Design und Implementierung eines Localhost Signaturgateways*

David Derler[1] · Christof Rath[1] · Moritz Horsch[2] · Tobias Wich[3]

Technische Universität Graz[1]
{david.derler | christof.rath}@iaik.tugraz.at

Technische Universität Darmstadt[2]
horsch@cdc.informatik.tu-darmstadt.de

ecsec GmbH[3]
tobias.wich@ecsec.de

Zusammenfassung

Die elektronische Signatur gilt als Schlüsselkomponente im elektronischen Geschäftsverkehr. Allerdings setzt die Vielzahl an Signaturkarten und -applikationen mit unterschiedlichen Benutzeroberflächen und Programmabläufen große Hürden für eine breite Benutzerakzeptanz. Für neues Vertrauen und Akzeptanz auf Nutzer- und Serviceseite, bedarf es insbesondere einer flexiblen, interoperablen, benutzerfreundlichen Open Source Signaturanwendung. Wir präsentieren eine universelle Signaturanwendung mit modularem Aufbau für eine breite Unterstützung von beliebigen Signaturkarten, -verfahren, und -anwendungen und zeigen exemplarisch die Integration der österreichischen Handy-Signatur [ATru].

1 Einleitung

Obwohl die elektronische Signatur als Schlüsselkomponente für den elektronischen Geschäftsverkehr gilt und ein breites Anwendungsspektrum bietet, findet sie in der Praxis nur eine geringe Verbreitung. Aus Sicht des Benutzers stellt in erster Linie die große Vielzahl an unterschiedlichen Signaturkarten und -applikationen eine Herausforderung dar. Insbesondere stellt die Installation mehrerer Signaturapplikationen, deren unterschiedliche Handhabung und die verschiedenen Programmabläufe eine große Hürde für eine breite Benutzerakzeptanz dar. Neben dem Wunsch nach einer einfach zu bedienenden Anwendung, die ein breites Spektrum von Signaturszenarien unterstützt, spielt besonders das Vertrauen in die Signaturapplikation eine wichtige Rolle. Für die Steigerung von Vertrauen, Transparenz und Akzeptanz auf Nutzer- und Serviceseite, bedarf es daher einer flexiblen, interoperablen, benutzerfreundlichen und als Open Source verfügbaren Signaturanwendung. Neue Impulse und eine Belebung des Europäischen Signaturmarktes kann hier insbesondere die eIDAS-Verordnung [Euro14] der Europäischen Kommission bieten.

* Die Autoren wurden von der Europäischen Kommision über das Projekt FP7-FutureID (EU-Fördervertrag Nummer 318424) unterstützt. Dieser Beitrag erweitert die in [HDRH+14] präsentierten Ergebnisse.

1.1 Existierende Ansätze

Es existieren bereits verschiedene Signaturapplikationen, die einerseits proprietäre, kartenspezifische Lösungen aber andererseits zum Teil auch universelle Frameworks bereitstellen. Dazu zählen die österreichische MOCCA Plattform [MOC], die belgischen Signaturkomponenten [Corn], die portugiesische Cartão de Cidadão Middleware [Midd], die OpenSC Middleware [Palj], der Sirius Signaturserver [KuVe] und das Digital Signature Service Projekt [Nara]. Die belgischen Signaturkomponenten und der Sirius Signaturserver setzen, so wie unserer Ansatz, auf den OASIS Digital Signature Services (DSS) [OASI] Standard.

1.2 Unser Beitrag

Eine breite Benutzerakzeptanz der elektronischen Signatur kann nur erreicht werden, wenn der Aufwand und die Lasten für den Benutzer massiv reduziert werden und eine fundierte Vertrauensbasis geschaffen wird. Dies heißt unter anderem die Nutzung unterschiedlicher Signaturkarten und -anwendungen zu vereinfachen. Wir verfolgen den Ansatz der Entwicklung einer universellen, modularen und quellenoffenen Signaturapplikation auf Basis des FutureID Clients [FID]. Damit setzen wir auf eine bestehende und etablierte Applikation auf, die unter anderem durch die ISO 24727 [ISO/08] und die eCard-API [BSI12] Architektur ein hervorragendes Framework für Modularität und eine breite Unterstützung für beliebige Geräte zur Signaturerstellung, wie beispielsweise Smartcards, bietet. Um die Stärken diese Ansatzes zu zeigen, präsentieren wir in diesem Beitrag die Integration der österreichischen Handy-Signatur [ATru].

2 Grundlagen

In diesem Abschnitt diskutieren wir grundlegende Konzepte die für diesen Artikel von Bedeutung sind.

2.1 Fortgeschrittene/Qualifizierte Elektronische Signaturen

Die europäische Signaturrichtlinie [Euro00] definiert elektronische Signaturen allgemein als: „Daten in elektronischer Form, die anderen elektronischen Daten beigefügt oder logisch mit ihnen verknüpft sind und die zur Authentifizierung dienen“.

