

Elektronische Gesundheitskarte und Telematikinfrastruktur

Feature: Hash based Certificate Status List (HCSL)

Version: 1.0.0_CC
Revision: 1710549
Stand: 01.09.2026
Status: zur Abstimmung
freigegeben
Klassifizierung: öffentlich_Entwurf
Referenzierung: gemF_HCSL

Dokumenteninformation

Gender-Hinweis

Aus Gründen der besseren Lesbarkeit wird in diesem Dokument überwiegend die männliche Form verwendet. Sämtliche Personenbezeichnungen gelten gleichermaßen für alle Geschlechter.

Änderungen zur Vorversion

Anpassungen des vorliegenden Dokumentes im Vergleich zur Vorversion können Sie der nachfolgenden Tabelle entnehmen.

Dokumentenhistorie

Version	Datum	Grund der Änderung, besondere Hinweise	Bearbeitung
0.5.0	01.06.2026	interne Abstimmungen	gematik
1.0.0_CC	01.09.2026	zur Abstimmung freigegeben	gematik

Inhaltsverzeichnis

1 Einordnung des Dokuments.....	5
1.1 Zielsetzung.....	6
1.2 Zielgruppe.....	6
2 Epic und User Story.....	7
2.1 Epic Hashbasierte Certificate Status Listen in der TI.....	7
2.1.1 User Stories.....	7
3 Einordnung in die Telematikinfrastruktur.....	8
4 Technisches Konzept.....	9
4.1 HCSL, Positivlisten (HPL) und Negativlisten (HNL).....	10
4.1.1 Aufbau einer HPL.....	11
4.1.2 Aufbau einer HNL.....	12
4.1.3 Die zwei Signaturen einer HCSL.....	13
4.2 Unterschiedliche Paradigmen: OCSP und HCSL.....	14
4.3 Nutzung OCSP vs. HCSL.....	15
4.4 Namensschema für die Listen.....	15
4.5 Delta/Update-Dateien.....	16
4.6 Bereitstellung durch den TSP.....	16
4.7 Nutzung einer Liste im Fachdienst als HCSL-Client.....	18
4.8 Nutzung einer Liste in beschränkten Laufzeitumgebungen (HSM).....	19
4.9 nonQES vs. QES.....	22
4.10 Post-Quanten-Kryptographie.....	22
4.11 Zusammenfassung Aufbau HCSL und Delta-Dateien.....	22
5 Spezifikation.....	25
5.1 Anforderungen an den TSP.....	25
5.1.1 HCSL: Header.....	27
5.1.2 HPL: Body.....	27
5.1.3 HNL: Body.....	28
5.1.4 HPL: Trailer (zwei Signaturen).....	29
5.1.5 HNL: Trailer (zwei Signaturen).....	30
5.1.6 Delta/Update-Dateien.....	31
5.1.7 Adaptive Fixierung der HLen.....	32
5.2 HCSL-Client.....	33
5.3 HCSL-Client in beschränkter Laufzeitumgebungen (HSM).....	35
5.4 Sicherheit.....	36
5.5 Betrieb.....	36

79	6 Merkle-Hashbäume (informativ).....	37
80	7 Beispielimplementierung.....	38
81	8 Anhang A - Verzeichnisse.....	39
82	8.1 Abkürzungen.....	39
83	8.2 Abbildungsverzeichnis.....	39
84	8.3 Tabellenverzeichnis.....	40
85	8.4 Referenzierte Dokumente.....	40
86	8.4.1 Dokumente der gematik.....	40
87	8.4.2 Weitere Dokumente.....	40
88		
89		

1 Einordnung des Dokuments

Dieses Dokument definiert Hash-basierte Zertifikatsstatuslisten (HCSL). Mithilfe dieser Listen liefert ein TSP (genauer dessen OCSP-Responder) Informationen über den Sperrstatus aller vom TSP ausgestellten Zertifikate, die noch zeitlich gültig sind. Die Listen besitzen eine spezielle Struktur, die die Auswertung und das Update dieser Listen effizient ermöglicht. Weiterhin besitzen diese Listen eine besondere kryptographische Eigenschaft, die die Auswertung in beschränkten Laufzeitumgebungen (bspw. ePA-VAU-HSM) möglich macht. Traditionelle Zertifikatssperrlisten (CRL) besitzen diese Leistungsmerkmale nicht.

Die Verfügbarkeit von Sperrinformationen bzw. gerade die Information über die Nichtgesperrtheit von Zertifikaten ist absolut kritisch für die Verfügbarkeit von quasi allen Anwendungen der TI.

Wie Erfahrungen in der PU zeigen, führen Störungen bei den OCSP-Respondern innerhalb weniger Minuten zu massiven Störungen bei allen Anwendungen der TI.

Einige Anwendungen der TI (ePA, E-Rezept, KIM) müssen fast alle existierenden SMC-B-Zertifikate quasi ständig verarbeiten. Dafür ist das traditionell verwendete OCSP-Protokoll nicht konzipiert. Es ist möglich mittels der Hash-basierten Zertifikatsstatuslisten eine Leistungssteigerung der Sperrstatusabfrage mit einem Faktor größer 960 zu erzielen.

Ein noch weitaus wichtigerer Aspekt ist jedoch, dass durch die Verwendung der HCSL die Last an den OCSP-Respondern deterministisch wird (alle Fachdienste laden genau alle 10 Minuten eine neue HCSL bzw. Delta-Informationen vom HCSL-Downloadpunkt am OCSP-Responder). Damit gefährden Hochlastphasen bei einer Anwendung der TI nicht die Verfügbarkeit anderer Anwendungen der TI.

Ein Fachdienst (bspw. ein ePA-Aktensystem) hat mittels der HCSL die Zertifikatsstatusinformationen aller Zertifikate eines TSP lokal verfügbar. Im Fall der Anfrage eines Primärsystem muss für die Ermittlung des Status des SMC-B-AUT-Zertifikats keine OCSP-Anfrage durchgeführt werden, sondern die notwendigen Informationen liegen schon lokal im Fachdienst vor.

Aufgrund der speziellen Struktur der HCSL ist es möglich

1. effizient in diesen Listen zu suchen,
2. effizient nur Änderungen zwischen zwei Listen zu erstellen und zu übertragen.
3. Statusinformationen in beschränkten Laufzeitumgebungen kryptographisch auswerten zu können (Merkle tree authentication path) oder Objekten (bspw. JWT) Statusinformationen beifügen zu können.

Diese Leistungen sind mit klassischen CRLs nicht erreichbar. Kaskadierte Bloomfilter [CRLITE] erreichen Punkt 3 nicht und sind schwächer bei Punkt 2.

Die OCSP-Responder der TSP besitzen eine HTTP-Schnittstelle. Diese schon existierende HTTP-Schnittstelle wird verwendet, um zusätzlich HCSL als statische Dateien anzubieten. D. h. ein Teil des OCSP-Responders (HTTP-Server) erhält eine zusätzliche Funktionalität ohne die bestehende Funktionalität zu verändern.

Auch in der TI-2.0 wird es weiterhin zertifikatsbasierte Operationen (Authentisierung, Verschlüsselung) u. a. mittels SMC-B oder HSM-B geben. Für diese Zertifikate müssen auch weiterhin performant und verlässlich Statusinformationen für die Anwendungen der TI verfügbar sein.

134 HCSL dient als performante, stabilitätsverbessernde Ergänzung zu OCSP. Die OCSP-
135 Funktionalität bei den TSP der TI bleibt weiterhin bestehen.

136 **1.1 Zielsetzung**

137 Durch die HCSL wird die Verfügbarkeit und Performanz der Anwendungen der TI
138 gesteigert. Die Motivation für die Verwendung von HCSL kommt aus in der PU mehrfach
139 aufgetretenen Problemsituationen und den dabei gesammelten Erkenntnissen.

140 Im Folgenden sind bei Verwendung des Begriffs SMC-B immer auch alle Ausprägungen
141 der SM-B [gemKPT_PKI_TIP#2.7.3.3 Herausgeber der SMC-B] mit eingeschlossen.

142 **1.2 Zielgruppe**

143 Das Dokument definiert die HCSL und richtet sich an TSPs und HCSL-Clients (ePA-
144 Aktensysteme, E-Rezept-Fachdienst, zentrale IDP etc.).

2 Epic und User Story

2.1 Epic Hashbasierte Certificate Status Listen in der TI

Als Leistungserbringer möchte ich performant und zuverlässig (insbesondere verfügbar) einen Fachdienst der TI (ePA, E-Rezept, KIM etc.) nutzen.

2.1.1 User Stories

- Als Patient möchte ich zuverlässig in einer Apotheke ein E-Rezept einlösen können, unabhängig davon, ob es gerade eine hohe Last bei den OCSP-Responder der TI gibt oder dort sogar eine temporäre Störung.
- Als Arzt möchte ich zuverlässig KIM-Nachrichten versenden und erhalten können.
- Als TSP möchte ich die Last auf meine OCSP-Responder reduzieren, indem bestimmte Fachdienste der TI, die im Normalfall sehr viele OCSP-Antworten benötigen, nun HCSL verwenden.
- Als Fachdienst möchte ich die Performanz der Use-Cases und die die Resilienz gegenüber temporärer Nichtverfügbarkeit der Sperrinformationen bei den TSP verbessern.

Kommentar im Rahmen der Kommentierung:

Bei dieser sehr technisch orientierten Spec hier, passt Epic/User Story evtl. gar nicht (ganz weglassen/löschen?). Durch die HCSL kann man am Ende ja nichts grundsätzlich mehr oder weniger machen. Es wird nur verfügbarer und performanter.

3 Einordnung in die Telematikinfrastruktur

Die TSP der TI erzeugen End-Entity-Zertifikate (EE-Zertifikate) für die Chipkarten der TI und andere kryptographische Ausführungsumgebungen (HSM-B etc.). Diese Zertifikate (und die damit verbundenen privaten Schlüssel) bilden die Grundlage fast aller Anwendungsfälle in der TI. Der Status dieser Zertifikate muss performant und verlässlich in den Fachdiensten zur Verfügung stehen, ansonsten können quasi alle Anwendungsfälle nicht mehr durchgeführt werden.

Ein TSP hat für die Kartenherausgabe und die damit verbundene Zertifikatserzeugung typischer Weise mehrere CA. Diese CAs werden technisch durch CA-Zertifikate repräsentiert. Jedes EE-Zertifikat lässt sich bei einer Zertifikatsprüfung auf eine dieser CAs zurückführen. Ein TSP stellt Sperrinformationen für die EE-Zertifikate aller seiner CAs über mehrere HTTP-Schnittstellen zur Verfügung.

Über diese HTTP-Schnittstellen fordern Clients (bspw. ein ePA-Aktensystem) Sperrinformationen i. d. R. für genau ein EE-Zertifikat an. Dafür verwenden die Clients das OCSP-Protokoll, was vor ca. 30 Jahren entwickelt wurde für interaktive Sperrinformationsabfragen von wenigen Zertifikaten. D. h. der TSP muss jede OCSP-Anfrage einzeln bearbeiten und per signierter OCSP-Response beantworten. Dass Clients in kurzer Zeit Sperrinformationen über quasi alle EE-Zertifikate einer CA benötigen, war bei Design von OCSP nicht vorgesehen und führt jetzt zu Problemen, die mehrmalige und teilweise lang anhaltende Ausfälle von Anwendungen in der TI verursacht haben.

Aus den Erfahrungen mit diesen Ausfällen und den Erfahrungen aus 30 Jahren Evolution von Mechanismen zur Verteilung von Sperrinformationen im PKI-Bereich entstanden die HCSL. Die HCSL werden an den HTTP-Schnittstellen der TSP zusätzlich zum OCSP-Protokoll angeboten.

Die HCSL erlauben einem Client effizient Statusinformationen über alle Zertifikate einer CA auf einmal zu erhalten. Die HCSL sind so konstruiert, dass sie sich über Delta-Dateien bei einem Client effizient aktualisieren lassen.

Die HCSL werden unabhängig von Client-Anfragen immer alle 10 Minuten von einer CA erzeugt und stehen ab dann Clients zur Verfügung gestellt. Eine CA muss also im Moment der Client-Anfrage keine Berechnungen mehr durchführen, sondern liefert vorab erzeugte HCSL als statische Dateien aus. Das ist ein Paradigmen-Wechsel zur OCSP-Technologie, bei der eine Antwort immer auf Anfrage eines Client von einer CA erzeugt (und vom einem HSM signiert) werden muss, was negative Auswirkungen auf die Performanz in Last-Szenarien hat.

4 Technisches Konzept

Erfahrungen in der Produktivumgebung der TI haben deutlich gemacht wie kritisch die Verfügbarkeit von Sperrinformationen für kryptographische Schlüssel sind. Sind diese Informationen nicht verfügbar können Anmeldevorgänge (Login), Signaturprüfungen oder der Mailversand (Verschlüsselung) nicht durchgeführt werden -- Anwendungsfälle können nicht durchgeführt werden, die Anwendungen bleiben aus Nutzersicht stehen.

Lastspitzen bei einer Anwendung, die zu vermehrten Abfragen an die OCSP-Responder der TSP führen, beeinflussen ebenfalls andere Anwendungen der TI negativ. Die genauere Analyse der aufgetretenen Problemfälle hat gezeigt, dass das OCSP-Protokoll nur unzureichend für Anwendungen wie ePA, das E-Rezept und teilweise auch bei KIM geeignet ist, bei denen ein Fachdienst ständig von der Mehrzahl der existierenden SMC-B-Zertifikate den aktuellen Sperrstatus kennen muss.

