```

4.2.1.2.3 CCES_VerifyOptions

```
typedef enum
{
        e_NoCertificateCheck            = 0x00000000,
        e_CheckCertificateWithOCSP      = 0x00000001,
        e_CheckCertificateWithCRL       = 0x00000002,
        e_CheckAlgorithms               = 0x00000004,
        e_StoreCertStatusEvidence       = 0x00000008,
        e_ShowProgress                  = 0x00000010,
        e_NoUserInterface               = 0x00000020,
} CCES_VerifyOptions;
```

Diese Optionen sind, soweit sinnvoll, durch eine logische Oderverknüpfung auf Bitebene miteinander kombinierbar. So bedeutet die übergebene Option 0x00000011, dass eine Online-Prüfung per OCSP durchgeführt werden soll (0x00000001) und gleichzeitig der Fortschritt angezeigt werden soll (0x00000010).

4.2.1.2.4 CCES_TimeStampOptions

typedef enum

{

e_TSPs	= 0x00000001,
e_ATSs	= 0x00000002,
e_singleATS	= 0x00000004,
e_signRequest	= 0x00000010

} CCES_TimeStampOptions;

Die Optionen e_TSPs, e_ATSs und e_singleATS können jeweils mit der Option e_signRequest kombiniert werden. Beispielsweise bedeutet die übergebene Option 0x00000011, dass TimeStamp-Token gemäß [RFC3161] erzeugt werden, wobei die Authentisierung durch Signatur des Requests mit einer CMS-basierten Signatur realisiert wird.

4.2.1.2.5 CCES_ERRenewalOptions

Für die Erneuerung von EvidenceRecords stehen folgende Optionen zur Verfügung:

typedef enum

{

e_TimeStampRenewal,

e_HashTreeRenewal,

} CCES_ ERRenewalOptions

4.2.1.2.6 CCES_EncryptOptions

Die CCES_EncryptOptions bestehen aus zwei unabhängigen Teilen. Neben dem „High-Level-Verschlüsselungs-Format“ können optionale Object Identifier (OID) als mit „\0“ terminierender String übergeben werden, die die Algorithmen zur ContentEncryption und KeyEncryption spezifizieren. Wird von diesen Optionen kein Gebrauch gemacht, so werden die folgenden Standard-Algorithmen verwendet:

- ContentEncryption: des-ede3-cbc (OID=1.2.840.113549.3.7)
- KeyEncryption : rsaEncryption (OID=1.2.840.113549.1.1.1)

OIDs für alternative Verschlüsselungsalgorithmen finden sich beispielsweise auch in [ISIS-MTT] (Part 6 – Cryptographic Algorithms, Table 3 – Content Encryption Algorithms und Table 5 – Key Encryption Algorithms).

typedef struct {

CCES_ HighLevelEncryptionFormat	pHLEFormat;
PBYTE	ContentEncryptionOID;

```
        PBYTE                                   KeyEncryptionOID;
} CCES_EncryptOptions;

typedef enum
{
        e_CMS,
        e_SMIME,
        e_PGP
} CCES_ HighLevelEncryptionFormat;
```

4.2.1.2.7 CCES_GetContentTypeOptions

```
typedef enum
{
        e_ReadTypeFromContainer,
        e_DetermineType,
        e_CompletelyParseContainer
} CCES_GetContentTypeOptions;
```

4.2.1.3 Datenstrukturen für Prüfergebnisse

Im Rahmen der CCES Signature API werden folgende Datenstrukturen für Prüfergebnisse definiert:

- CCES_VerificationResultArray
- CCES_VerificationResult
- CCES_CheckSignature
- CCES_CheckCertificatePath
- CCES_CheckSingleCert
- CCES_CertStatus
- CCES_CheckResult

4.2.1.3.1 CCES_VerificationResultArray

Mit diesem Datentyp kann eine Vielzahl von Prüfergebnissen vom Typ CCES_VerificationResult zurückgeliefert werden.

```
typedef CCES_VerificationResult[ ] CCES_VerificationResultArray
```

4.2.1.3.2 CCES_VerificationResult

Das Prüfergebnis CCES_VerificationResult für eine elektronische Signatur besteht im Wesentlichen aus drei Bestandteilen:

- Dem Prüfzeitpunkt,
- den Ergebnissen der mathematischen Signaturprüfung und
- den Ergebnissen der Validierung des Zertifikatspfades.

```
typedef struct {
        CCES_Time                   verificationTime;
        CCES_CheckSignature         signatureOK;
        CCES_CheckCertificatePath   certChainOK;
} CCES_VerificationResult;
```

4.2.1.3.3 CCES_CheckSignature

Bei der mathematischen Prüfung der Signatur werden folgende Prüfungen durchgeführt:

- Prüfung der Gültigkeit des Formates der Signatur
- Prüfung der Integrität der Angaben in einem Signaturmanifest (bei XML-Signaturen)
- Prüfung der mathematischen Gültigkeit der Signatur
- Prüfung der Eignung des Hash-Algorithmus, sofern die Option e_CheckAlgorithms gesetzt wurde
- Prüfung der Eignung des Signatur-Algorithmus, sofern die Option e_CheckAlgorithms gesetzt wurde

```
typedef struct {
        CCES_CheckResult            formatOK;
        CCES_CheckResult            manifestOK;
        CCES_CheckResult            sigMathOK;
        CCES_CheckResult            hashAlgUpToDate;
        CCES_CheckResult            sigAlgUpToDate
} CCES_CheckSignature;
```

4.2.1.3.4 CCES_CheckCertificatePath

Die Prüfung des Zertifikatspfades besteht aus der Prüfung einer Liste von Zertifikaten und der Prüfung ob das Wurzelzertifikat vertrauenswürdig ist.

```
typedef struct {
        CCES_CheckSingleCert        certificatePath[ ];
        CCES_CheckResult            rootTrusted;
}CCES_CheckCertificatePath;
```

4.2.1.3.5 CCES_CheckSingleCert

Bei der Prüfung eines einzelnen Zertifikates werden folgende Punkte abgeprüft:

- Prüfung der Verkettung
- Prüfung des Gültigkeitszeitraumes des Zertifikates
- Prüfung, ob Erweiterungen OK sind. Dies umfasst folgende Fragen:
    - Wurden alle kritischen Erweiterungen erkannt?
    - Sind BasicConstraints richtig gesetzt?
    - Ist die KeyUsage/Extended Key Usage richtig gesetzt?