Das folgend definierte Verfahren der hashbasierten Zertifikatsstatuslisten (HCSL)

1. erbringt durch eine effizientere Kodierung eine Leistungssteigerung mit einem Faktor von mehr als 960.
2. führt zu einem deterministischen Abfragemuster bei den TSPs, d. h., egal welche Last aktuell bei einer Anwendung vorhanden ist, die Abfragefrequenz und damit auch die zu übermittelnde Datenmenge am TSP (Last am TSP) bleibt konstant.
3. ermöglicht die Extraktion von Einzelinformationen eines SMC-B-Zertifikat aus einer Liste aller SMC-B-Zertifikate im Fachdienst, wobei das Extrakt wieder separat prüfbar ist (bspw. von einer VAU oder einem HSM) ohne die gesamte Liste kennen zu müssen.
4. führt zu einer Verbesserung beim Datenschutzes (Privacy by Design), weil ein TSP nicht mehr erfährt wann über welches EE-Zertifikat Sperrinformationen von welchem Fachdienst abgerufen werden (bspw. wann welche Arztpraxis eine ePA öffnen möchte).

Klassische CRL [RFC-5280] sind deutlich schlechter bei Punkt 1, weil Sie Informationen wie Sperrgründe und Sperrzeiten notwendiger Weise mit übertragen und eine ineffizientere Kodierung verwenden. Anwendungen der TI verwenden diese Informationen nicht. Auch ermöglichen klassische CRLs Punkt 3 nicht, d. h., ein Fachdienst kann aus einer Sperrliste kein einzeln kryptographisch prüfbares Extrakt für ein bestimmtes Zertifikat erzeugen. Solch ein Extrakt (nur wenige 100 Bytes groß) kann dann in einer VAU oder einem HSM kryptographisch sicher geprüft werden, ohne die gesamte Liste kennen zu müssen. Ein weiterer Vorteil ist, dass man solch ein Extrakt, da es nur wenige Bytes groß ist, auch zu prüfenden Objekten (bspw. JWT) beifügen kann.

Je nach Sperrquote bei den SMC-B-Zertifikaten benötigt man in den Positiv-Listen für alle SMC-B-Zertifikate der TI in Summe ca. 5,5 MiB. Und für die Negativlisten in Summe ca. 0,3 MiB. Bei der Verwendung von OCSP benötigt man dafür mehr als 4800 MiB.

Bei einem TSP-X.509-nonQES-SMC-B und -HBA stehen in dessen OCSP-Responder die SHA-256-Hashwerte aller von ihm ausgestellten Zertifikate bereit, die aktuell zeitlich noch gültig sind (vgl. [gemSpec_PKI#GS-A_4693-*]). Mit diesen Informationen erzeugt der OCSP-Responder zwei Listen: eine hashbasierte Positivliste (HCSL/HPL) und eine hashbasierte Negativliste (HCSL/HNL). Diese signiert der OCSP-Responder. Die signierten Listen stellt der OCSP-Responder auf seiner üblichen HTTP-Schnittstelle für anfragenden Clients zur Verfügung. Alle 10 Minuten aktualisiert der OCSP-Responder diese signierten Listen. Ein OCSP-Responder bietet zusätzlich zu der bereits bestehenden OCSP-Funktionalität die HCSL inkl. Delta-Dateien an.

Hinweis: Wenn im Folgenden von Hashwerten gesprochen wird, so ist stets die Verwendung der kryptographisch sicheren Hashfunktion SHA-256 [FIPS-180] für die Erzeugung der Hashwerte impliziert.

4.1 HCSL, Positivlisten (HPL) und Negativlisten (HNL)

Es gibt bei den HCSL zwei Typen: Positivlisten (HPL) und Negativlisten (HNL). Ob ein Client die HPL oder HNL verwenden möchte, hängt vom Client und dessen Anwendungsfällen ab. Es bleibt dem Client überlassen. Oft wird die Verwendung von HNL von Vorteil sein.

Ein TSP besitzt i. d. R. mehrere CAs und damit auch CA-Zertifikate, die Grundlage von EE-Zertifikatserstellungen sind. Für jede CA wird je eine Positivliste (HPL) und eine Negativliste (HNL) erzeugt.

Der Grundgedanke ist, dass man in der Positivliste einer CA (HPL), deren nichtgesperrten Zertifikate aufführt, und in der Negativliste deren gesperrte Zertifikate. Betrachtet man die Historie der verschiedenen Methoden zur Sperrinformationsübermittlung im PKI-Bereich, so sieht man insbesondere aus dem Ansatz der kaskadierten Bloomfilter [CRLITE], dass man diese Listen -- wenn man Positivlisten und Negativlisten zusammen betrachtet -- deutlich effizienter kodieren kann als im naiven Ansatz. Solch eine Optimierung der Kodierung wird bei HCSL eingesetzt (vgl. Abschnitt 4.1.1).

Eine HCSL -- egal ob HPL oder HNL -- besteht aus drei Bestandteilen:

1. Header,
2. Body,
3. Trailer.

Im Header befindet sich eine Zeitinformation, der "Authorization Key Identifier" (AKI) der eine CA aus Sicht eines EE-Zertifikats identifiziert und der "Subject Key Identifier" (SKI) der signierenden Identität (gleich wie beim "signierenden" OCSP-Responder-Zertifikat) und eine Längeninformation (HLen).

Der Body besteht aus einer Liste von auf HLen (vgl. folgenden Abschnitt) gekürzten Hashwerten von EE-Zertifikaten der CA.

Im Trailer befinden sich zwei Signaturen. Die erste Signatur wird klassisch über Header plus Body erzeugt. Die zweite Signatur wird über Header und dem Merkle-Hashbaum aus dem Elementen des Bodys erzeugt.

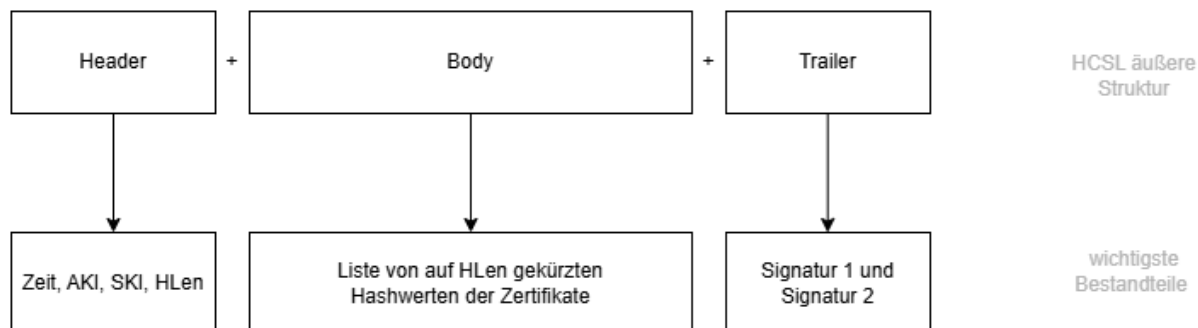


Abbildung 1: Grundsätzlicher Aufbau einer HCSL

4.1.1 Aufbau einer HPL

Ein TSP kennt alle von ihm erzeugten Zertifikate und deren Hashwerte (vgl. [gemSpec_PKI#GS-A_4693-*]). Sei P die Menge aller zeitlich und auch nach Sperrstatus gültigen Zertifikate und N die Menge aller zwar zeitlich noch gültigen aber nicht in P seienden Zertifikate. Sei Hash_P die Menge der Hashwerte von Zertifikaten aus P. Analog Hash_N von N. Nach Konstruktion müssen Hash_P und Hash_N disjunkt sein.

Ein TSP kürzt schrittweise die Hashwerte von Elementen Hash_P und Hash_N solange Hash_P und Hash_N noch disjunkt sind. Die so ermittelte Kürzungslänge der Hashwerte ist HLen. Die aus Hash_P stammende auf HLen gekürzte und sortierte Liste der Elemente aus Hash_P ist der Body einer HPL. Um die Auswertung eine HCSL im Client zu vereinfachen werden bei der Kürzung der Hashwerte nur Vielfache von 8 Bit betrachtet. Ein HLen gleich eins bedeutet, dass die Hashwerte auf die ersten 8 Bit gekürzt wurden, HLen gleich 2 bedeutet eine Kürzung auf 16 Bit etc..

Beispiel:

Folgend sei eine CA eines TSP gegeben, dessen zeitlich aktuell gültigen EE-Zertifikate bzw. deren Hashwerte folgende Mengen Hash_P und Hash_N erzeugen.

```
Hash_P = {
H_1 = 6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b
H_2 = d4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35
H_3 = 4e07408562bedb8b60ce05c1decfe3ad16b72230967de01f640b7e4729b49fce
H_4 = 4e227777d4dd1fc61c6f884f48641d02b4d121d3fd328cb08b5531fcacdabf8a
H_5 = ef2d127de37b942baad06145e54b0c619a1f22327b2ebbcfbec78f5564afe39d
H_6 = e7f6c011776e8db7cd330b54174fd76f7d0216b612387a5ffcfb81e6f0919683
H_7 = 7902699be42c8a8e46fbbb4501726517e86b22c56a189f7625a6da49081b2451
H_8 = 2c624232cdd221771294dfbb310aca000a0df6ac8b66b696d90ef06fdefb64a3
}

Hash_N= {
H_9 = 19581e27de7ced00ff1ce50b2047e7a567c76b1cbaebabe5ef03f7c3017bb5b7
H_10= 1944dc15364204a80fe80e9039455cc1608281820fe2b24f1e5233ade6af1dd5
}
```

Die CA hat im Beispiel (nur) 10 Zertifikate erzeugt, die aktuell zeitlich noch gültig sind. Sie hat eine Sperrrate von $2/10 = 0.2$, was in der Produktivumgebung der TI ungewöhnlich hoch wäre. Der TSP kürzt die Hashwerte auf 8 Bit (Hash_P_8 und Hash_N_8). Er erhält folgende Mengen Hash_P_8 und Hash_N_8:

```
Hash_P_8 = {
H_1_8 = 6b
H_2_8 = d4
H_3_8 = 4e
H_4_8 = 4e
H_5_8 = ef
H_6_8 = e7
H_7_8 = 79
H_8_8 = 2c
}

Hash_N_8 = {
H_9_8 = 19
H_10_8= 19
}
```

Der TSP erkennt im Beispiel drei Dinge: (1) H_3_8 ist gleich H_4_8, deshalb ist die Kardinalität von HP_8 gleich 7, (2) H_9_8 ist gleich H_10_8, deshalb ist die Kardinalität von Hash_N_8 gleich 1 und (3) Hash_P_8 und Hash_N_8 sind disjunkt. Aufgrund von (3) ist

HLen gleich 1. Bei der Erzeugung einer HPL würde der Body also 7 Elemente (auf 8 Bit gekürzte Hashwerte) enthalten.

Die Elemente im Body werden als Zahlen interpretiert und aufsteigend sortiert im Body aufgeführt. In einer HCSL sind also die Elemente im Body immer aufsteigend sortiert. (Im Beispiel ist der so berechnete Body gleich (hexdump) 2c4e6b79d4e7ef.)

Hinweis: Bei SMC-B-CA in der PU der TI, die viele Zertifikate ausgegeben haben, ergibt sich üblicher Weise eine HLen von fünf, bei anderen "kleineren" CAs eine HLen um die drei.

4.1.2 Aufbau einer HNL

Fast analog zur HPL wird eine HNL erzeugt -- es gibt eine Besonderheit. Die Menge Hash_N wird am Anfang um zwei besondere Elemente erweitert: das "Nullelement" (256 mal 0-Bits) und das "Einselement" (256 mal 1-Bits). Da die verwendete Hashfunktion eine kryptographisch sichere Hashfunktion ist, kann es keine Zertifikate bei einem TSP geben, die das Nullelement oder Einselement als Hashwert besitzen. Die so erweiterte Menge wird analog zum im vorherigen Abschnitt aufgeführten Ablauf auf eine durch den Algorithmus berechnete HLen gekürzt. Analog wird die erzeugte Menge sortiert und das Ergebnis als Body der HNL aufgeführt.

Der fachliche Hintergrund für die Erweiterung von HN um die zwei besonderen Elemente wird in Abschnitt 4.8 erläutert.

Um das Beispiel aus dem vorherigen Abschnitt fortzuführen: Hash_N_8 wäre dann

```
Hash_N_8 = {  
  H_Nullelement_8 = 00  
  H_9_8            = 19  
  H_10_8           = 19  
  H_Einselement_8 = ff  
}
```

Hash_N_8 und Hash_P_8 sind immer noch disjunkt. So bleibt HLen gleich eins. Da H_9_8 und H_10_8 gleich sind, ist die Kardinalität der berechneten Menge drei. Im Body der HNL würden dann also diese drei Elemente in sortierter Reihenfolge aufgenommen werden, d. h. im Beispiel gleich (hexdump) 0019ff.

4.1.3 Die zwei Signaturen einer HCSL

Im Trailer einer HCSL befinden sich zwei Signaturen. Die Signaturen werden mit der OCSP-Identität erzeugt -- wie auch die Signaturen von OCSP-Responses.

Die Erstellung der ersten Signatur ist einfach: Header und Body werden konkateniert und darüber wird eine Signatur erstellt. Dies ist die erste Signatur. Im Header einer HCSL ist der SKI der OCSP-Identität enthalten. Der Signaturwert wird -- ähnlich wie bei einer JWS -- auf eine einheitliche Länge ge-paddet [TR-03111#5.2.1 Plain Format] und binär im Trailer aufgeführt.

Die Erstellung der zweiten Signatur ist ein wenig komplexer. Sie wird über die Konkatenation von Header und Wurzelement des Hashbaumes aus den Elementen aus dem Body erzeugt -- also über der Konkatenation von Header und einem Hashwert (Wurzelknoten im Hashbaum, vgl. folgend H_0). Wieder wird die Signatur kodiert und ge-paddet gemäß [TR-03111#5.2.1 Plain Format] als zweites Element in Trailer aufgeführt.

Im Beispiel aus Abschnitt 4.1.1 ist der Body = [0x2c, 0x4e, 0x6b, 0x79, 0xd4, 0xe7, 0xef]. Es gilt Hash(0x2c) = d03502c....ce0b33e7, dieser sei mit H_a bezeichnet. Analog werden H_b bis H_g, als Hashwerte der folgenden Element aus Body bezeichnet. Dann ergibt sich

das Wurzelement, wie in der folgenden Abbildung dargestellt, als H_0
 $= b9ed45efb9b2debb3fc8629aa08d99c6caf7b1cac3fe285f489535ab03c508b6$.