- Prüfung, ob das Zertifikat gesperrt wurde
- Prüfung der Eignung des Hash-Algorithmus, sofern die Option e_CheckAlgorithms gesetzt wurde
- Prüfung der Eignung des Signatur-Algorithmus, sofern die Option e_CheckAlgorithms gesetzt wurde

```
typedef struct {
        CCES_CheckResult        chainingOK;
        CCES_CheckResult        validityPeriodOK;
        CCES_CheckResult        extensionsOK;
        CCES_KeyUsage           certKeyUsage;
        CCES_CertStatus         certStatus;
        CCES_Time               revocationTime;
        CCES_DataContainer      certStatusEvidence;
        CCES_CheckResult        hashAlgUpToDate;
        CCES_CheckResult        sigAlgUpToDate;
} CCES_CheckSingleCert;
```

Sofern das Zertifikat gesperrt ist, wird in der Variable revocationTime der Sperrzeitpunkt eingetragen. Außerdem kann, sofern die Option e_StoreCertStatusEvidence in CCES_VerifyOptions gesetzt wurde, in der Variable certStatusEvidence die zur Prüfung verwendete Sperrliste oder Antwort des OCSP-Responders in Form eines Datencontainers vom Typ e_OctetStringType abgelegt werden.

Bei der Überprüfung der Key Usage Erweiterung im Zertifikat wird diese auch als CCES_KeyUsage-Element abgelegt.

4.2.1.3.6 CCES_CertStatus

typedef enum

{

e_notChecked,

e_NotRevoked,

e_unspecified,

e_keyCompromise,

e_cACompromise,

e_affiliationChanged,

e_superseded,

e_cessationOfOperation,

e_certificateHold,

e_removeFromCRL,

e_privilegeWithdrawn,

e_aACompromise

} CCES_CertStatus

4.2.1.3.7 CCES_CheckResult

typedef enum

{

e_CheckOk,

e_CheckNotOk,

e_NotChecked

} CCES_CheckResult

4.2.2 Administrative Funktionen

Die administrativen Funktionen lassen sich in folgende Gruppen unterteilen:

- Funktionen zur Instanzen-Verwaltung
- Funktionen zur PSE-Verwaltung
- Sonstige Service-Funktionen

4.2.2.1 Funktionen zur Instanzen-Verwaltung

In diesem Abschnitt sind die Funktionen zur Verwaltung von Instanzen einer CCES Signature Library beschrieben. Durch die Verwendung mehrerer Instanzen kann eine parallele Verarbeitung ermöglicht werden.

4.2.2.1.1 AcquireLibInstance

AcquireLibInstance(pPSEName : CCES_DataContainer, hLibInstance : CCES_InstanceHandle) : CCES_ErrorCode	
Beschreibung	Bevor eine Funktion einer CCES-Signature-Library verwendet werden kann, muss zunächst ein InstanceHandle angefordert werden. Dieses Handle muss anschließend an alle Funktion übergeben werden. Das Handle ermöglich die parallele Verwendung verschiedener Instanzen einer CCES-Signature-Library.
Parameter	*hLibInstance* – In diesen Zeiger wird das Schnittstellen Handle geschrieben. *pPSEName* – Dieser Datencontainer muss den UserFriendlyName eines verfügbaren PSEs enthalten.
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError • CCES_OutOfMemory • CCES_InvalidParam • CCES_InvalidDataContainer • CCES_InvalidDataType • CCES_InvalidPSEHandle

4.2.2.1.2 FreeLibHandle

FreeLibHandle(hLibInstance : CCES_InstanceHandle) : CCES_ErrorCode	
Beschreibung	Nachdem eine Instanz einer CCES-Signature-Library nicht mehr benötigt wird, muss das zugehörige InstanceHandle wieder freigegeben werden. Nach dem Aufruf dieser Funktion ist das InstanceHandle ungültig.
Parameter	hLibInstance – Das freizugebende InstanceHandle
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError

- CCES_InvalidLibHandle

4.2.2.2 Funktionen zur PSE-Verwaltung

In diesem Abschnitt sind die Funktionen zur PSE-Verwaltung beschrieben. Durch diese Funktionen können verfügbare soft- oder hardware-basierte Personal Security Environments verwaltet werden.

4.2.2.2.1 QueryPSEs

QueryPSEs(hQueryPSE : CCES_QueryPSEHandle) : CCES_ErrorCode	
Beschreibung	Diese Funktion initialisiert die Aufzählungsfunktion der verfügbaren PSEs. Mit dem zurückgegebenen QueryPSE-Handle kann anschließend die Funktion getNextPSE wiederholt aufgerufen werden.
Parameter	*hQueryPSE* – In diesen Zeiger schreibt die Funktion das QueryPSE Handle.
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError • CCES_OutOfMemory • CCES_FunctionNotSupported • CCES_FunctionBlockedByLicence

4.2.2.2.2 GetNextPSE

GetNextPSE (hQueryPSE : CCES_QueryPSEHandle, pDataContainer : CCES_DataContainer) : CCES_ErrorCode	
Beschreibung	Die Funktion kann mit einem zuvor erzeugten QueryPSE-Handle aufgerufen werden. Bei jedem Aufruf dieser Funktion wird das nächste verfügbare PSE zurückgegeben. Nach dem Funktionsaufruf enthält der DataContainer den „User Friendly Name“ des PSEs.
Parameter	*hQueryPSE* – gültiges QueryPSE Handle *pDataContainer* – In diesen Container wird der User Friendly Name des PSEs geschrieben. Der Typ des DataContainer ist e_ASCIIType.
Bemerkung	Der Inhalt der DataContainers muss über die FreeData-Container Funktion wieder freigegeben werden.

Rückgabe Fehlercode, wobei folgende möglich sind:

- CCES_NoError
- CCES_OutOfMemory
- CCES_FunctionNotSupported
- CCES_FunctionBlockedByLicence
- CCES_InvalidPSEHandle
- CCES_NoMorePSEs

4.2.2.2.3 ReadCertListFromPSE

ReadCertListFromPSE (hQueryPSE : CCES_QueryPSEHandle, pCertList : CCES_DataContainer) : CCES_ErrorCode

Beschreibung Mit dieser Funktion können alle in einem PSE gespeicherten Zertifikate ermittelt werden.

Parameter *hQueryPSE* – gültiges QueryPSE Handle
pDataContainer – In diesem Container vom Typ e_CertificateListType wird die Liste der im betreffenden PSE gespeicherten Zertifikate zurückgeliefert.

Rückgabe Fehlercode, wobei folgende möglich sind:

- CCES_NoError
- CCES_OutOfMemory
- CCES_FunctionNotSupported
- CCES_FunctionBlockedByLicence
- CCES_InvalidPSEHandle
- CCES_InvalidDataContainer
- CCES_InvalidDataType
- CCES_InvalidPSEHandle
- CCES_PSENotAvailable

4.2.2.2.4 GetCertFromPSE

GetCertFromPSE (hQueryPSE : CCES_QueryPSEHandle, pCertId: CCES_DataContainer, pCertificate : CCES_DataContainer) : CCES_ErrorCode

Beschreibung Mit dieser Funktion kann ein bestimmtes in einem PSE gespeichertes Zertifikat ausgelesen werden.

Parameter	*hQueryPSE* – gültiges QueryPSE Handle *pDataContainer* – In diesem Datencontainer vom Typ e_CertificateListType wird das gewünschte Zertifikat bezeichnet. pDataContainer – In diesem Datencontainer vom Typ e_CertificateType wird das gewünschte Zertifikat zurückgeliefert.
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError • CCES_OutOfMemory • CCES_FunctionNotSupported • CCES_FunctionBlockedByLicence • CCES_InvalidPSEHandle • CCES_InvalidDataContainer • CCES_InvalidDataType • CCES_InvalidPSEHandle • CCES_PSENotAvailable • CCES_CertificateNotFound

4.2.2.2.5 FreeQueryPSEHandle

FreeQueryPSEHandle(hQueryPSE : CCES_QueryPSEHandle) : CCES_ErrorCode

Beschreibung	Nach dem die Auswahl des geeigneten PSEs abgeschlossen ist, muss das QueryPSE Handle wieder freigegeben werden. Nach dem Aufruf dieser Funktion ist das Handle ungültig.
Parameter	hQueryPSE – freizugebendes Handle
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError • CCES_FunctionNotSupported • CCES_FunctionBlockedByLicence • CCES_InvalidPSEHandle

4.2.2.3 Sonstige Service-Funktionen

4.2.2.3.1 GetRandom

GetRandom(	**hLibInstance : CCES_InstanceHandle, pDataContainer : CCES_DataContainer) : CCES_ErrorCode**
Beschreibung	Diese Funktion füllt den gegebenen Buffer mit Zufallszahlen. Sofern eine Chipkarte vorhanden ist, werden diese Zufallszahlen mit dem Zufallszahlengenerator der Karte erzeugt.
Parameter	*hLibInstance* – Das InstanceHandle der verwendeten CCES-Signature-Library Instanz. *pDataContainer* – Der Datencontainer, der mit Zufallszahlen gefüllt werden soll. Der Datencontainer ist vom Typ OctetString. Die Anzahl der angeforderten Zufallsbytes wird innerhalb des Datencontainers, in der iDataBufferSize-Variable, übergeben.
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError • CCES_OutOfMemory • CCES_FunctionNotSupported • CCES_FunctionBlockedByLicence • CCES_InvalidDataContainer • CCES_InvalidDataType • CCES_InvalidLibHandle

4.2.2.3.2 ChangePin

ChangePin(hLibInstance : CCES_InstanceHandle) : CCES_ErrorCode	
Beschreibung	Diese Funktion ruft einen Dialog zur Änderung der PIN auf.
Parameter	*hLibInstance* – Das InstanceHandle der verwendeten CCES-Signature-Library Instanz.
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError • CCES_UserCancel • CCES_OutOfMemory

- CCES_FunctionNotSupported
- CCES_FunctionBlockedByLicence
- CCES_InvalidLibHandle
- CCES_PSEInitFailed
- CCES_PSENotAvailable
- CCES_BadPin
- CCES_ChangePinFailed

4.2.2.3.3 FreeDataContainer

FreeDataContainer(pDataContainer : CCES_DataContainer) : CCES_ErrorCode	
Beschreibung	Diese Methode löscht die Daten innerhalb des gegebenen Datencontainers. Die Ergebnisse der Funktionen werden in der Regel in Datencontainern zurück gegeben. Hierbei werden die Container von der aufrufenden Methode angelegt, der Containerinhalt wird allerdings innerhalb einer CCES-Signature-Library allokiert. Um den Speicher des Containerinhalts wieder freizugeben, muss diese Funktion aufgerufen werden.
Parameter	*pDataContainer* – Datencontainer, dessen Inhalt freigegeben werden soll. Der Container selbst wird nicht freigegeben.
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError • CCES_InvalidDataContainer

4.2.2.3.4 GetErrorMessage

GetErrorMessage(nErrorcode : CCES_ErrorCode, nLanguageID : CCES_DataContainer, pErrorMessage : CCES_DataContainer) : CCES_ErrorCode	
Beschreibung	Diese Funktion liefert die Fehlermeldung die zu dem entsprechenden CCES-Signature-API ErrorCode gehört.
Parameter	*nErrorcode* – Der Fehlercode, zu dem die Meldung ermittelt werden soll. *nLanguageID* – Sprach-ID gemäß [ISO639-1] für die Sprache in der die Fehlermeldung zurückgeliefert werden soll.

Die Sprach-ID wird in Form eines Datencontainers vom Typ e_ASCIIType übergeben.
pErrorMessage – Datencontainer vom Typ e_ASCIIType, in dem die Fehlermeldung zurückgeliefert wird.

Rückgabe Fehlercode, wobei folgende möglich sind:

- CCES_NoError
- CCES_UnknownError
- CCES_UnsupportedLanguage
- CCES_FunctionNotSupported
- CCES_FunctionBlockedByLicence

4.2.2.3.5 GetContentType

GetContentType(pDataContainer : CCES_ DataContainer, nGCTypeOptions : CCES_GetContentTypeOptions, nDataType : CCES_DataType) : CCES_ErrorCode

Beschreibung Diese Funktion ermittelt für einen übergebenen Datencontainer den Datentyp.

Parameter *pDataContainer* – Datencontainer dessen Typ ermittelt werden soll.
nGCTypeOptions – In Abhängigkeit von den übergebenen Optionen wird lediglich der als Teil des Containers übergebene Typ ausgelesen, der vermeintliche Typ des Datencontainers ermittelt oder eine vollständige Überprüfung der Syntax des Containers durchgeführt.
nDataType – der durch die Funktion ermittelte Datentyp des übergebenen Datencontainers.

Rückgabe Fehlercode, wobei folgende möglich sind:

- CCES_NoError
- CCES_UnknownError
- CCES_InvalidDataContainer
- CCES_InvalidDataType
- CCES_FunctionNotSupported
- CCES_FunctionBlockedByLicence

4.2.2.3.6 GetLibraryInfo

GetLibraryInfo(hLibInstance : CCES_InstanceHandle, pLibraryInfo : CCES_LibraryInfo) : CCES_ErrorCode

Beschreibung	Diese Funktion liefert grundsätzliche Informationen zur CCES-Signature-Library zurück.
Parameter	*hLibInstance* – Handle zu einer Instanz der CCES-Signature-Library. *pLibraryInfo* – Informationen zur Bibliothek in Form einer CCES_LibraryInfo Datenstruktur.
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError • CCES_UnknownError • CCES_InvalidLibHandle

4.2.3 Sign

Sign(hLibInstance : CCES_InstanceHandle, pDataContainer : CCES_DataContainer, pExistingSignature : CCES_DataContainer, pAttributeCertificates : CCES_DataContainer, nSignOptions : CCES_SignOptions, pSignatureContainer : CCES_DataContainer) : CCES_ErrorCode

Beschreibung	Diese Funktion erstellt eine Signatur der gegebenen Daten. Falls bereits eine von den Daten getrennte Signatur existiert, so kann diese Signatur mit übergeben werden.