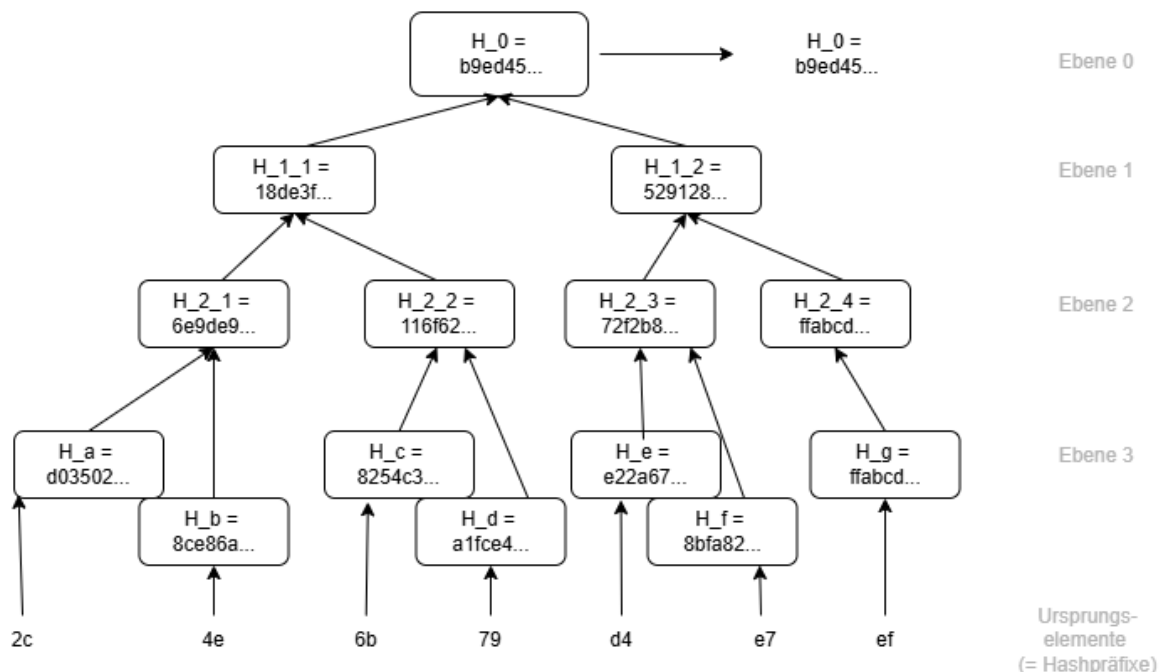


Abbildung 2: Beispiel-Berechnung von H_0 für die zweite Signatur in einer HPL

Im Hashbaum werden benachbarte Knoten konkateniert, und darüber der Hashwert erzeugt ($H_{2_1} = H(H_a || H_b)$). Dieser Hashwert ist dann das Element der nächsten Ebene. Elemente ohne Nachbarn (vgl. H_g im Beispiel) werden ohne Änderung auf die nächste Ebene kopiert (vgl. A_28718-*). Dies tut man pro Ebene so lange bis man nur noch ein Element erhält, dieses Element ist dann die Wurzel (H_0), auch Wurzelement genannt.

Hinweis: Im Gegensatz zu einer OCSP-Response enthält eine HCSL kein Zertifikat der signierenden OCSP-Identität (vgl. Datei hcsl-certs.cbor, A_28694-* bzw. oft sind alle OCSP-Identitäten sowieso schon durch die TLS-Auswertung im Client vorhanden).

Im Fall der HNL gibt es noch eine Besonderheit. Die direkte Nachbarschaftsbeziehung zwischen Knoten auf der Blatt-Ebene muss fixiert werden. Als Beispiel nehmen wir Folgendes an. Der Body einer HNL enthält 4 Elemente (Hashpräfixe): a, b, c, d. Da diese bei der Erzeugung des Bodys sortiert worden sind, gilt: $a < b < c < d$. Nun bilden wir aus allen direkten Nachbarn durch konkatenieren neue Elemente (die dann $H_{len} * 2$ lang sind): $a || b$, $b || c$, $c || d$. Diese drei neuen Elemente sind die Grundlage für die Erzeugung des Hashbaumes mit dem Wurzelement H_0 .

Die folgender Abbildung stellt die Konstruktion des Hashbaums dar und insbesondere des Wurzelements für das Beispiel der HNL aus Abschnitt 4.1.2. Der Body enthält drei Elemente: 00, 19, ff. Es entstehen daraus zwei Elemente, nämlich 0019 und 19ff, die die Grundlage für die Konstruktion des Hashbaums (weil Blätter des Hashbaums) und damit von H_0 sind. Es gilt $H(0019) = b35d97...$ und $H_0 = (H_{1_1} || H_{1_2})$.

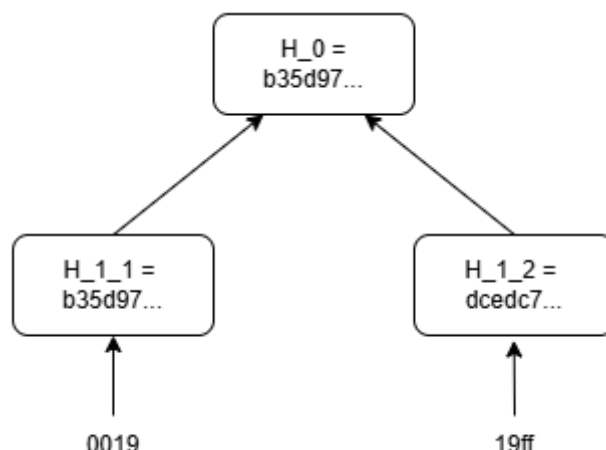


Abbildung 3: Beispiel-Berechnung von H_0 bei einer HNL für die zweite Signatur

Analog zur HPL wird anschließend die zweite Signatur über die Konkatenation von Header und Wurzelelement (H_0) erzeugt.

4.2 Unterschiedliche Paradigmen: OCSP und HCSL

Ein ganz entscheidender Unterschied zwischen OCSP und HCSL ist, dass OCSP Client-zentriert (interaktiv) ist und HCSL Client-unabhängig ist, wie folgend erläutert.

Bei OCSP kommt ein OCSP-Client auf den OCSP-Responder zu. Der OCSP-Responder erfährt erst im Moment der Anfrage, für welches Zertifikat der Client eine Sperrinformation erhalten möchte. Erst nach Auswertung des OCSP-Requests kann ein OCSP-Responder im Normalfall damit beginnen die Antwort für den Client zusammenzustellen. Dafür muss der OCSP-Responder aus seiner Datenbank den Sperrstatus des angefragten Zertifikats bestimmen und mit Hilfe der HSMs des OCSP-Responders die OCSP-Response signieren. Ein Caching ist nur im sehr beschränkten Maße fachlich möglich. Für die Mehrzahl der Clients wird die OCSP-Response eigens erzeugt. Lastkurven in den OCSP-Clients (bspw. eines ePA-Aktensystems) wirken sich direkt auf die Last im OCSP-Responder aus.

Bei HCSL ist die Kernidee, dass ein TSP ganz unabhängig von Client-Anfragen alle 10 Minuten die HCSLs (Plural) erzeugt. Die HCSLs werden als statische Dateien an der HTTP-Schnittstelle des OCSP-Responder abgelegt. Im Moment der Anfrage eines Clients soll der TSP maximal wenig Arbeit leisten müssen. Er liefert statische Dateien aus -- es ist keine Datenbank-Auswertung, kein HSM-Zugriff und auch kein Parsen von Request-Parametern auf Anwendungsebene im TSP notwendig. Auch wird es damit möglich die HCSL, mit geringem technischen Aufwand auf mehrere DL-Punkte in der TI zu "spiegeln" (vgl. Content Delivery Network (CDN)).

Das ist auch die Motivation für das folgende Namensschema für den Bezug der HCSL als statische Dateien an der HTTP-Schnittstelle -- es wird eben gerade keine RESTful-Schnittstelle verwendet.

4.3 Nutzung OCSP vs. HCSL

Eine Anwendung muss entscheiden, ob im entsprechenden Anwendungsfall die Nutzung von OCSP oder HCSL von Vorteil ist.

Für die Prüfung von AUT- oder ENC-Zertifikaten ist die Verwendung von HCSL besser geeignet als die Verwendung von OCSP. Dort wurden auch die meisten Last- und Verfügbarkeitsprobleme in der PU beobachtet. In diesem Kontext wird die Nutzung von HCSL am meisten helfen.

Bei Szenarien, die die langfristige Prüfbarkeit von OSIG-Signaturen (Kompromissmodell) benötigen, ist meist die Verwendung von OCSP mit Einbettung von OCSP-Antworten kurz nach dem Signaturvorgang (Signaturzeitpunkt) von Vorteil.

HCSL und OCSP bleiben als Möglichkeiten zum Bezug von Statusinformation von Zertifikaten bei den TSP der TI auch weiterhin parallel bestehen.

Ein TSP verwendet für die Erstellung von HCSL- oder OCSP-Informationen die gleiche Datenbasis. Wenn eine HCSL und eine OCSP-Antwort zum gleichen Zeitpunkt erzeugt wurden, werden sie deshalb den gleichen Zustand für das betrachtete EE-Zertifikat liefern. Ähnliches gilt bei dem Client-seitige Caching von Statusinformationen [gemSpec_PKI#A_23225] -- unterschiedliche Auskunftszeitpunkte erzeugen immer die Möglichkeit einer anderen, aktualisierten Information.

4.4 Namensschema für die Listen

An der HTTP-Schnittstelle des OCSP-Responders des TSP werden zusätzlich zur bisherigen OCSP-Funktionalität folgende Dateiarnten per HTTP-GET zur Verfügung gestellt.

1. Eine Datei namens hcs1-certs.cbor, die alle CA-Zertifikate und alle OCSP-Signer-Zertifikate des TSP aufführt (A_28694-*).
2. Pro CA eine Datei namens hcs1-config-<AKI>.cbor, die einem Client ermöglicht, das optimale Zeitintervall für den HCSL-Bezug bei einem OCSP-Responder zu ermitteln (vgl. Abschnitt 4.6 und A_23191-*).
3. HPL mit der Namenskonvention hpl-<Erzeugungszeit>-<AKI>
4. Delta-Dateien für HPL mit der Namenskonvention d-hpl-<VonZeit>-<ZuZeit>-<AKI>
5. HNL mit der Namenskonvention hnl-<Erzeugungszeit>-<AKI>
6. Delta-Dateien für HNL mit der Namenskonvention d-hnl-<VonZeit>-<ZuZeit>-<AKI>

Beispiel:

Ein TSP hat eine CA mit SKI e5fa44f2b31c1fb553b6021e7360d07d5d91ff5e und erzeugt um 1780130615 (Unix-Zeit) eine HPL. Es entsteht auf dem Download-Punkten eine Datei namens hpl-1780130615-e5fa44f2b31c1fb553b6021e7360d07d5d91ff5e. Zehn Minuten später erzeugt der TSP die nächste (also folgende) HPL. Weil 1780130615 + 60*10 = 1780131215, hat diese HPL den Dateinamen hpl-1780131215-e5fa44f2b31c1fb553b6021e7360d07d5d91ff5e. Nach Erzeugung dieser HPL, erzeugt der TSP die Delta-Datei namens d-hpl-1780130615-1780131215-e5fa44f2b31c1fb553b6021e7360d07d5d91ff5e.

4.5 Delta/Update-Dateien

Fast immer werden sich zwei zeitlich nahe beieinander liegende HCSL nur wenig unterscheiden:

- Im Header wird eine jüngere Zeitinformation aufgeführt.
- Im Body wird es im Normalfall (wenn überhaupt) nur wenige Aktualisierungen geben (einige Zertifikate kommen hinzu oder werden gesperrt und damit entfernt).
- Die Signaturen im Trailer werden sich unterscheiden.

Wenn es keine Aktualisierungen im Body gibt, weil kein Zertifikat hingekommen oder gesperrt (entfernt) wurde, so ist eine Delta-Datei nur 206 Bytes groß.

In der PU der TI werden bei einer SMC-B-CA innerhalb eines 10 Minuten Intervalls üblicher Weise nur wenige Zertifikate (im niedrigen zweistelligen Bereich) hinzugefügt. Abhängig von der HLen ergeben sich üblicher Weise Dateigrößen von um die 200 Bytes bei Delta-Dateien.

Damit ist es von Vorteil, wenn ein Client nur die Änderungen beziehen kann, mit der er eine bereits lokal vorhandene HCSL aktualisieren kann. Der genaue Aufbau der Delta-Dateien ist in Abschnitt 5.1.6 definiert. Ein TSP stellt Delta-Dateien mindestens für zwei zeitlich aufeinander folgende HCSLs zur Verfügung.

4.6 Bereitstellung durch den TSP

Der TSP-X.509 nonQES SMC-B erzeugt die HCSL im 10-Minuten-Takt, unabhängig davon, ob ein Client diese anfragt. Der TSP erzeugt nach Erzeugung einer HCSL immer eine Delta-Datei zur letzten erzeugten HCSL.

Der TSP fängt initial an die HCSL zu erzeugen. Der TSP muss dabei im Normalfall mehrere CA-en betrachten (A_23191-*). Er fängt mit der ersten CA mit SKI "<SKI>" an. Es ist der Zeitpunkt t. Er erzeugt die HPL und die HNL für Zeitpunkt t für die erste CA. Er erzeugt die Datei hcs1-config-<SKI>.cbor mit folgenden Inhalt:

```
{
  "type": "hcs1-config-data",
  "ski" : <SKI>,
  "start": t (Unix-Zeit als integer > 0),
  "delay": d (Sekunden als integer > 0),
  "period": p (Sekunden als integer > 0)
}
```

Diese Datei, die erzeugte HPL und die erzeugte HNL stellt der TSP auf den Download-Punkten zur Verfügung (A_23191-*). Zwischen Erstellungszeit (t) und dem Zeitpunkt, zu dem die HCSL-Dateien auf dem Download-Punkt zur Verfügung stehen, gibt es im Normalfall eine Verzögerung (wenige Sekunden). Diese hängt von der Implementierung und den technischen Rahmenbedingungen im TSP ab. Der TSP trägt in "delay" einen Wert, bei dem zu erwarten ist, dass nach dieser Verzögerungszeit im Normalfall die Dateien nach der Erzeugung auf dem Download-Punkt zur Verfügung stehen. Der Wert p ist im Normalfall 600 Sekunden (= 10 Minuten, vgl. A_23191-*). Ab dem Zeitpunkt t erstellt der TSP immer zu den Zeitpunkten $t + kp$ ($k \in \mathbb{N}^*$) für diese CA eine neue HPL, eine HNL und entsprechende Delta-Dateien und stellt all diese auf dem Download-Punkt zur Verfügung gestellt. Analog geht der TSP mit seinen anderen CAs vor.

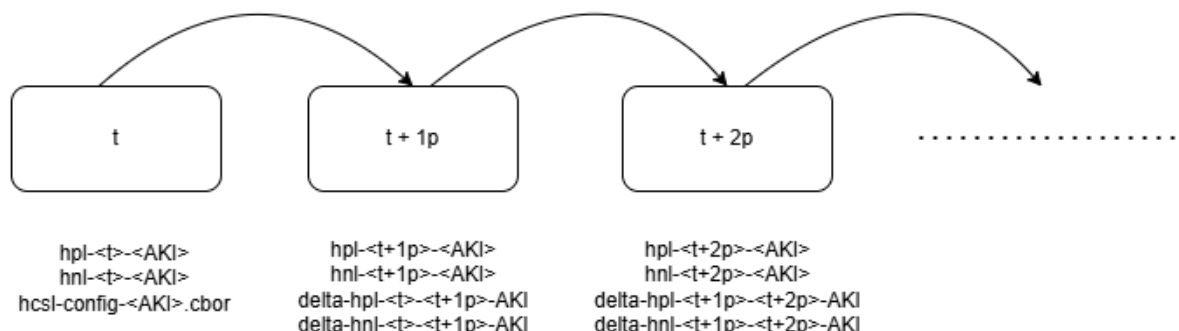


Abbildung 4: Erzeugung der HCSL+Delta-Dateien mit Periode p im TSP

Für jeden Zeitpunkt $t + kp$ ($k \in \mathbb{N}^*$) existiert eine HPL und eine HNL für eine CA des TSP. Dabei ist der Zeitpunkt t spezifisch für den Startpunkt der Erzeugung im TSP für eine CA. Sollte es eine grobe Störung im TSP geben (bspw. mehrere Stunden Stromausfall), so startet der TSP für die CA erneut mit neuer `hcsl-config-<SKI>.cbor` und neuer "start"-Zeit darin.

Für zwei aufeinander folgende Zeitpunkte $t + kp$ und $t + (k + 1)p$ existiert für jede HCSL eine Delta-Datei, die es einem HCSL-Client erlaubt effizient eine lokal für Zeitpunkt $t + kp$ vorliegende HCSL auf die HCSL für den Zeitpunkt $t + (k + 1)p$ zu aktualisieren.

Im Normalfall hat ein TSP mehrere CAs, jede mit spezifischer SKI und damit spezifischer `hcsl-config-<SKI>.cbor`.

Für die Erzeugung einer Delta-Datei benötigt man keine kryptographischen Operationen oder privates Schlüsselmaterial. Eine Delta-Datei führt die Änderungen auf, die ein Client ein an einer lokal vorhandenen HCSL für Zeitpunkt t vornehmen muss um auf die HCSL für den Zeitpunkt $t + mp$ ($m \in \mathbb{N}^*$) zu gelangen. D. h. jeder kann zwischen zwei beliebigen HCSL einer CA eine Delta-Datei erzeugen.

4.7 Nutzung einer Liste im Fachdienst als HCSL-Client

Ein HCSL-Client (bspw. ein ePA-Aktensystem) muss für die Nutzung der HCSL zwei Dinge tun:

1. Der HCSL-Client muss die regelmäßige Aktualisierung seiner lokal vorliegenden HCSL pro CA (vgl. Periodenlänge "period" in Abschnitt 4.6) durchführen. Vor der regelmäßigen Aktualisierung gibt es einmalig den initialen Bezug der HCSL. Die regelmäßige Aktualisierung findet im Client statt, egal ob es aktuell im Client EE-Zertifikate zu prüfen gibt oder nicht.
2. Zum Zeitpunkt einer Zertifikatsstatusprüfung verwendet der HCSL-Client die bei ihm lokale vorhandenen HCSL-Informationen für die Prüfung. Eine Zertifikatsstatusprüfung kann also ohne externe Abhängigkeiten im Client selbst durchgeführt werden.

Punkt 1: (a) Regelmäßige Aktualisierung der HCSL im Client:

Im Client liegen die Downloadpunkte der HCSL (vgl. ServiceSupplyPoints in der TSL, Informationen aus dem zu prüfenden EE-Zertifikat selbst oder aus anderen Quellen) pro CA und die CA-Zertifikate (und damit die SKI der CAs) vor. Pro CA prüft der Client, ob auf einen der Downloadpunkte der CA HCSL-Daten grundsätzlich vorliegen, in dem er die

547 Datei hcs1-config-<AKI>.cbor per HTTP-GET bezieht. Ist diese nicht vorhanden, so ist
548 aktuell die Verwendung von HCSL für diese CA nicht möglich.

549 Nehmen wir an, dass die Verwendung von HCSL für die betrachtete CA möglich ist und im
550 Client noch keine HCSL für die betrachtete CA vorliegt.

551 Sei t_{aktuell} die aktuelle Zeit im Client (Unix-Zeit in Sekunden). Mit den Informationen aus
552 der HCSL-Config-Datei (start, period, delay) kann der Client wie folgt z berechnen.

$$k = \lfloor (t_{\text{aktuell}} - t_{\text{start-aki}} - \text{delay}) / p \rfloor$$

$$z = t_{\text{start-aki}} + kp$$

556 Nun kann der Client vom Download-Punkt die gewünscht HCSL (bspw. hpl-<z>-<AKI>,
557 A_28954-*) beziehen. Ab jetzt führt der Client alle "period" Sekunden eine Aktualisierung
558 der nun lokal vorliegenden HCSL bei sich durch (vgl. Abschnitt 4.5). Dafür verwendet er
559 die Delta-Dateien, also period Sekunden später bezieht er bspw. d-hpl-<z>-<z+p>-
560 <AKI> (A_28947-*).

561 Sollte der Client eine signifikante Anzahl von Perioden verpasst haben, weil er bspw. zu
562 Wartungszwecken stromlos war, so bezieht er erneut initial eine aktuelle HCSL und
563 aktualisiert diese ab dann in period-Rhythmus mittels Delta-Dateien.

564 Punkt 1: (b) Vorbereitung der HCSL-Daten für eine Zertifikatsstatusprüfung:

565 Die nun ständig aktuell gehaltenen im Client lokal vorhandenen HCSL-Daten werden nach
566 jeder Aktualisierung aufbereitet, damit eine später stattfindende EE-
567 Zertifikatsstatusprüfung maximal schnell erfolgen kann.

568 Bei der nun (per Delta-Datei aktualisierten) HCSL prüft der Client die erste Signatur (vgl.
569 Abschnitt 4.1.3). Sollte im Client eine beschränkte Laufzeitumgebung (vgl. Abschnitt 4.8)
570 vorliegen, so prüft der Client auch die zweite Signatur. Sollte eine Signaturprüfung
571 fehlschlagen (Signaturprüfung ergibt INVALID), so muss ein Alarm erzeugt werden (vgl.
572 Abschnitt A_29029-*).

573 Ist die Signaturprüfung erfolgreich so überführt ein Client im Normalfall die Daten aus der
574 HCSL in eine für seine Umgebung optimale Datenstruktur. Bspw. werden die Elemente
575 aus dem Body (Hashwertpräfixe) in ein Associative Array (Dictionary) überführt. Es steht
576 dem Client auch frei dies nicht zu tun und in der HCSL per binärer Suche im Moment der
577 Zertifikatsstatusprüfung im sortierten Body effizient zu suchen.

578 Punkt 2: Moment der Zertifikatsstatusprüfung im Client:

579 Im Client liegt nun ein zu prüfendes EE-Zertifikat vor. Zunächst wird die übliche
580 Signaturkettenprüfung der Zertifikatssignatur(en) durchgeführt. Bei positiven
581 Prüfergebnis verbleibt als Aufgabe nur noch die Zertifikatsstatusprüfung. Der Client
582 bezieht aus dem EE-Zertifikat den AKI. Liegt lokal eine zeitlich gültige HCSL -- bzw. schon
583 dessen in optimale Datenstrukturen überführten Daten vor, so wird der auf HLen gekürzte
584 Hashwert des Zertifikats in den im Body aufgeführten Wertes aus der HCSL gesucht. Das
585 Suchergebnis liefert den Zertifikatsstatus -- findet sich der gekürzte Hashwert in einer
586 aktuellen HPL ist das Zertifikat "good" anderenfalls ist es gesperrt, analog mit inverser
587 Semantik bei einer HNL.

4.8 Nutzung einer Liste in beschränkten Laufzeitumgebungen (HSM)

Eine Zertifikatsstatusinformation soll in einer beschränkten Laufzeitumgebung verwendet werden, bspw. im Rahmen einer Berechtigungsvergabe ePA innerhalb eines VAU-HSM. Dafür bereitet der "äußere" Fachdienst (= eine Komponente die nicht das VAU-HSM ist) die Informationen aus einer HCSL in Bezug auf das zu prüfende Zertifikat für ein VAU-HSM auf.

Zunächst betrachten wir den Fall, dass das zu prüfende EE-Zertifikat gültig ist und deshalb dessen gekürzter Hashwert in einer aktuellen HPL aufgeführt ist. Als Beispiel geht es um das EE-Zertifikat mit dem Hashwert 2c624232...b64a3 (vgl. H_8 aus Abschnitt 4.1.1). Aus dem HLen-Wert der HPL ergibt sich, dass man den auf 8 Bit gekürzten Hashwert (weil HLen=1) im Folgenden betrachten muss: also 2c. In folgender Abbildung verfolgen wir den Pfad im Hashbaum von Blatt 2c zur Wurzel H_0.

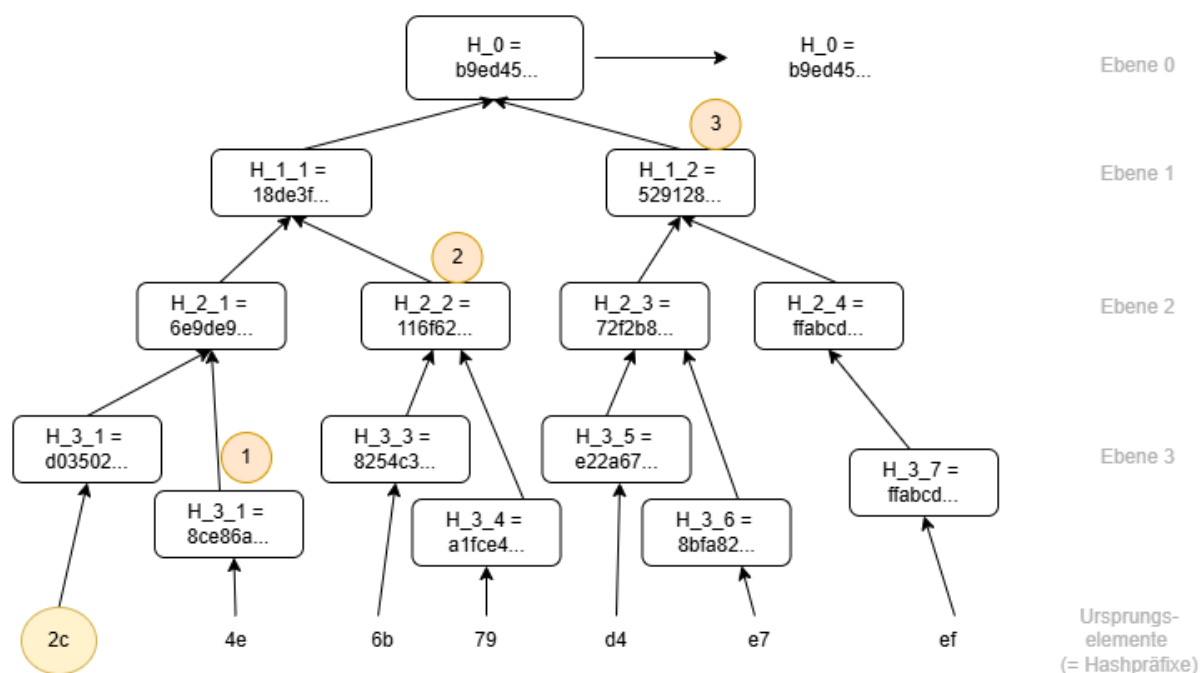


Abbildung 5: Pfad von einem Blatt-Element zur Wurzel im Hashbaum

Der Pfad zur Wurzel H_0 lässt sich beschreiben als l("2c"), r(H_3_1), r(H_2_2), r(H_1_2) -- "l" bedeutet es handelt sich um einen links stehendes Element handeln, analog "r" für ein rechts stehendes Element. Wenn man der beschränkten Laufzeitumgebung also

1. das zu prüfende EE-Zertifikat,
 2. der Header der aktuellen HPL (also 53 Bytes),
 3. den Pfad (hier im Beispiel 99 Bytes) und
 4. die zweite Signatur (also 64 Bytes) der HPL übergibt,
- so kann

1. das VAU-HSM das EE-Zertifikat wie üblich kryptographisch prüfen,

2. dann falls mit positiven Prüfergebnis geprüft (also EE-Zertifikat ist Teil der TI-PKI) den Hashwert berechnen,
3. den Hashwert auf die im HPL-Header angegebene Länge kürzen (HLen),
4. mit dem Ergebnis und dem Pfad das Wurzelement H_0 berechnen,
5. anschließend die zweite Signatur über Header || H_0 prüfen.