Parameter	*hLibInstance* – Das InstanceHandle der verwendeten CCES-Signature-Library Instanz *pDataContainer* – Der Datencontainer mit den zu signierenden Daten. *pExistingSignature* – Bereits bestehende Signatur für die Erstellung einer parallelen Signatur. NULL wenn nicht benötigt. *pAttributeCertificates* – Attributzertifikate, die mit in die Signatur integriert werden sollen. NULL wenn nicht

benötigt.
nSignOptions – Optionen für Signaturformate
pSignatureContainer – In diesen Container wird das Ergebnis der Funktion abgelegt.

Rückgabe Fehlercode, wobei folgende möglich sind:

- CCES_NoError
- CCES_UserCancel
- CCES_OutOfMemory
- CCES_InvalidParam
- CCES_FunctionNotSupported
- CCES_FunctionBlockedByLicence
- CCES_FeatureNotSupported
- CCES_FeatureBlockedByLicence
- CCES_InvalidDataContainer
- CCES_InvalidDataType
- CCES_InvalidLibHandle
- CCES_PSEInitFailed
- CCES_PSENotAvailable
- CCES_BadPin

In Abhängigkeit vom Datentyp der in pDataContainer übergebenen Daten sind verschiedene Signaturformate und zusätzliche in die Signatur einzubeziehende Informationen (Attributzertifikate oder existierende Signaturen) zulässig oder nicht. Unzulässige nSignOptions führen zur Fehlermeldung CCES_InvalidParam. Unzulässige zusätzliche Informationen werden ignoriert.

Außerdem sei angemerkt, dass die Übergabe von XML-spezifischen Signaturoptionen nicht direkt über die API, sondern im Rahmen der übergebenen XML-Datenstruktur geschieht. Deshalb muss die übergebene XML-Struktur zumindest ein entsprechendes Signature-Feld enthalten, in dessen SignedInfo-Feld die notwendigen Instruktionen enthalten sind.

Input-Typ	e_NoHighLevelFormat	e_CMSEnveloping	e_CMSDetached	e_SMIMEClearSigned	e_SMIMEOpaqueSigned	e_PGP	e_XML	e_PDF	Output-Typ
e_FileType	X	x[7] (A,S)	x (A,S)			x			e_FileType[8]
e_HashType	X	x (A,S)	x (A,S)			x			e_AbstractASN1Type, e_CMSType, bzw. e_PGPType
e_OctetStringType	X	x	x			x			e_AbstractASN1Type,

[7] (A,S) deutet an, dass die Übergabe von Attribut-Zertifikaten und existierenden Signaturen möglich ist.
[8] In Abhängigkeit vom Signaturformat wird im gleichen Verzeichnis eine Datei mit der entsprechenden Endung angelegt (.p7s für CMS, .pgp für PGP und .der, falls kein High-Level-Signatur-Format gewählt wurde und lediglich die Ausgabe des per OID referenzierten Signaturalgorithmus als DER-codierte ASN.1-Daten abgelegt werden).

		(A,S)	(A,S)						e_CMSType, bzw. e_PGPType
e_MIMEType	x	x (A,S)	x (A,S)	x (A,S)	x (A,S)	x			e_AbstractASN1Type, e_CMSType, e_PGPType, bzw. e_SMIMEType
e_CMSType	x	x (A,S)	X (A,S)			x			e_AbstractASN1Type, e_CMSType, bzw. e_PGPType
e_SMIMEType	x	x (A,S)	x (A,S)	x (A,S)	x (A,S)	x			e_AbstractASN1Type, e_CMSType, e_PGPType, bzw. e_SMIMEType
e_PGPType	x	x (A,S)	x (A,S)			x			e_AbstractASN1Type, e_CMSType, bzw. e_PGPType
e_XMLType	x	x (A,S)	x (A,S)			x	x		e_AbstractASN1Type, e_CMSType, e_PGPType, bzw. e_XMLType
e_PDFType	x	x (A,S)	x (A,S)			x		x	e_AbstractASN1Type, e_CMSType, e_PGPType, bzw. e_PDFType
e_AbstractASN1Type	x	x (A,S)	x (A,S)			x			e_AbstractASN1Type, e_CMSType, bzw. e_PGPType

4.2.4 Hash

Hash(	**hLibInstance : CCES_InstanceHandle,** **pDataContainer : CCES_DataContainer,** **nHashOptions : CCES_HashOptions,** **pHashContainer : CCES_DataContainer) : CCES_ErrorCode**

Beschreibung	Diese Funktion erstellt Hashwerte aus den übergebenen Daten.
Parameter	*hLibInstance* – Das InstanceHandle der verwendeten CCES-Signature-Library Instanz *pDataContainer* – Der Datencontainer mit den zu hashenden Daten. *nHashOptions* – Optionen für zu verwendende Hashfunktion und Ausgabeformat. *pHashContainer* – In diesen Container wird das Ergebnis der Funktion abgelegt.
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError • CCES_UserCancel • CCES_OutOfMemory • CCES_InvalidParam • CCES_FunctionNotSupported • CCES_FunctionBlockedByLicence • CCES_FeatureNotSupported • CCES_FeatureBlockedByLicence • CCES_InvalidDataContainer • CCES_InvalidDataType • CCES_InvalidLibHandle

Die zu verwendende Hashfunktion ergibt sich aus dem Object Identifier in den übergebenen nHashOptions.

Der Typ des Funktionsergebnisses ergibt sich aus dem Typ der Eingabedaten und ggf. den nHashOptions:

- e_FileType
  Wird ein e_FileType mit einer Liste von Dateien übergeben, dann hängt der Rückgabewert von den übergebenen nHashOptions ab:
  - e_manyHashes

führt dazu, dass für jede übergebene Datei eine entsprechende Datei mit der Endung .hsh im gleichen Verzeichnis erzeugt wird, die einen zugehörigen Datencontainer vom Typ e_HashType enthält.

  - o e_singleHashtree
    führt dazu, dass ein Element vom Typ e_HashTreeType produziert und in der Datei HashTree.htr abgelegt wird. Dies geschieht, indem die einzelnen Hashwerte konkateniert werden.

- e_OctetStringType
  liefert einen Datencontainer vom Typ e_HashType zurück.

Werden andere Datentypen übergeben wird der Fehler CCES_InvalidDataContainer zurückgeliefert.

4.2.5 CountSignatures

CountSignatures(hLibInstance : CCES_InstanceHandle, pSignatureContainer : CCES_DataContainer, nNumberOfSignatures : CCES_Number) : CCES_ErrorCode

Beschreibung	Diese Funktion ermittelt die Anzahl der in einem Datencontainer vorhandenen Signaturen.
Parameter	*hLibInstance* – Das InstanceHandle der verwendeten CCES-Signature-Library Instanz. *pSignatureContainer* – Der Datencontainer in dem die Signaturen (ggf. inklusive der Nutzdaten) abgelegt sind. *nNumberOfSignatures* – Anzahl der vorhandenen Signaturen
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError • CCES_OutOfMemory • CCES_InvalidParam • CCES_FunctionNotSupported • CCES_FunctionBlockedByLicence • CCES_FeatureNotSupported • CCES_FeatureBlockedByLicence • CCES_InvalidDataContainer • CCES_InvalidDataType • CCES_InvalidLibHandle

Die folgenden Datentypen für Signaturcontainer werden unterstützt:
- e_CMSType
- e_SMIMEType
- e_XMLType
- e_PDFType

Werden andere Datentypen übergeben wird der Fehler CCES_InvalidDataContainer zurückgeliefert.