Lässt sich die zweite Signatur mit positivem Prüfergebnis prüfen (VALID), so ist das EE-Zertifikat aktuell gültig.

Die beschränkte Laufzeitumgebung muss also -- dank der zweiten Signatur und den Pfadinformationen -- gar nicht die komplette HPL kennen, sondern nur wenige Bytes daraus. Die beschränkte Laufzeitumgebung ist also sicherheitstechnisch unabhängig, d. h. die Laufzeitumgebung muss dem "äußeren" Fachdienst nicht vertrauen, sondern kann die für sie relevanten Informationen aus einer HCSL sicher auswerten, ohne die HCSL vollständig kennen zu müssen.

Als Beispiel nehmen wir an eine HPL hat 200.000 Element im Body und HLen ist gleich 4. Dann ist die Baumhöhe 18. Header (53), Pfad $((4+1)*18)$ und zweite Signatur (64 Bytes) haben dann in Summe 207 Bytes.

Die Nutzung einer HNL in einer beschränkten Laufzeitumgebung verläuft sehr ähnlich wie die Nutzung einer HPL -- es gibt jedoch eine kleine Besonderheit: im Hashbaum wird die direkte Nachbarschaftsbeziehung zwischen Hashpräfixen festgehalten. (Erinnerung: die Elemente im Body (= Hashwertpräfixe) sind sortiert in einer HCSL.) Damit ergibt sich für das eben aufgeführte Beispiel -- Zertifikat mit Präfix "2c" ist zu prüfen -- bei der Nutzung einer aktuellen HNL folgender Graph.

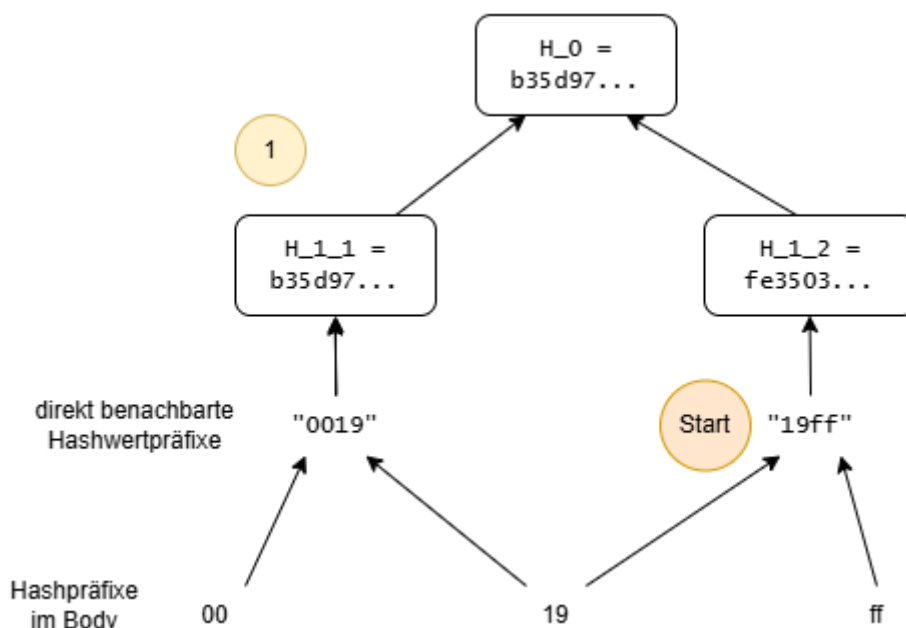


Abbildung 6: Pfad von einem Blatt-Element zur Wurzel im Hashbaum

Da gilt $19 < 2c < ff$, wird für die Ermittlung des relevanten Pfades im Graph der rechte Knoten betrachtet. Die beschränkte Laufzeitumgebung muss als Pfad erhalten $r("19ff")$, $l(H_{1_1})$. Damit kann sie H_0 berechnen und mit dem Header und H_0 kann sie die zweite Signatur prüfen. Lässt sich die zweite Signatur mit positivem Prüfergebnis prüfen, so steht fest: "19ff" ist Teil des Hashbaums und deshalb sind 19 und ff direkte Nachbarn. Deshalb ist 2c nicht Teil des Hashbaums, deshalb ist das betrachtete EE-Zertifikat aktuell nicht gesperrt.

Da bei der Erzeugung der HNL das Nullelement und das Einselement eingebracht werden, kann ein HCSL-Client mit beschränkter Laufzeitumgebung aus einem Pfad des Merkle-Hashbaum das Vorhandensein oder Nichtvorhandensein jedes möglichen Hashwertes erkennen. Im Sinne von: es gibt keine "Lücken" an den Rändern. Hätte im o. g. Beispiel ein Zertifikat bspw. den gekürzten Hashwert 15 (mit $15 < 19$), ist aus dem Pfadanfang 0019 klar, dass 00 und 19 direkte Nachbar sind und deshalb 15 nicht im Merkle-Hashbaum ist, und deshalb das zu prüfende EE-Zertifikat auch nicht gesperrt ist. Ohne initiale Einbringung des Nullelement im o. g. Beispiel wäre dies ansonsten nicht erkennbar.

4.9 nonQES vs. QES

Bei dem Thema HCSL gelten die Vorgaben aus [gemF_HCSL] nur für den nonQES-Bereich. Der QES-Bereich wird aktuell nicht betrachtet. Sollten HCSL in einer späteren Umsetzungsphase auch im QES-Bereich verwendet werden, so ist sicherlich die Verwendung der HPL mit HLen gleich 32 die zu wählende Vorgehensweise.

4.10 Post-Quanten-Kryptographie

Ein Wechsel auf PQC-Signaturverfahren sind im Kontext HCSL leicht möglich. Aktuell verwenden alle die für HCSL relevanten CA-en ECDSA als Signaturverfahren. Kommt dort es in der TI-PKI zum Einsatz von PQC-Signaturverfahren wie ML-DSA, so werden die zwei Signaturen im Trailer einer HCSL mit ML-DSA anstatt von ECDSA erzeugt. Aus dem Header (SKI) ist eindeutig für einen Client zu erkennen welches Signaturverfahren verwendet werden muss. An anderen Inhalten einer HCSL muss keine Änderung vorgenommen werden, weil moderne kryptographisch sichere Hashfunktionen in ihrer Sicherheitsleistung nicht substantiell durch kryptographisch relevante Quanten-Computer (CRCQ) beeinträchtigt werden.

4.11 Zusammenfassung Aufbau HCSL und Delta-Dateien

In folgender Tabelle ist der Aufbau einer HCSL nochmal informativ in Gesamtüberblick dargestellt.

Tabelle 1: Zusammenfassung Aufbau HCSL

offset	Länge	Beschreibung
0	4	Erkennungsmerkmal (magic bytes) "hpl1" oder "hnl1"
4	8	Erstellungszeit als 64-Bit Unixzeit kodiert
12	20	Authority Key Identifier (AKI) der CA (= SKI im CA-Zertifikat) über deren Zertifikate eine Aussage in der HCSL gemacht wird
32	20	Subject Key Identifier (SKI) der signierenden Identität (SKI des Signaturzertifikats = OCSP-Responder-Signaturzertifikat)

52	1	hlen -- Anzahl der Bytes auf den die Zertifikats-Hashwerte gekürzt kollisionsfrei werden. Der Wert ist ein Wert zwischen [1,32]
Ende Header		
53	8	nr -- Anzahl der nun folgenden auf hlen gekürzten Zertifikats-Hashwerte nr ist in Network-Byteorder (big-endian) kodiert
61	nr*hlen	nr*hlen mal auf hlen gekürzte Hashwerte
Ende Body		
61 + nr*hlen	64	erste ECDSA-Signatur im plain-format über die Daten von offset 0 bis direkt vor der ersten Signatur
61 + nr*hlen + 64	64	zweite ECDSA-Signatur im plain-format über die Daten von offset 0 bis zum Ende des Headers (bis offset 53) konkateniert mit dem Wurzelement des Merkle Hashbaums

673

674 Eine HCSL wie in der Tabelle aufgeführt hat damit eine Gesamtlänge von $53 + 8 +$
675 $nr*hlen + 64 + 64$ Bytes.

676 In folgender Tabelle ist der Aufbau einer Delta-Datei informativ in Gesamtüberblick
677 dargestellt.

678

679 **Tabelle 2: Zusammenfassung Aufbau Delta-Datei**

offset	Länge	Beschreibung
0	1	Erkennungsmerkmal (magic bytes) das Byte "d"
1	53	Header der neuen HCSL
54	8	Erstellungszeit der alten HCSL als 64-Bit Unixzeit kodiert
62	8	nr_add, Anzahl der hinzugekommenen gekürzten Hashwerte, 64-Bit Wert, Network-Byteorder (big-endian)
70	nr_add*hlen	nr_add mal auf hlen gekürzte Hashwerte, die hinzugefügt werden müssen
70+nr_add*hlen	8	nr_rm, Anzahl der zu entfernenden gekürzten Hashwerte, 64-Bit Wert, Network-Byteorder (big-endian)

Feature: Hash based Certificate Status List (HCSL)

70 + nr_add*hlen + 8	nr_rm*hlen	nr_rm mal auf hlen gekürzte Hashwerte, die entfernt werden müssen
70 + nr_add*hlen + 8 + nr_rm*hlen	64	erste ECDSA-Signatur der neuen HCSL
70 + nr_add*hlen + 8 + nr_rm*hlen + 64	64	zweite ECDSA-Signatur der neuen HCSL

680

681 Eine Delta-Datei bei der zwischen alter und neuer HCSL sich keine Elemente im Body
682 geändert haben, was in der Praxis am häufigsten vorkommt, hat damit eine Größe von
683 206 Bytes.

5 Spezifikation

5.1 Anforderungen an den TSP

A_28953 -TSP-X.509 nonQES: AKI einer CA ist 160-Bit lang

Ein TSP-X.509 nonQES SMC-B bzw. HBA MÜSSEN sicherstellen, dass der SKI in allen seiner CA-Zertifikate 160-Bit lang ist, und der SKI einer CA als AKI in den EE-Zertifikaten aufgeführt ist. [$\leq$]

Hinweis: die Forderung aus A_28953 wird von allen TSPs der TI erfüllt, da sie die SKI und AKI gemäß [RFC-5280#4.2.1.2 Option 1] erzeugen. Der Zweck der Anforderung A_28953 ist die Abhängigkeit transparent zu machen.

A_23191 -TSP-X.509 nonQES: HPL und HNL alle 10' erstellen

Ein TSP-X.509 nonQES SMC-B bzw. HBA MUSS für alle seine CAs jeweils eine HPL und eine HNL alle 10 Minuten erstellen und auf der HTTP-Schnittstelle seines OCSP-Responders den Clients zum Download zur Verfügung stellen.

Dabei MUSS folgende Namenskonvention verwendet werden:

Wenn

1. $\langle \text{time} \rangle$ die Erzeugungszeit (Unix-Zeit) ist (vgl. Headerdefinition einer Liste, A_28696-*) als natürliche Zahl ASCII-kodiert und
2. $\langle \text{AKI} \rangle$ der Authorization Key Identifier, der EE-Zertifikate der entsprechenden CA (also der Subject Key Identifier (SKI) im CA-Zertifikat) im Hexformat (kleingeschrieben) ist,

so MUSS eine HPL unter den Dateinamen $\text{hpl-}\langle \text{time} \rangle\text{-}\langle \text{AKI} \rangle$ veröffentlicht werden.

Analog MUSS dies für die HNL gelten, entsprechend mit "hnl": Dateinamen $\text{hnl-}\langle \text{time} \rangle\text{-}\langle \text{AKI} \rangle$.

[$\leq$]

Hinweis: ein AKI ist 160 Bit lang, weshalb der AKI im Hexformat kodiert 40 Zeichen ($=160 \cdot (1/8) \cdot 2$) lang ist.

A_28972 -TSP-X.509 nonQES: HCSL eine Woche lang zur Verfügung stellen

Ein TSP-X.509 nonQES SMC-B bzw. HBA MUSS die von ihm erzeugten HCSLs (HPL und HNL) für mindestens eine Woche HCSL-Clients an dessen HTTP-Schnittstellen zur Verfügung stellen. Ab dann kann er die HCSL, die älter als eine Woche sind löschen, und ebenfalls alle Delta-Dateien, die mit diesen HCSL in Beziehung stehen. [$\leq$]

Hinweis: Anwendungsfälle, die Zertifikatsstatusinformationen von ggf. bereits zeitlich abgelaufenen Zertifikaten benötigen, müssen dafür OCSP verwenden. Aktuell benötigt Anwendung der TI diesen Usecase.

Implementierungshinweis zu A_28972-*:

Eine Möglichkeit wäre täglich per cronjob den aktuellen Zeitpunkt t zu ermitteln und dann alle Dateien $\text{hpl-}\langle i \rangle\text{-}*$, $\text{hnl-}\langle j \rangle\text{-}*$, $\text{d-hpl-}\langle k \rangle\text{-}*$, $\text{d-hnl-}\langle j \rangle\text{-}*$ zu ermitteln bei denen i, j, k und j jeweils kleiner als $t - 60 \cdot 60 \cdot 24 \cdot 7$ ist. Diese kann der TSP dann löschen.