4.2.6 Verify

Verify(	**hLibInstance : CCES_InstanceHandle,** **pSignatureContainer : CCES_DataContainer,** **nVerifyOptions : CCES_VerifyOptions,** **nSignatureIndex: CCES_Number,** **pContentData : CCES_DataContainer,** **pVerificationResult : CCES_VerificationResultArray,** **nNumberOfResults : CCES_Number) : CCES_ErrorCode**

Beschreibung: Diese Funktion verifziert eine übergebene Signatur.

Parameter:
hLibInstance – Das InstanceHandle der verwendeten CCES-Signature-Library Instanz.
pSignatureContainer – Der Datencontainer in dem die Signaturen (ggf. inklusive der Nutzdaten) abgelegt sind.
nVerifyOptions – Prüfoptionen bzgl. der Zertifikatsprüfung (online, lokal, etc.)
nSignatureIndex – Der Index der zu prüfenden Signatur, bzw. 0 für alle existierenden Signaturen
pContentData – Datencontainer für Nutzdaten bei clear-signed oder detached Signaturen. NULL wenn nicht benötigt.
pVerificationResult – In diesen Container wird das Ergebnis der Signaturprüfung abgelegt.
pNumberOfResults – Gibt an, wie viele Prüfergebnisse zurückgeliefert wurden.

Rückgabe: Fehlercode, wobei folgende möglich sind:
- CCES_NoError
- CCES_OutOfMemory
- CCES_InvalidParam
- CCES_FunctionNotSupported
- CCES_FunctionBlockedByLicence

- CCES_FeatureNotSupported
- CCES_FeatureBlockedByLicence
- CCES_InvalidDataContainer
- CCES_InvalidDataType
- CCES_InvalidLibHandle
- CCES_CertificateChainError
- CCES_NotExistingSignature
- CCES_OCSPConnectError
- CCES_OCSPRespSigVerifyError
- CCES_OCSPDecodeResponseError

Folgende Datencontainer-Typen können übergeben werden:

- e_CMSType
- e_SMIMEType
- e_PGPType
- e_XMLType
- e_PDFType
- e_AbstractASN1Type

Werden andere Datentypen übergeben wird der Fehler CCES_InvalidDataContainer zurückgeliefert.

4.2.7 ValidateCertificate

ValidateCertificate(hLibInstance : CCES_InstanceHandle,
pCertificateContainer : CCES_DataContainer,
nReferenceTime : CCES_Time,
nKeyUsage : CCES_KeyUsage,
nVerifyOptions : CCES_VerifyOptions,
pVerificationResult : CCES_CheckCertificatePath) :
CCES_ErrorCode

Beschreibung	Diese Funktion validiert eine übergebene Zertifikatskette zu einem bestimmten Referenzzeitpunkt.
Parameter	*hLibInstance* – Das InstanceHandle der verwendeten CCES Signature Library Instanz. *pCertificateContainer* – Der Datencontainer in dem das Zertifikat oder die Zertifikatskette abgelegt ist. *nReferenceTime* – Referenzzeitpunkt bzgl. dem das

Zertifikat geprüft werden muss.

nKeyUsage – Vorgesehene Schlüsselnutzung, die mit der im Zertifikat befindlichen KeyUsage Extension konsistent sein muss.

pVerificationResult – In diesen Container wird das Ergebnis der Prüfung abgelegt.

Rückgabe Fehlercode, wobei folgende möglich sind:

- CCES_NoError
- CCES_OutOfMemory
- CCES_InvalidParam
- CCES_FunctionNotSupported
- CCES_FunctionBlockedByLicence
- CCES_FeatureNotSupported
- CCES_FeatureBlockedByLicence
- CCES_InvalidDataContainer
- CCES_InvalidDataType
- CCES_InvalidLibHandle
- CCES_CertificateChainError
- CCES_OCSPConnectError
- CCES_OCSPRespSigVerifyError
- CCES_OCSPDecodeResponseError

Der übergebene Zertifikatscontainer kann vom Typ e_CertificateType oder e_CertificateListType sein. Werden andere Datentypen übergeben, wird der Fehlercode CCES_InvalidDataContainer zurückgeliefert.

In beiden Fällen ist es die Aufgabe der CCES-Signature-Library die zur Verifikation der übergegebenen Zertifikate möglicherweise zusätzlich nötigen Zertifikate bereitzuhalten oder zu ermitteln.

4.2.8 GetTimestamp

GetTimestamp (hLibInstance : CCES_InstanceHandle, pDataContainer : CCES_DataContainer, pURLofTSA : CCES_DataContainer, nTimestampOptions : CCES_TimestampOptions, pTimestampContainer : CCES_DataContainer) : CCES_ErrorCode

Beschreibung	Diese Funktion fordert einen Zeitstempel für die gegebenen Daten an.
Parameter	*hLibInstance* – Das InstanceHandle der verwendeten CCES-Signature-Library Instanz. *pDataContainer* – Enthält die Daten, für die ein Zeitstempel angefordert werden soll. *pURLofTSA* – Enthält die URL des Zeitstempel-Dienstes. *nTimestampOptions* – Optionen zur Zeitstempelanfrage *pTimestampContainer* – Container in dem das Funktionsergebnis zurückgegeben wird.
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError • CCES_InvalidParam • CCES_FunctionNotSupported • CCES_FunctionBlockedByLicence • CCES_FeatureNotSupported • CCES_FeatureBlockedByLicence • CCES_InvalidDataContainer • CCES_InvalidDataType • CCES_InvalidLibHandle • CCES_TSPConnectError

Der Typ des zurückgegebenen Zeitstempel-Tokens hängt vom Typ des übergebenen Datencontainers (und im Fall einer mittels e_FileType übergebenen Liste von Dateien auch von den übergebenen TimestampOptions) ab:

- e_FileType
  Wird ein e_FileType mit einer Liste von Dateien übergeben, dann hängt der Rückgabewert von den übergebenen nTimestampOptions ab:
  - e_TSPs

führt dazu, dass für jede übergebene Datei eine entsprechende Datei mit der Endung .tss im gleichen Verzeichnis erzeugt wird, die den Zeitstempel gemäß [RFC3161] enthält.

  - o e_ATSs
    führt dazu, dass aus den mittels Dateinamen referenzierten Dateien ArchiveTimeStamps gemäß [BrHu04] erzeugt und in Dateien mit der Endung .ats abgelegt werden. In den reducedHashtrees innerhalb der ArchiveTimeStamps, die aus einem binären Hashbaum abgeleitet wurden, sind die zur Rekonstruktion des Pfades zur zeitgestempelten Wurzel notwendigen Hashwerte abgelegt.
  - o e_singleATS
    führt dazu, dass aus den mittels Dateinamen referenzierten Dateien ein einziger ArchiveTimeStamp gemäß [BrHu04] erzeugt wird und in der Datei ArchiveTimeStamp.ats abgelegt wird. Der reducedHashtree besteht in diesem Fall aus der schlichten Konkatenation der Hashwerte der übergebenen Daten.

- e_HashType und e_OctetStringType
  liefern ein Zeitstempel-Token gemäß [RFC3161] als e_OctetstringType zurück.
- e_HashTreeType
  liefert schließlich ein ArchiveTimeStamp-Token gemäß [BrHu04] als e_Octetstring zurück.

Werden andere Datentypen übergeben wird der Fehler CCES_InvalidDataContainer zurückgeliefert.

4.2.9 VerifyTimestamp

VerifyTimestamp (hLibInstance : CCES_InstanceHandle, pDataContainer : CCES_DataContainer, pTimestampContainer : CCES_DataContainer, nVerifyOptions : CCES_VerifyOptions, pVerificationResult : CCES_VerificationResult) : CCES_ErrorCode

Beschreibung

Diese Funktion überprüft die Gültigkeit eines gegebenen Zeitstempels.

Parameter*hLibInstance* – Das InstanceHandle der verwendeten CCES Signature Library Instanz.
pDataContainer – Container mit den zu überprüfenden Daten
pTimestampContainer – zu überprüfender Zeitstempel
nVerifyOptions – Optionen zur Zeitstempelprüfung
pVerificationResult – In diesen Container wird das Ergebnis der Prüfung des Zeitstempels abgelegt.

Rückgabe

Fehlercode, wobei folgende möglich sind:

- CCES_NoError
- CCES_OutOfMemory
- CCES_InvalidParam
- CCES_FunctionNotSupported
- CCES_FunctionBlockedByLicence
- CCES_FeatureNotSupported
- CCES_FeatureBlockedByLicence
- CCES_InvalidDataContainer
- CCES_InvalidDataType
- CCES_InvalidLibHandle
- CCES_CertificateChainError
- CCES_OCSPConnectError
- CCES_OCSPRespSigVerifyError
- CCES_OCSPDecodeResponseError

4.2.10 CreateEvidenceRecord

CreateEvidenceRecord (hLibInstance : CCES_InstanceHandle, pDataContainer : CCES_DataContainer, pEncryptionMethod : CCES_DataContainer, pCryptoInfo : CCES_DataContainer, pERContainer : CCES_DataContainer) : CCES_ErrorCode	
Beschreibung	Diese Funktion erzeugt aus den übergebenen Daten einen EvidenceRecord gemäß [BrHu04].
Parameter	*hLibInstance* – Das InstanceHandle der verwendeten CCES-Signature-Library Instanz. *pDataContainer* – Enthält die Daten, aus denen ein EvidenceRecord erzeugt werden soll. pEncryptionMethod – ist vom Typ e_OctetString und enthält die EncryptionMethod gemäß [BrHu04] oder NULL, falls nicht benötigt. *pCryptoInfo* – ist vom Typ e_OctetString und enthält die CryptoInfos gemäß [BrHu04] oder NULL, falls nicht benötigt. *pERContainer* – Container in dem der erzeugte EvidenceRecord zurückgegeben wird.
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError • CCES_InvalidParam • CCES_FunctionNotSupported • CCES_FunctionBlockedByLicence • CCES_FeatureNotSupported • CCES_FeatureBlockedByLicence • CCES_InvalidDataContainer • CCES_InvalidDataType • CCES_InvalidLibHandle • CCES_TSPConnectError

Die Daten aus denen ein EvidenceRecord erzeugt werden soll, können in folgender Form übergeben werden:

- e_FileType
  Wird ein e_FileType mit einer Liste von Dateien übergeben, so wird der Typ der Daten anhand der Dateiendung unterschieden.

  - o .p7c, pgp oder .smime bezeichnet verschlüsselte Daten vom Typ e_CMSType, e_PGPType bzw. e_SMIMEType, wobei die Art der Verschlüsselung mit der übergebenen EncryptionMethod übereinstimmen muss. Ist dies nicht der Fall wird die Fehlermeldung CCES_InvalidParam zurückgeliefert.
  - o .hsh bezeichnet einen Datencontainer vom Typ e_HashType
  - o .htr bezeichnet einen Datencontainer vom Typ e_HashTree
  - o jede andere Endung bedeutet, dass die Datei im Klartext vorliegt
- e_HashTreeType bedeutet, dass im Datencontainer ein reduzierter Hashbaum übergeben wird.

Werden andere Datentypen übergeben wird der Fehler CCES_InvalidDataContainer zurückgeliefert.

4.2.11 RenewEvidenceRecord

RenewEvidenceRecord (hLibInstance : CCES_InstanceHandle, pOldEvidenceRecord : CCES_DataContainer, pRenewalOptions: CCES_ERRenewalOptions, pDataContainer : CCES_DataContainer, pNewEvidenceRecord : CCES_DataContainer) : CCES_ErrorCode	
Beschreibung	Diese Funktion erneuert einen übergebenen Evidence Record gemäß [BrHu04].
Parameter	*hLibInstance* – Das InstanceHandle der verwendeten CCES-Signature-Library Instanz. *pOldEvidenceRecord* – enthält den zu erneuernden Evidence Record. pRenewalOptions – enthält die Information, ob lediglich ein TimeStampRenewal oder ein kompletter HashTreeRenewal durchgeführt werden soll. *pDataContainer* – Enthält die Daten, aus denen im Fall eines HashTreeRenewal ein neuer Hashbaum zur Erneuerung des EvidenceRecords erzeugt werden soll oder NULL, falls nicht benötigt. *pNewEvidenceRecord* – Container in dem der erneuerte EvidenceRecord zurückgegeben wird.
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError • CCES_InvalidParam • CCES_FunctionNotSupported • CCES_FunctionBlockedByLicence

- CCES_FeatureNotSupported
- CCES_FeatureBlockedByLicence
- CCES_InvalidDataContainer
- CCES_InvalidDataType
- CCES_InvalidLibHandle
- CCES_TSPConnectError

Sofern die Option e_HashTreeRenewal gewählt wurde, können die Daten, aus denen der Hashbaum zur Erneuerung des EvidenceRecords erzeugt werden soll, wie bei der Funktion CreateEvidenceRecord übergeben werden.

4.2.12 VerifyEvidenceRecord

VerifyEvidenceRecord (hLibInstance : CCES_InstanceHandle, pDataContainer : CCES_DataContainer, pEvidenceRecord : CCES_DataContainer, nVerifyOptions : CCES_VerifyOptions, pVerificationResult : CCES_VerificationResult) : CCES_ErrorCode

Beschreibung	Diese Funktion überprüft die Gültigkeit eines gegebenen EvidenceRecords.
Parameter	*hLibInstance* – Das InstanceHandle der verwendeten CCES-Signature-Library Instanz. *pDataContainer* – Container mit den zu überprüfenden Daten. *pEvidenceRecord* – zu überprüfender Evidence Record *nVerifyOptions* – Optionen zur Überprüfung des Evidence-Records *pVerificationResult* – In diesen Container wird das Ergebnis der Prüfung des Evidence Records abgelegt.