A_28694 -TSP-X.509 nonQES: Bereitstellung von hcsl-certs.cbor

Ein TSP-X.509 nonQES SMC-B bzw. HBA MUSS für alle seine CAs, für die er HCSLs zur Verfügung stellt,

1. die CA-Zertifikate und
2. die dazugehörigen verwendeten OCSP-Signer-Zertifikate
an den HTTP-Schnittstellen seines OCSP-Responders den HCSL-Clients zur Verfügung stellen. Dafür führt er die CA- und OCSP-Signer-Zertifikate DER-kodiert in einem Array auf (Sortierung egal), kodiert dieses Array mittels CBOR [RFC-CBOR] und speichert das Ergebnis in einer Datei namens hcs1-certs.cbor. Diese Datei MUSS der TSP über dessen HTTP-Schnittstellen allen HCSL-Clients bereit stellen.
[<=]

A_28952 -TSP-X.509 nonQES: Bereitstellung von hcs1-config-<aki>.cbor

Ein TSP-X.509 nonQES SMC-B bzw. HBA MUSS für alle seine CAs, für die er HCSLs zur Verfügung stellt, an der HTTP-Schnittstellen seines OCSP-Responders je CA eine Datei namens hcs1-config-<aki>.cbor den HCSL-Clients zur Verfügung stellen. Dabei MUSS er <aki> im Dateinamen mit der Hexadezimaldarstellung (0-9a-f) der AKI der jeweiligen CA ersetzen. Die Datei MUSS mittels CBOR [RFC-CBOR] kodiert sein und folgende Struktur aufweisen:

```
{
  "type": "hcs1-config-data",
  "aki" : <AKI> (binär kodiert),
  "start": t (Unix-Zeit als integer > 0),
  "delay": d (Sekunden als integer > 0),
  "period": p (Sekunden als integer > 0)
}
```

Die AKI in der Datenstruktur wird binär kodiert und ist i. d. R. ein SHA-1-Wert, also 160-Bit = 20 Byte lang.

Der Wert "period" MUSS die Periode sein mit der der TSP für die entsprechende CA die HCSL ab "start"-Zeitpunkt immer erzeugt (vgl. A_23191-*). Der Wert "delay" ist die Wartezeit, die ein HCSL-Client nach start+k*p mit k>=0 warten muss damit er erwarten kann, dass die HCSL für den Zeitpunkt start + k*p am Download-Punkt verfügbar ist. [<=]

Beispiel (vgl. Abschnitt 7):

```
$ ./create-hcs1-config.py
hcs1-start-54bc93f447a363a6289e11df150ca93f9155071d.cbor, starting time
2026-04-05 12:07:34
$ cbor2 -p hcs1-start-54bc93f447a363a6289e11df150ca93f9155071d.cbor
{
  "type": "hcs1-start-data",
  "aki": "T\\x93\\xf4G\\xa3c\\xa6(\\x9e\\u0011\\xdf\\u0015\\f\\xa9?\\x91U\\u0007\\u001d",
  "start": 1775383654,
  "delay": 10,
  "period": 600
}
$ xxd -p hcs1-start-54bc93f447a363a6289e11df150ca93f9155071d.cbor
a564747970656f6863736c2d73746172742d6461746163616b695454bc93
f447a363a6289e11df150ca93f9155071d6573746172741a69d234666564
656c61790a66706572696f64190258
```

A_28695 -TSP-X.509 nonQES: grundsätzlicher Aufbau einer HCSL

Ein TSP-X.509 nonQES SMC-B bzw. HBA MUSS eine HCSL als Konkatenation von folgenden drei Bestandteilen erstellen:

1. Header (vgl. A_28696-*),
2. Body (vgl. A_28702-* und A_28714-*),

3. Trailer bestehend aus zwei Signaturen (vgl. A_28717-* und A_28722-*).

[<=]

5.1.1 HCSL: Header

A_28696 -TSP-X.509 nonQES: Aufbau Header einer HCSL

Ein TSP-X.509 nonQES SMC-B bzw. HBA MUSS den Header einer HCSL aus folgenden Bestandteilen durch Konkatination erzeugen:

Header := "hpl1 | hnl1" (Magic Bytes) || <time> || <AKI> || <SKI> || <HLen>

1. Bei einer HPL MÜSSEN die ersten 4 Byte "hpl1" sein. Bei einer HNL MÜSSEN die ersten 4 Byte "hnl1" sein.
2. time MUSS die Erzeugungzeit (Unixzeit). Diese MUSS als 64-Bit lang als unsigned int big-endian kodiert werden.
3. AKI MUSS der Authorization Key Identifier (160 Bit) aus dem "signierenden" Zertifikat (das gleiche Zertifikat wie das OCSP-Signer-Zertifikat) sein und damit gleich dem SKI aus dem CA-Zertifikat (vgl. [gemSpec_PKI#A_23142]) sein.
4. SKI MUSS der Subject Key Identifier (160 Bit) aus dem "signierenden" Zertifikat (das gleiche Zertifikat wie das OCSP-Signer-Zertifikat) sein.
5. HLen MUSS eine 8-Bit-Zahl aus dem Intervall [1,32] sein. (Hinweis: der Wert von HLen wird bei der Erzeugung des Bodys ermittelt und anschließend im Header bei HLen eingetragen. Nach Konstruktion können nur Werte von 1 bis 32 angenommen werden.)

[<=]

5.1.2 HPL: Body

A_28702 -TSP-X.509 nonQES: Erzeugung des Bodys einer HPL

Ein TSP-X.509 nonQES SMC-B bzw. HBA MUSS bei der Erzeugung des Bodys einer HPL einer CA wie folgt vorgehen:

1. Sei HP die Liste der SHA-256-Hashwerte der von der betrachteten CA ausgestellten (signierten) EE-Zertifikate, die jetzt gültig sind (also nicht gesperrt und zeitlich gültig).
2. Sei HN die Liste der SHA-256-Hashwerte der von der betrachteten CA ausgestellten (signierten) EE-Zertifikate, die jetzt gesperrt und zeitlich noch gültig sind. Nach Konstruktion sind HP und HN disjunkt. (vgl. "Bemerkung zu A_28702-*" Punkt (1)). (Hinweis: EE-Zertifikate die zeitlich gültig sind, und vom einem OCSP-Responder weder als "good" noch als "revoked" beantwortet werden, sollen auch Teil der HN sein.)
3. Sei $i=1$, falls die Start-HLen aus A_28711-* noch nicht existiert, anderenfalls sei $i=\text{Start-HLen}$ aus A_28711-*. Sei HP_i die auf die ersten 8-Bit gekürzten Hashwerte der Elemente aus HP. Analog HN_i . So lange HP_i und HN_i nicht disjunkt sind, wird i um eins erhöht und HP_i und HN_i als die ersten $i*8$ Bit gekürzten Hashwerte (als Hashwerteprefixe bezeichnet) aus HP und HN berechnet. (Hinweis: Da HP und HN disjunkt sind, wird die Schleife spätestens bei $i=32$ terminieren. Im Regelfall sehr viel früher.)
4. HLen ist das i bei dem die Schleife aufhört, also HP_i und NL_i disjunkt sind.

5. Die Elemente aus HP_HLen werden als Zahlen (big-endian unsigned) interpretiert, ggf. doppelte Element werden entfernt, und die Elemente (Zahlen) werden aufsteigend sortiert. Sei N die Anzahl der Elemente der so erzeugten Liste.

6. Die Element aus der so erzeugten sortierten Liste werden kontaktiert aufgeschrieben, und das Ergebnis sei C.

7. N wird als 64 Bit unsigned big-endian kodiert und N || C ist der Body der HPL.

[<=]

Bemerkungen zu A_28702-*:

1. Sollte HP leer sein, so hat die CA noch keine Zertifikate erstellt. Dann kann es auch keine EE-Zertifikatsprüfung geben, dann wird auch keine HCSL benötigt.

2. Es kann in seltenen Fällen vorkommen dass HN leer ist, wenn noch kein EE-Zertifikat gesperrt ist. Dann ist bereits im ersten Schleifendurchlauf (i=1) die Disjunktheit von HP_1 und HN_1 gegeben und HLen ist damit gleich eins.

3. Die Sortierung der Elemente im Body ermöglicht es später einem Fachdienst Element effizient zu suchen (binäre Suche). Bei einer HNL hat die Sortierung eine zusätzliche Funktion, wie in Abschnitt 4.8 erläutert.

4. Start-HLen ist CA-spezifisch, d. h. verschiedene CAs eines TSP können unterschiedliche Start-HLen-Werte besitzen. Die Start-HLen hängt hauptsächlich von der Anzahl der von einer CA bereits erstellten und noch zeitlich gültigen EE-Zertifikate ab.

5.1.3 HNL: Body

A_28714 -TSP-X.509 nonQES: Erzeugung des Bodys einer HNL

Ein TSP-X.509 nonQES SMC-B bzw. HBA MUSS bei der Erzeugung des Bodys einer HPL einer CA wie folgt vorgehen:

1. Sei HP die Liste der SHA-256-Hashwerte der von der betrachteten CA ausgestellten (signierten) EE-Zertifikate, die jetzt gültig sind (also nicht gesperrt und zeitlich gültig).

2. Sei HN die Liste der SHA-256-Hashwerte der von der betrachteten CA ausgestellten (signierten) EE-Zertifikate, die jetzt gesperrt und zeitlich noch gültig sind. Nach Konstruktion sind HP und HN disjunkt.

(Hinweis: EE-Zertifikate die zeitlich gültig sind, und vom einem OCSP-Responder weder als "good" noch als "revoked" beantwortet werden, sollen auch Teil der HN sein.)

3. Zur HN werden der Hashwerte 0...0 (256 mal 0-Bit, als Null-Element bezeichnet) und der Hashwert 1...1 (256 mal 1-Bit, als Eins-Element bezeichnet) hinzugefügt. Das Ergebnis wird als HN* bezeichnet, um es eindeutig vom initialen HN unterscheiden zu können.

4. Nun wird analog zu A_28702-* durch schrittweises Erhöhen von i das HLen gesucht bei dem HP_HLen und HN*_HLen erstmalig disjunkt sind.

5. Die Elemente aus HN*_HLen werden als Zahlen (big-endian unsigned) interpretiert, ggf. doppelte Element werden entfernt, und die Elemente (Zahlen) werden aufsteigend sortiert. Sei N die Anzahl der Elemente der so erzeugten Liste.

6. Die Element aus der so erzeugten sortierten Liste werden kontaktiert aufgeschrieben, und das Ergebnis sei C.

7. N wird als 64 Bit unsigned big-endian kodiert und N || C ist der Body der HPL.

[<=]

Bemerkungen zu A_28714-*:

1. Da eine kryptographische sichere Hashfunktion (SHA-256) verwendet wird, kann es keine Zertifikate der CA geben, deren Hashwert das Null-Element oder das Eins-Element sind. Deshalb sind auch HP und HN* disjunkt.

5.1.4 HPL: Trailer (zwei Signaturen)

A_28717 -TSP-X.509 nonQES: Erzeugung des Trailers (zwei Signaturen) bei einer HPL

Ein TSP-X.509 nonQES SMC-B bzw. HBA MUSS bei der Erzeugung des Trailers einer HPL einer CA wie folgt vorgehen:

Zunächst MUSS der TSP über Header || Body und mittels der OCSP-Identität (deren SKI im Header aufgeführt ist) eine auf SHA-256 basierte ECDSA-Signatur erzeugen. Diese Signatur MUSS er gemäß [TR-03111#5.2.1] (plain encoding) kodieren. Das Ergebnis sei Signatur_1, diese ist 64 Byte lang.

Für die zweite Signatur MUSS vom TSP analog über Header || Wurzelement erzeugt werden, wobei das Wurzelement gemäß A_28718-* erzeugt wird. Die Signatur MUSS er gemäß [TR-03111#5.2.1] (plain encoding) kodieren. Das Ergebnis sei Signatur_2.

Der Trailer ist die Konkatenation Signatur_1 || Signatur_2. [**<=**]

A_28718 -Erzeugung des Wurzelements bei einer HPL

Ein Produkttyp, der das Wurzelement für die Erzeugung oder Auswertung der zweiten Signatur der HPL (vgl. A_28717-*) berechnen möchte, MUSS wie folgt vorgehen:

1. Er überführt die Elemente im Body der Reihenfolge nach in eine Liste L. Wegen A_28702-* sind Elemente und damit die eben erzeugte Liste aufsteigend sortiert.
2. Er berechnet von jedem Element von L den Hashwert, indem er vom ersten bis zum letzten Element von L die Liste durchläuft und jeweils das Ergebnis (Hashwert) in eine neue Liste überführt. Die neue Liste sei mit LH benannt. (Hinweis: es gilt $LH[i] = \text{Hash}(\text{Liste}[i])$ für alle aus der Indexmenge der Liste.)
3. Er nimmt sich schrittweise aus LH immer zwei Elemente (Element 1 und 2, Element 3 und 4, usw.), konkateniert diese zwei Elemente und berechnet den Hashwert der Konkatenation. Die Ergebnisse fügt er schrittweise an eine neue Liste Namens LH_neu an. Sollte die Liste eine ungerade Anzahl von Elementen enthalten so wird das letzte (also einzelne) Element unverändert an LH_neu angehängt.
4. LH wird nun mit LH_neu überschrieben und Schritt (3) wiederholt.
5. Schritt 4 wird so lange wiederholt bis LH nur noch ein Element enthält. Dieses Element ist das gesuchte Wurzelement.

[**<=**]

Zur Verdeutlichung wird die Berechnung des Wurzelement nochmal als Pseudo-Code folgend aufgeführt.

```
# Liste_Body enthält aufsteigend sortiert die Elemente von Body
# Annahme Liste_Body ist nichtleer
LH = []
for x in Liste_Body:
    LH.append(SHA_256_Hash(x))
while Länge(LH)>1:
    LH_neu = []
    while Länge(LH)>2:
```

```
914         LH_neu.append(SHA_256_Hash(LH[0] || LH[1]))
915         # || soll hier konkatenieren bedeuten
916         LH = LH[2:] # ersten zwei Element in LH löschen
917     if Länge(LH)>0:
918         # war am Start des Schleifendurchlaufs die Länge von LH ungerade
919         # so wird das letzte Element unverändert übernommen.
920         LH_neu.append(LH[0])
921     LH = LH_neu
922     print("Wurzelelement=", LH[0])
923
```

5.1.5 HNL: Trailer (zwei Signaturen)

A_28722 -TSP-X.509 nonQES: Erzeugung des Trailers (zwei Signaturen) bei einer HNL

Ein TSP-X.509 nonQES SMC-B bzw. HBA MUSS bei der Erzeugung des Trailers einer HNL bis auf eine Ausnahme genau so vorgehen in Kontext HPL gemäß A_28717-*. Die Ausnahme ist, dass bei der Erzeugung des Wurzelelement nach A_28723-* vorgegangen werden MUSS. [≤]

Bei der Berechnung des Wurzelelements zur Berechnung der zweiten Signatur im Trailer einer HNL ist der Beginn der Konstruktion des Hashbaums -- die Blätter -- anders als im HPL-Fall. Direkt benachbarte Elemente im Body werden konkateniert. Um das Beispiel aus Abschnitt 4.1.3 aufzugreifen: Im Body einer HNL sind bspw. vier Elemente $a < b < c < d$. Es werden jeweils die direkten Nachbarn konkateniert und es entsteht eine Liste aus drei Elementen $[a||b, b||c, c||d]$. Diese drei Elemente sind die Blätter des Hashbaums. Mit diesen Blättern wird analog zu A_28718-* das Wurzelelement durch schrittweise Reduktion erzeugt.