Rückgabe	Fehlercode, wobei folgende möglich sind:

- CCES_NoError
- CCES_OutOfMemory
- CCES_InvalidParam
- CCES_FunctionNotSupported
- CCES_FunctionBlockedByLicence
- CCES_FeatureNotSupported
- CCES_FeatureBlockedByLicence

- CCES_InvalidDataContainer
- CCES_InvalidDataType
- CCES_InvalidLibHandle
- CCES_CertificateChainError
- CCES_OCSPConnectError
- CCES_OCSPRespSigVerifyError
- CCES_OCSPDecodeResponseError

Sofern in pDataContainer Daten übergeben wurden, erfolgt bei der Überprüfung des Evidence Records die erneute Berechnung des kompletten Hashbaumes. In diesem Fall erfolgt die Datenübergabe wie bei der erstmaligen Erzeugung eines EvidenceRecords mit der Funktion `CreateEvidenceRecord`.

4.2.13 Encrypt

Encrypt(	**hLibInstance : CCES_InstanceHandle, pDataContainer : CCES_DataContainer, pRecipients : CCES_DataContainer, nVerifyOptions : CCES_VerifyOptions, nEncryptOption : CCES_EncryptOptions, pEncryptedContainer : CCES_DataContainer) : CCES_ErrorCode**
Beschreibung	Diese Funktion verschlüsselt die gegebenen Daten mit den öffentlichen Schlüsseln der angegebenen Empfänger. Die zugehörigen Zertifikate müssen in der lokalen Datenbank der CCES-Signature-Library zur Verfügung stehen.
Parameter	*hLibInstance* – Das InstanceHandle der verwendeten CCES Signature Library Instanz. *pDataContainer* – Der Datencontainer mit den zu verschlüsselnden Daten. *pRecipients* – Eine Liste von Empfängerzertifikaten, mit deren öffentlichen Schlüsseln die Daten verschlüsselt werden sollen. *nOptions* – Prüfung der Verschlüsselungszertifikate (online/lokal) *pEncryptedContainer* – In diesen Container wird der Ergebnis der Funktion abgelegt.
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError • CCES_OutOfMemory

- CCES_InvalidParam
- CCES_FunctionNotSupported
- CCES_FunctionBlockedByLicence
- CCES_FeatureNotSupported
- CCES_FeatureBlockedByLicence
- CCES_InvalidDataContainer
- CCES_InvalidDataType
- CCES_InvalidLibHandle
- CCES_CertificateChainError
- CCES_OCSPConnectError
- CCES_OCSPRespSigVerifyError
- CCES_OCSPDecodeResponseError
- CCES_InvalidCertificate

Der Typ des zurückgegebenen Datencontainers hängt vom Typ des übergebenen Datencontainers und von den übergebenen CCES_EncryptOptions ab:

- e_FileType
  Wird ein e_FileType mit einer Liste von Dateien übergeben, dann wird im gleichen Verzeichnis eine verschlüsselte Datei mit einer, von den CCES_EncryptOptions abhängigen, formatspezifischen (zusätzlichen) Endung erzeugt.
  - e_CMS führt zu .pc7
  - e_SMIME führt zu .smime
  - e_PGP führt zu .pgp
- e_MIMEType liefert einen e_SMIMEType zurück
- e_OctetStringType
  In diesem Fall ist der Rückgabewert wieder von den übergebenen CCES_EncryptOptions abhängig:
  - e_CMS liefert einen e_CMSType-Container
  - e_PGP liefert einen e_PGPType-Container

Werden andere Datentypen übergeben wird der Fehler CCES_InvalidDataContainer zurückgeliefert.

4.2.14 Decrypt

Decrypt(	**hLibInstance : CCES_InstanceHandle, pEncryptedContainer : CCES_DataContainer, pDecryptedContainer : CCES_DataContainer) : CCES_ErrorCode**
Beschreibung	Diese Funktion entschlüsselt gemäß CMS, S/MIME oder PGP verschlüsselte Datencontainer.
Parameter	*hLibInstance* – Das InstanceHandle der verwendeten CCES-Signature-Library Instanz. *pEncryptedContainer* – Der Datencontainer mit den verschlüsselten Daten. *pDecryptedContainer* – In diesen Container wird das Ergebnis der Entschlüsselungsfunktion abgelegt.
Rückgabe	Fehlercode, wobei folgende möglich sind: • CCES_NoError • CCES_UserCancel • CCES_OutOfMemory • CCES_InvalidParam • CCES_FunctionNotSupported • CCES_FunctionBlockedByLicence • CCES_FeatureNotSupported • CCES_FeatureBlockedByLicence • CCES_InvalidDataContainer • CCES_InvalidDataType • CCES_InvalidLibHandle • CCES_PSEInitFailed • CCES_PSENotAvailable • CCES_CannotDecryptData • CCES_BadPin

Der Typ des zurückgegebenen Datencontainers hängt vom Typ des übergebenen Datencontainers ab.

- e_FileType
  Wird ein e_FileType mit einer Liste von verschlüsselten Dateien übergeben, dann wird im gleichen Verzeichnis die jeweils unverschlüsselte Datei abgelegt.

Zulässige Dateiendungen sind .pc7, .smime und e_PGP, wobei in den Dateien Datencontainer vom Typ e_CMSType, e_SMIMETyp bzw. e_PGPType erwartet werden. Ist dies nicht der Fall, so wird der Fehler CCES_InvalidDataContainer zurückgegeben.

- e_SMIMEType liefert einen e_MIMEType zurück
- e_CMSType und e_PGPType liefern einen e_OctetstringType zurück.

Werden andere Datentypen übergeben wird der Fehler CCES_InvalidDataContainer zurückgeliefert.

Referenzen

[AI-API-99] Adobe Inc.: *Acrobat Digital Signature API*, Adobe Developer Technologies – Technical Note #5192, revised November 10th, 1999, via http://partners.adobe.com/asn/acrobat/docs/digsig.pdf

[AI-API-03] Adobe Inc.: *Acrobat Digital Signature API*, Adobe Developer Technologies – Technical Note #5192, Version: Acrobat 6.0, May 2003, via http://partners.adobe.com/asn/acrobat/sdk/reg/Documentation/Extended_API_For_Plugins/DigitalSignatureAPIRef.pdf

[AI-PDF] Adobe Inc.: *Portable Document Format Reference Manual*, Version 1.3-1.5, via http://partners.adobe.com/asn/tech/pdf/specifications.jsp

[AI-TIFF] Adobe Developer Association: *TIFF Specification*, Revision 6.0, June 3rd, 1992, via http://partners.adobe.com/asn/developer/pdfs/tn/TIFF6.pdf

[BaSch00] Bartosch, M., Schneider, J.: *Nutzen und Grenzen von Kryptographie-Standards und ihrer APIs*, Tagungsband „Systemsicherheit“, DuD Fachbeiträge, Vieweg, 2000, SS. 206-226

[BGPT03] Brandner, R.; Gondrom, T., Pordesch, U.; Tielemann, M.: *Archive Time-Stamps Syntax (ATS),* Internet-Draft, July 2003 via http://ltans.edelweb.fr/draft-brandner-etal-ats-00.txt

[BrHu04] Brandner, R.; Hunter, B.: *Evidence Record Syntax*, Internet-Draft, February 2004, via http://ltans.edelweb.fr/draft-ietf-ltans-ers-00.txt

[CT-API] Attrott, J.; Eckstein, L.; Kowalski, B.; Moos, R.; Reimer, H.; Struif, B.: *CT-API - Anwendungsunabhängiges CardTerminal-Application Programming Interface für Chipkartenanwendungen*, Version 1.1.1, via https://www.secure.trusted-site.de/Download/CTAPI/CTAPI111.pdf

[HPC-Spec] Struif, B.: *German Health Professional Card and Security Module Card, Specification – Pharmacist & Physician* – Version 2.0 vom 31.7.2003, via

[ISIS-MTT] TeleTrusT e.V.: *ISIS-MTT-Spezifikation*, Version 1.1, März 2004, via http://www.isis-mtt.de/

[ISO639-1] ISO 639-1:2002 -- *Codes for the representation of names of languages -- Part 1: Alpha-2 code*

[ISO8824-2] ISO 8824-2:1998 – Information Technology – *Abstract Syntax Notation One (ASN.1): Information Object Specication*, International Standard ITU-T Rec. X.681 (1997)

[JCA] Sun Inc.: *Java Cryptographic Architecture (JCA)*, via http://www.javasoft.com/products/jdk/1.2/docs/guide/security/CryptoSpec.hml

[JCE] Sun Inc.: *Java Cryptography Extension (JCE)*, via http://www.javasoft.com/products/jce

[JXS-API] Sun Inc.: *XML Digital Signature APIs*, JSR-000105, via http://jcp.org/aboutJava/communityprocess/review/jsr105/index.html

[MS-CAPI] Microsoft Inc.: *Cryptography Reference (Microsoft CryptoAPI), Platform SDK: Security*, via http://msdn.microsoft.com/library/en-us/security/security/cryptography_reference.asp

[OCF] Open Card Consortium, *OpenCard Framework V 1.2*, via http://www.opencard.org/docs/1.2/index.html

[PC/SC] PC/SC Workgroup, *PC/SC Workgroup Specifications 1.0*, via http://pcscworkgroup.com

[PKCS#7] RSA Labs: *PKCS #7 - Cryptographic Message Syntax Standard*, via http://www.rsalabs.com/pkcs/pkcs-ia7/index.html (siehe auch [RFC2315])

[PKCS#11] RSA Labs: *PKCS#11: Cryptographic Token Interface Standard*, Version 2.11, via http://www.rsasecurity.com/rsalabs/pkcs/pkcs-11/index.html

[RFC1421] Linn, J.: *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures*, RFC 1421, 1993, via http://www.ietf.org

[RFC2045] Freed, N.; Borenstein, N.: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, RFC2045, 1996, via http://www.ietf.org

[RFC2078] Linn, J.: *Generic Security Services Application Programming Interface*, Version 2 (GSS-API v2), RFC2078, 1997, via http://www.ietf.org

[RFC2315] Kaliski, B.: *PKCS #7: Cryptographic Message Syntax*, Version 1.5, RFC 2315, 1998, via http://www.ietf.org

[RFC2440] Callas, J.; Donnerhacke, L.; Finney, H.; Thayer, R.: *OpenPGP Message Format*, RFC 2440, 1998, via http://www.ietf.org

[RFC2479] Adams, C.: *Independent Data Unit Protection – Generic Security Service Application Program Interface (IDUP-GSS-API) v2*, RFC 2479, 1998, via http://www.ietf.org

[RFC2628] Smyslov, V.: *Simple Cryptographic Program Interface (Crypto API), RFC 2628*, via http://www.ietf.org

[RFC2630] Housley, R.: *Cryptographic Message Syntax (CMS)*, RFC 2630, via http://www.ietf.org

[RFC2633] Ramsdell, B.: *S/MIME Version 3 Message Specification*, RFC 2633, via http://www.ietf.org

[RFC2744] Wray, J.: *Generic Security Service API Version 2 : C-bindings*, RFC2744, 2000, via http://www.ietf.org

[RFC2853] J. Kabat, J.; Upadhyay, M.: *Generic Security Service API Version 2 : Java Bindings*, 2000, via http://www.ietf.org

[RFC3156] Elkins, M., Del Torto, D., Levien, R., Roessler, T.: *MIME Security with OpenPGP*, 2001, via http://www.ietf.org

[RFC3161] Adams, C.; Cain, P.; Pinkas, D.: *Internet X.509 Public Key Infrastructure – Time Stamp Protocol (TSP)*, RFC 3161, via http://www.ietf.org

[RFC3275] D. Eastlake, D.; Reagle, J.; Solo, D.: *(Extensible Markup Language) XML-Signature Syntax and Processing*, RFC 3275, via http://www.ietf.org

[RFC3280] R. Housley, W. Polk, W. Ford, D. Solo: *Internet X.509 Public Key Infrastructure - Certificate and Certificate Revocation List (CRL) Profile*, RFC3280, via http://www.ietf.org

[RFC3369] Housley, R.: *Cryptographic Message Syntax (CMS)*, RFC 3369, via http://www.ietf.org

[RFC3447] J. Jonsson, J.; Kaliski, B.: *PKCS #1 – RSA Cryptography Specifications Version 2.1*, 2003, via http://www.ietf.org

[SAP-SSF] SAP AG: *Secure Store and Forward (SSF) API Specification*, via http://www.sap.com/partners/icc/scenarios/pdf/bc_ssf_api.pdf

[SigAll-API] Signaturbündnis / SRC GmbH: *SigAll-API – Specification of the Application Programming Interface to the Signature Card*, Version 0.9 vom 15.10.2004

[TTT-GOIC] TeleTrusT e.V.: German Office Identity Card – Elektronischer Dienstausweis, Version 1.0, 06.07.2000, via http://www.teletrust.de/dokumente/oic_1-0.pdf

[US-GSC-IS] NIST: *Government Smart Card Interoperability Specification*, Version 2.1., via http://csrc.nist.gov/publications/nistir/nistir-6887.pdf

[XML] T. Bray, E. Maler, J. Paoli, C. M. Sperberg-McQueen: *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Recommendation, 2000, via http://www.w3.org/TR/2000/REC-xml-20001006

[X/O-GCS] X/Open: *Generic Cryptographic Service API (GCS-API) Base*, Preliminary Specification, June 1996, ISBN: 1-85912-195-0, via http://www.opengroup.org/publications/catalog/p442.htm

[X/O-CDSA] X/Open: Common Security: *CDSA and CSSM*, Version 2.3, Technical Standard, May 2000, ISBN: 1-85912-202-7, via http://www.opengroup.org/publications/catalog/c914.htm