A_28723 -Erzeugung des Wurzelelements bei einer HNL

Ein Produkttyp. der das Wurzelelement für die Erzeugung oder Auswertung der zweiten Signatur der HNL (vgl. A_28722-*) berechnen möchte, MUSS wie folgt vorgehen:

1. Er überführt die Elemente im Body der Reihenfolge nach in eine Liste. Wegen A_28702-* sind Elemente und damit die eben erzeugte Liste aufsteigend sortiert.
2. Er iteriert über die Elemente der Liste und konkateniert das aktuelle Element mit dem nachfolgenden Element, solange ein Nachfolgeelement in der Liste existiert. Die schrittweise erzeugten Konkatenationen sind die Element einer neuen Liste, die Grundlage für die Verarbeitung in den folgenden Schritten bildet. Beispiel: $[a, b, c, d]$ wird zu $[a||b, b||c, c||d]$, was die neue Liste ist.
3. Er berechnet von jedem Element der neuen Liste den Hashwert, indem er vom ersten bis zum letzten Element der Liste die Liste durchläuft und jeweils das Ergebnis (Hashwert) in eine neue Liste überführt. Die neue Liste sei mit LH benannt. (Hinweis: es gilt $LH[i] = \text{Hash}(\text{Liste}[i])$ für alle aus der Indexmenge der Liste.)
4. Er nimmt sich schrittweise aus LH immer zwei Elemente (Element 1 und 2, Element 3 und 4, usw.), konkateniert diese zwei Elemente und berechnet den Hashwert der Konkatenation. Die Ergebnisse fügt er schrittweise an eine neue Liste Namens LH_neu an. Sollte die Liste eine ungerade Anzahl von Elementen enthalten so wird das letzte (also einzelne) Element unverändert an LH_neu angehängt.
5. LH wird nun mit LH_neu überschrieben und Schritt (3) wiederholt.
6. Schritt 4 wird so lange wiederholt bis LH nur noch ein Element enthält. Dieses Element ist das gesuchte Wurzelelement.

[≤]

5.1.6 Delta/Update-Dateien

A_28946 -TSP-X.509 nonQES: Delta-Dateien erzeugen und bereitstellen

Ein TSP-X.509 nonQES SMC-B bzw. HBA erzeugt für alle seine CAs jeweils eine HPL und eine HNL alle "period" Sekunden (vgl. A_28952-*) und stellt diese auf der HTTP-Schnittstelle seines OCSP-Responders zur Verfügung (A_23191-*). Er MUSS nach jeder Erzeugung einer weiteren HPL bzw. HNL eine Delta-Datei (definiert in A_28947-*) zwischen vorheriger und nun neu erzeugter HPL bzw. HNL erzeugen. Er KANN auch beliebige weitere Delta-Dateien erzeugen und zur Verfügung am Download-Punkt stellen. [$\leq$]

Hinweis: Die Erzeugung einer Delta-Datei benötigt keine kryptographische Operationen und damit auch keine privaten Schlüssel.

A_28947 -TSP-X.509 nonQES: Definition Delta-Datei zwischen zwei HCSL

Ein Produkttyp, der eine Delta-Datei zwischen zwei HCSL erzeugen möchte, MUSS folgende Vorgaben beachten.

Notation: Eine HCSL ist die Quell-HCSL (Quelle) und die andere die Ziel-HCSL (Ziel).

1. Der Produkttyp MUSS prüfen, ob Quelle und Ziel den gleichen Typ (Positivliste oder Negativlisten) haben und ob beide die gleichen HLen besitzen. Falls nicht so MUSS er die Erstellung der Delta-Datei abbrechen.
2. Er MUSS prüfen, ob die Body von Quelle und Ziel gleich sind, d. h. die Anzahl der Elemente bei Quelle und Ziel die gleiche ist und Quelle und Ziel die gleichen Elementen enthalten. Falls ja ist add_array und rm_array leer und die Schritte 3 und 4 entfallen. (Hinweis: dieser Fall wird häufig auftreten.)
3. Er MUSS die Elemente im Body des Zieles durchlaufen und Elemente, die nicht in der Quelle in einem Array add_array speichern.
4. Er MUSS die Elemente im Body der Quelle durchlaufen und Elemente, die nicht im Ziel enthalten sind in einem Array rm_array speichern.
5. Sei dst_header der Header des Zieles (53 Bytes) und dst_signatures die beiden Signaturen des Zieles (128 Byte).
6. Sei src_time die Zeitinformation ("time") gemäß A_28696 aus der Quelle.

Dann MUSS der Produkttyp folgende Datenstruktur erzeugen:

offset	Länge	Beschreibung
0	1	Erkennungsmerkmal (magic bytes) das Byte "d"
1	53	dst_header, Header der neuen HCSL
54	8	src_time, Erstellungszeit der alten HCSL als 64-Bit Unixzeit kodiert
62	8	nr_add = Größe von add_array, Anzahl der hinzugekommenen gekürzten Hashwerte, als 64-Bit Wert in Network-Byteorder (big-endian)
70	nr_add*hlen	add_array = nr_add mal auf hlen gekürzte Hashwerte, die hinzugefügt werden müssen
70+nr_add*hlen	8	nr_rm = Größe von rm_array, Anzahl der zu

		entfernenden gekürzten Hashwerte, als 64-Bit Wert in Network-Byteorder (big-endian)
70 + nr_add*hlen + 8	nr_rm*hlen	nr_rm mal auf hlen gekürzte Hashwerte, die entfernt werden müssen
70 + nr_add*hlen + 8 + nr_rm*hlen	64	erste ECDSA-Signatur des Ziels
70 + nr_add*hlen + 8 + nr_rm*hlen + 64	64	zweite ECDSA-Signatur des Ziels

Diese Datenstruktur MUSS unter den Dateinamen d-<Type der HCSL>-<src_time>-<dst_time>-<aki> abgespeichert und HCSL-Clients zur Verfügung gestellt werden. (Hinweis: dst_time und aki sind ebenfalls im dst_header enthalten.)【<=】

Hinweise zu A_28947-*:

Eine Delta-Datei bei denen add_array und rm_array leer sind, hat eine Größe von 206 Bytes.

Eine Delta-Datei, bei der zwischen Quelle und Ziel 10 Zertifikate hingekommen sind und eine HLen von 4 besteht, hat die eine Größe von 206+4*10 = 246 Bytes.

5.1.7 Adaptive Fixierung der HLen

A_28706 -TSP-X.509 nonQES: adaptive Fixierung der HLen bei Erzeugung einer HCSL

Ein TSP-X.509 nonQES SMC-B bzw. HBA MUSS pro CA einen Wert Namens Start-HLen pflegen. Initial ist der Wert 1.

Bei der Erzeugung einer neuen HCSL MUSS er gemäß A_28696-* prüfen, ob der HLen-Wert aus der neuen HCSL größer als der für die CA gespeicherte Wert Start-HLen ist. Falls ja MUSS er den Wert Start-HLen für diese CA auf den neuen HLen-Wert setzen. 【<=】

Hinweise zu A_28706-*:

Start-HLen ist CA-spezifisch, d. h. verschiedene CAs eines TSP können unterschiedliche Start-HLen-Werte besitzen. Die Start-HLen hängt hauptsächlich von der Anzahl der von einer CA bereits erstellten und noch zeitlich gültigen EE-Zertifikate ab.

Die Zertifikatssignaturen sind nicht-deterministische/randomisierte Signaturen. Da die Zertifikatssignaturen Teil der in einer HCSL betrachteten Zertifikatshashwerte sind, sind auch die Zertifikatshashwerte zufällig. Deshalb kann es vorkommen, dass die minimal notwendige HLen bei der Erzeugung von HCSL einer CA variiert. Mit der adaptiven Fixierung der Start-HLen nach A_28706-* (vgl. auch Verwendung von Start-HLen in A_28702-* und A_28714-*) wird sichergestellt, dass sich die Delta-Dateien effektiv erzeugen lassen.

A_29028 -TSP-X.509 nonQES: Neustart bei Änderung der HLen

Ein TSP-X.509 nonQES SMC-B bzw. HBA MUSS wenn er nach A_28706-* die HLen auf einen neuen größeren Wert fixieren muss, den Prozess nach für die entsprechende CA neu beginnen. D. h., er muss eine neue hcsl-conf-<aki>.cbor (A_28952-*) erzeugen mit

1025 neuer Start-Zeit "start" und ab dann mit Periode "period" gemäß A_28952-* und
1026 A_23191-* regelmäßig neue HCSL erzeugen. [≤=]

1027 Erläuterung zu A_29028-*:

1028 In seltenen Fällen, wenn bei einer CA sich bspw. die Anzahl der neu erzeugten Zertifikate
1029 plötzlich verdoppelt, kommt es zu einer Erhöhung der HLen und der Fixierung der neuen
1030 HLen (A_28706-*). In diesem Fall bringt die Verwendung der Delta-Dateien keine
1031 Effizienzgewinn. Deshalb startet die CA und auch ein HCSL-Client mit einer neuen HCSL,
1032 die dann die neue HLen besitzt. Anschließend werden vom Client wieder Delta-Dateien
1033 (mit neuem HLen-Wert) zur lokalen Aktualisierung verwendet.

1034 5.2 HCSL-Client

1035 Ein HCSL-Client kennt

- 1036 1. aus den Informationen aus den zu prüfenden EE-Zertifikate selbst,
- 1037 2. aus einer aktuellen TSL, oder
- 1038 3. aus anderen Quellen

1039 die HCSL-Download-Punkte (URLs) der TSPs.

1040 **A_28711 -HCSL-Client: Prüfung der HCSL-Verfügbarkeit alle CAs**

1041 Ein HCSL-Client MUSS für alle für ihn relevanten TI-CAs des Typs TSP-X.509 nonQES SMC-
1042 B bzw. HBA prüfen, ob eine CA HCSL an der HTTP-Schnittstelle der CA zur Verfügung
1043 stellt. (Die Basis-URL der HTTP-Schnittstelle ist dem HCSL-Client aus den Informationen
1044 aus dem zu prüfende EE-Zertifikat bekannt oder aus der TSL).
1045 Für die Prüfung versucht der HCSL-Client die Datei hcs1-config-<aki>.cbor von der HTTP-
1046 Schnittstelle zu beziehen, wobei aki der SKI der CA in Hexadezimalschreibweise (0-9a-f,
1047 vgl. A_28952-*) ist. Ist solch eine Datei vorhanden, so unterstützt diese CA die HCSL-
1048 Bereitstellung. [≤=]

1049 Hinweis: die Frequenz der Prüfung kann bspw. einmal pro Woche sein.

1050 **A_28954 -HCSL-Client: regelmäßige Aktualisierung der lokalen HCSL**

1051 Ein HCSL-Client MUSS die HCSL aus A_28711-* initial beziehen und regelmäßig lokal bei
1052 sich aktualisieren. Aus den Informationen aus hcs1-config-<aki>.cbor (vgl. A_28711-*)
1053 kennt der HCSL-Client pro CA die "start"-Zeit, "period" und "delay" Zeit. Der HCSL-Client
1054 bezieht initial eine HPL oder HNL (vgl. A_23191-*).

1055 Sei t_{aktuell} die aktuelle Systemzeit im Client ist, er berechnet

1056 $k = \lfloor (t_{\text{aktuell}} - t_{\text{start-aki}} - \text{delay}) / p \rfloor$ und $z = t_{\text{start-aki}} + kp$. Er MUSS dann die HPL/NHL-<z>-
1057 <aki> initial beziehen.

1058 Ab dann aktualisiert der HCSL-Client in Abstand von "period"-Sekunden über Delta-
1059 Dateien (vgl. A28946-*) die lokal vorliegende HPL oder HNL für die jeweilige CA. [≤=]

1060 Hinweis:

1061 Direkt nach der lokalen Aktualisierung der HCSL werden die Delta-Dateien im Client
1062 gelöscht. Die Delta-Dateien haben dann ihren Zweck (Delta-Aktualisierung der HCSL)
1063 erfüllt und werden nicht mehr benötigt.

1064 **A_28959 -HCSL-Client: Bezug von Signaturzertifikaten**

1065 Ein HCSL-Client KANN über die HTTP-Schnittstelle einer CA, die HCSL anbietet, die
1066 Signaturzertifikate mit denen sich die Signatur einer HCSL prüfen lassen über den
1067 Dateinamen hcs1-certs.cbor (A_28694-*) abrufen. [≤=]

Oft wird ein HCSL-Client schon über die TSL die Signaturzertifikate lokal vorliegen haben. Falls nicht kann er über hcsl-certs.cbor die CA und die HCSL/OCSP-Signaturzertifikate erhalten.

A_29029 -HCSL-Client: Signaturprüfung bei Neubezug oder Aktualisierung der lokalen HCSL

Ein HCSL-Client MUSS nachdem er eine HCSL neu bezogen hat oder eine lokale HCSL über Delta-Datei aktualisiert hat, die Signatur der neuen HCSL prüfen. Dafür prüft er die erste Signatur aus den Daten bis direkt vor der ersten Signatur (vgl. A_28696-*, A_28714-*, A_28717-*) mittels ECDSA auf Basis von SHA-256. Ergibt die Signaturprüfung kein "VALID", so MUSS die Verwendung der HCSL für Zertifikatsstatusprüfungen unterbleiben. Und ein lokaler Alarm erzeugt werden.

Sollte ein HCSL-Client die Funktionalität der beschränkten Laufzeitumgebung verwenden, so MUSS er ebenfalls die zweite Signatur (A_28717-*, A_28722-*) mittels ECDSA auf Basis von SHA-256 prüfen. Dabei MUSS er wie in A_28718-* und A_28723-* definiert das dafür notwendige Wurzelement erzeugen. Ergibt die Signaturprüfung kein "VALID", so MUSS die Verwendung der HCSL für Zertifikatsstatusprüfungen unterbleiben. [$\leq$]

A_29030 -HCSL-Client: zeitliche Prüfung HCSL

Ein HCSL-Client MUSS nachdem er eine HCSL neu bezogen hat oder eine lokale HCSL über Delta-Datei aktualisiert hat, die Zeitinformation in der neuen/aktualisierten HCSL prüfen (vgl. A_28696-*, "time" Wert im Header). Sollte der Zeitwert (time) älter als eine Stunde sein, so MUSS die Verwendung der HCSL für Zertifikatsstatusprüfungen unterbleiben. Und ein lokaler Alarm erzeugt werden. [$\leq$]

A_29031 -HCSL-Client: lokale Aufbereitung/Verfügarmachung der HCSL-Daten

Ein HCSL-Client MUSS nach erfolgreicher Signaturprüfung (VALID) nach A_29029-* und zeitlicher Prüfung nach A_29030-* die Daten der HCSL für eine spätere Zertifikatsstatusprüfung verfügbar machen. Dafür KANN er den sortierten Body in eine andere Datenstruktur (associative Array/Dictionary) überführen. Der HCSL-Client MUSS sicherstellen, dass keine Zertifikatsstatusdaten, die älter als 60 Minuten sind für die Zertifikatsstatusprüfung verwendet werden. [$\leq$]

Erläuterung:

Die "Verfügarmachung" bedeutet, dass im Moment der EE-Zertifikatsstatusprüfung keine weitere Prüfung der HCSL-Daten (Signatur etc.) mehr notwendig ist, sondern nur noch der sortierte Body (binäre Suche) durchsucht werden muss oder die in ein associative Array/Dictionary überführte Datenstruktur (O(1)-Zugriffskomplexität).

A_28957 -HCSL-Client: EE-Zertifikatsprüfung auf Basis einer HPL

Ein HCSL-Client MUSS bei der Prüfung eines EE-Zertifikats auf Basis von HPL-Daten prüfen, ob

1. für die CA, die das EE-Zertifikat erzeugt hat, lokal HPL-Daten gemäß A_29031-* verfügbar sind,
2. ob der auf HLen (A_29031-*) gekürzte SHA-256 Hashwert als Element in den HPL-Body-Daten enthalten sind (ggf. unter Verwendung eines associative Array/Dictionary),
3. die HPL-Datenbasis nicht älter als 60 Minuten ist.

Liefert einer der Prüfungen ein nichtpositives Ergebnis, so MUSS das EE-Zertifikat als gesperrt bewertet werden. [$\leq$]

A_28958 -HCSL-Client: EE-Zertifikatsprüfung auf Basis einer HNL

Ein HCSL-Client MUSS bei der Prüfung eines EE-Zertifikats auf Basis von HNL-Daten prüfen, ob

1. für die CA, die das EE-Zertifikat erzeugt hat, lokal HNL-Daten gemäß A_29031-* verfügbar sind,
 2. ob der auf HLen (A_29031-*) gekürzte SHA-256 Hashwert nicht als Element in den HNL-Body-Daten enthalten sind (ggf. unter Verwendung eines associative Array/Dictionary),
 3. die HNL-Datenbasis nicht älter als 60 Minuten ist.
- Liefert einer der Prüfungen ein nichtpositives Ergebnis, so MUSS das EE-Zertifikat als gesperrt bewertet werden.【<=】

5.3 HCSL-Client in beschränkter Laufzeitumgebungen (HSM)

Eine wichtige Eigenschaft einer HCSL, ist dass aufgrund der Konstruktion der zweiten Signatur (Merkle-Hashbaum) eine Informationen aus einer HCSL sicher geprüft werden können, ohne dass dem Prüfenden die gesamte HCSL vorliegt. Damit wird es möglich eine HCSL auch innerhalb einer beschränkten Laufzeitumgebung wie bspw. einem HSM zu prüfen.

A_28955 -HCSL-Client in beschränkter Laufzeitumgebungen (HSM) für HPL

Ein HCSL-Client in beschränkter Laufzeitumgebungen (bspw. HSM) MUSS bei der Verwendung einer HCSL in der Variante Positiv-List (HPL) wie folgt vorgehen:

1. Der HCSL-Client erhält von "äußeren" Fachdienst (1) das zu prüfende EE-Zertifikat, (2) einen HCSL-Header (3) einen Pfad durch den Merkle-Hashbaum vom gekürzten Hashwert des EE-Zertifikat bis zum Wurzelements, und (4) eine "zweite Signatur" (vgl. A_28717-*).
2. Der HCSL-Client prüft den Pfad, d. h. prüft, ob der gekürzten Hashwert des EE-Zertifikat wirklich im Merkle-Hashbaum enthalten ist, und dabei berechnet er gleichzeitig das Wurzelement.
3. Der HCSL-Client prüft Zeitinformationen im HCSL-Header: ist die HCSL älter als 60 Minuten ist, dann FAIL.
4. Der HCSL-Client prüft die Signatur aus HCSL-Header || Wurzelement. Falls die Signaturprüfung kein positives Prüfergebnis ergibt, dann FAIL.
5. Falls alle o. g. Prüfungen ein positiven Prüfergebnis ergeben haben, so ist das betrachtete EE-Zertifikat nicht gesperrt (Enthalten in der Positiv-Liste).

【<=】

A_28956 -HCSL-Client in beschränkter Laufzeitumgebungen (HSM) für HNL

Ein HCSL-Client in beschränkter Laufzeitumgebungen (HSM) MUSS bei der Verwendung einer HCSL in der Variante Negativ-List (HNL) bis auf eine Ausnahme wie bei A_28955-* vorgehen.

Die Ausnahme ist bei Schritt 2 in A_28955-* -- hier prüft der HCSL-Client, ob der gekürzte EE-Zertifikatshash (folgend genannt x) eben nicht im Merkle-Hashbaum enthalten ist. Dafür prüft er, ob ein Paar von gekürzten Hashwerten, die direkte Nachbarn sind, im Merkle-Hashbaum existiert bei denen x als Zahlenwert zwischen den Paarelementen liegt. Falls dies zutrifft, also x ist nicht Teil des Merkle-Hashbaums. Deswegen ist der zu prüfenden EE-Zertifikatshash eben nicht Teil der Negativ-Liste (HNL). Deswegen ist das EE-Zertifikat nicht gesperrt.【<=】

Im welchen Format (Kodierung) der äußere Teil eines Fachdienstes die Informationen aus A_28955-* oder A_28956-* der beschränkten Laufzeitumgebung präsentiert, bleibt ihm überlassen.

1162 **5.4 Sicherheit**

1163 Es werden durch die HCSL keine weitergehenden Sicherheitsanforderungen an einen TSP
1164 gestellt, als die sowieso schon für ihn gelten.

1165 **5.5 Betrieb**

1166 Es werden durch die HCSL keine weitergehenden betrieblichen Anforderungen
1167 (Verfügbarkeit, Anbindung, Service-Zeiten etc.) an einen TSP gestellt, als die sowieso
1168 schon für ihn gelten.

6 Merkle-Hashbäume (informativ)

Hash-Bäume (auch Merkle-Hashbäume nach Ralph Merkle (1979) genannt) sind eine Datenstruktur in der Daten als Hashwerte von beliebigen Daten als Blätter eines Binärbaums angeordnet werden. Dabei werden zwei Nachbarelemente per konkateniert und per Hashwert in ein neues Element der nächste Baumebene überführt. Solange wird dies durchgeführt bis nur noch ein Element (das "Wurzelelement") als letztes Element übrig bleibt.

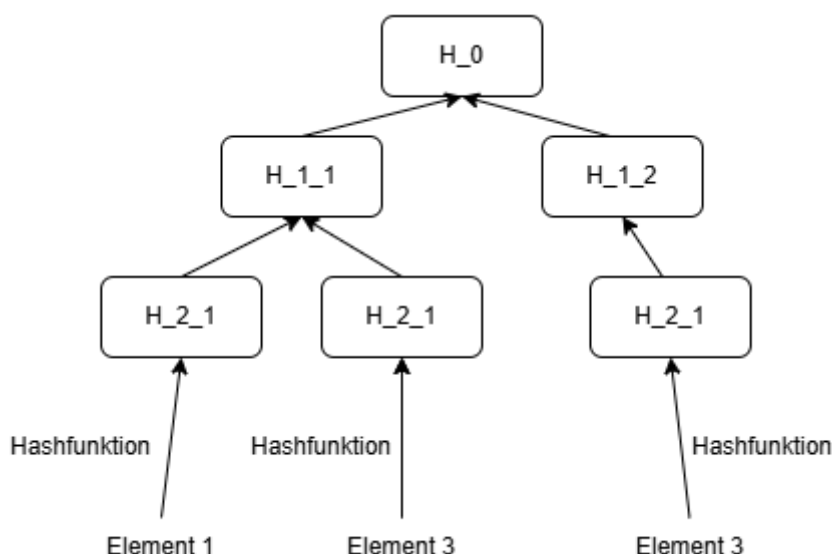


Abbildung 7: Beispiel Hashbaum über drei Elemente (beliebige Daten)

Mittels Hash-Bäumen kann man effizient die Integrität von großen Datenmengen festgestellt werden. Hash-Bäume werden u. a. bei

1. Datenbanksystemen
 2. Dateisystemen (ZFS)
 3. Peer-to-Peer-Anwendungen
 4. Blockchain-Systemen
 5. Source-Code-Revisionssysteme (git), oder
 6. HTTP-Caches
- verwendet.

7 Beispielimplementierung

1188

1189

1190

Die gematik stellt auf Anfrage Beispiel-Code, sowohl für die TSP-Seite als auch für die HCSL-Client-Seite, bereit.

8 Anhang A - Verzeichnisse

8.1 Abkürzungen

Kürzel	Erläuterung
AKI	Ein "Authorization Key Identifier" (AKI) ist ein innerhalb einer PKI eindeutiger Identifier (i. d. R. 160 Bit lang). Der AKI identifiziert den Aussteller eines Zertifikats bzw. einer Identität.
EE-Zertifikat	Ein End-Entity-Zertifikat ist ein Zertifikat einer natürlichen oder juristischen Person (Versicherter, Arztpraxis, Krankenhaus), die keine CA ist.
HCSL	Hash based Certificate Status List, wie in gemF_HCSL definiert
HNL	HCSL in der Variante "Hash Negativ List"
HPL	HCSL in der Variante "Hash Positive List"
MiB	1 MiB = 1024*1024 Bytes = 1048576 Bytes Im Gegensatz dazu 1 MB = 1000*1000 Bytes = 1_000_000 Bytes. Es gilt 1 MB < 1 MiB.
SKI	Ein "Subject Key Identifier" (SKI) ist ein innerhalb einer PKI eindeutiger Identifier (i. d. R. 160 Bit lang). Der SKI identifiziert eindeutig ein Zertifikats bzw. eine kryptographische Identität.

8.2 Abbildungsverzeichnis

Abbildung 1: Grundsätzlicher Aufbau einer HCSL.....	11
Abbildung 2: Beispiel-Berechnung von H_0 für die zweite Signatur in einer HPL.....	13
Abbildung 3: Beispiel-Berechnung von H_0 bei einer HNL für die zweite Signatur.....	14
Abbildung 4: Erzeugung der HCSL+Delta-Dateien mit Periode p im TSP.....	17
Abbildung 5: Pfad von einem Blatt-Element zur Wurzel im Hashbaum.....	20
Abbildung 6: Pfad von einem Blatt-Element zur Wurzel im Hashbaum.....	21
Abbildung 7: Beispiel Hashbaum über drei Elemente (beliebige Daten).....	37

8.3 Tabellenverzeichnis

Tabelle 1: Zusammenfassung Aufbau HCSL.....22

Tabelle 2: Zusammenfassung Aufbau Delta-Datei.....23

8.4 Referenzierte Dokumente

8.4.1 Dokumente der gematik

Die nachfolgende Tabelle enthält die Bezeichnung der in dem vorliegenden Dokument referenzierten Dokumente der gematik zur Telematikinfrastruktur.

[Quelle]	Herausgeber: Titel
[gemGlossar]	gematik: Glossar der Telematikinfrastruktur
[gemKPT_PKI_TIP]	gematik: Konzept PKI der TI-Plattform
[gemSpec_PKI]	gematik: Übergreifende Spezifikation Spezifikation PKI

8.4.2 Weitere Dokumente

[Quelle]	Herausgeber (Erscheinungsdatum): Titel
[CRLITE]	Larisch, J., Choffnes, D.R., Levin, D., Maggs, B.M., Mislove, A., Wilson, C.: CRLite: A scalable system for pushing all TLS revocations to all browsers. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017. pp. 539-556. https://jameslarisch.com/pdf/crlite.pdf
[FIPS-180]	Federal Information, Processing Standards Publication 180-4, Secure Hash Standard (SHS), March 2012 http://csrc.nist.gov/publications/fips/fips180-4/fips180-4.pdf
[RFC-5280]	RFC 5280 (Mai 2008): Internet X.509 Public Key Infrastructure – Certificate and Certificate Revocation List (CRL) Profile http://tools.ietf.org/html/rfc5280
[TR-03111]	BSI (2012): Elliptic Curve Cryptography, Version 2.10, 01.06.2018,

	https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr03111/TR-03111_node.html
	Federal Information, Processing Standards Publication 180-4, Secure Hash Standard (SHS), March 2012 http://csrc.nist.gov/publications/fips/fips180-4/fips180-4.pdf

1216