Eine *fortgeschrittene elektronische Signatur* nach [Euro00] erfordert darüber hinaus, dass „sie ausschließlich dem Unterzeichner zugeordnet ist“ und „die Identifizierung desselben ermöglicht“. Weiters ist es erforderlich, dass sie mit Mitteln erstellt wurde, „die der Unterzeichner unter seiner alleinigen Kontrolle halten kann“ und „sie so mit den Daten verknüpft ist, auf die sie sich bezieht, dass eine nachträgliche Veränderung der Daten erkannt werden kann“.

Eine *qualifizierte elektronische Signatur* ist eine fortgeschrittene Signatur, die auf einem qualifizierten Zertifikat nach [Euro00] beruht und von einer sicheren Signaturerstellungseinheit (SSCD) erstellt wird. Eine erforderliche Eigenschaft einer SSCD ist die Möglichkeit der Darstellung der zu signierenden Daten. Laut österreichischem Signaturgesetz [SigG10] ist eine qualifizierte elektronische Signatur der handschriftlichen Unterschrift gleichgestellt.

2.2 Österreichische Handy-Signatur

Die österreichische Handy-Signatur [ATru] ist eine server-basierte Signaturlösung, die die Voraussetzungen einer qualifizierten Signatur gemäß der europäischen Signaturrichtlinie [Euro00]

erfüllt. Der private Schlüssel des Benutzers wird serverseitig in einem Hardware Security Module (HSM) erzeugt und abgelegt. Durch entsprechende Verschlüsselung wird sichergestellt, dass sich der private Schüssel einzig unter der Kontrolle des Benutzers befindet und dass kein Unbefugter die Signaturerstellung im Namen des Inhabers auslösen kann. Der Zugriff auf den Schlüssel erfolgt mittels Zwei-Faktor-Authentifizierung, wobei das Mobiltelefon den Faktor Besitz erfüllt. Dabei werden Einmalpasswörter (TAN) via SMS auf ein, zuvor dem Benutzer zugeordnetes, Mobiltelefon gesendet. Als Interface in Richtung Anwendung dient, wie bei allen österreichischen Bürgerkarten-Implementierungen, der sogenannte Security-Layer [SEC13], ein XML-basiertes Anfrage-Antwort-Protokoll.

Grundsätzlich gilt folgender Ablauf für die Signaturerstellung: Die Security-Layer-Anfrage wird mittels HTTP POST, von einem Client an den Handy-Signaturserver gesendet. Dieser antwortet mit einem Formular, welches für die Übermittlung von Mobilnummer und Passwort des Benutzers vorgesehen ist. Sind die übermittelten Daten korrekt, wird ein TAN an das Mobiltelefon gesendet und erneut ein Formular an den Client zurückgeliefert, welches für die Übermittlung des TAN vorgesehen ist. Nach erfolgreicher Verifikation des TAN, wird die ursprüngliche Anfrage bearbeitet und das Ergebnis an das gewünschte Ziel gesendet.

2.3 OASIS DSS

OASIS DSS [OASI] ist ein Standard, der Anfrage- und Antwortformate für die Erstellung und Verifikation von Signaturen in verschiedenen Formaten (z.B. XML) definiert. Durch die Verwendung eines solchen Formates ist es möglich eine aufrufende Applikation vollkommen von der Signaturerstellung abzukapseln, was speziell im Kontext dieses Artikels die gewünschte Flexibilität bietet.

2.4 Java Cryptography Architecture

Die Java Cryptography Architecture (JCA) stellt eine einfach zu verwendende Programmierschnittstelle (API) zum Zugriff auf kryptographische Primitive bereit. Die Implementierungen dieser Primitive werden dabei in sogenannten Providern gekapselt und können somit einfach ausgetauscht werden, während die einheitliche API Interoperabilität sicherstellt. Ein Beispiel eines solchen Providers ist [BrHH13].

3 FutureID Client – Architektur

Im Rahmen des FutureID Projektes [FID] wird auf Basis der freien Open eCard App [OeC] der FutureID Client entwickelt. Er stellt eine Plattform für die Nutzung von beliebigen Chipkarten und anderen Geräten zur Authentisierung und Signaturerstellung bereit und fungiert damit als Schnittstelle zwischen Benutzer und von Diensteanbietern angebotenen Anwendungen. Die Unterstützung diverser Chipkarten und Geräte wird durch die Verwendung der ISO/IEC 24727 [ISO/08] Schnittstellen sowie einer modularen und erweiterbaren Architektur [WHPS+13] möglich. Des weiteren ist es möglich plattformabhängige Module auszutauschen, um z.B. eine mobile Variante der Applikation zu erstellen.

Abbildung 1 zeigt die Kernkomponenten des FutureID Client. Im Folgenden werden die für diesen Artikel relevanten Komponenten kurz erläutert, während wir den Leser für weitere Details auf [WHPS+13] verweisen. Im Kontext dieses Artikels besonders hervorzuheben ist die Add-on Komponente, welche es erlaubt den Client um zusätzliche Applikationslogik zu erweitern, die dann mittels Bindings Daten mit Applikationen von Diensteanbieter austauschen kann.

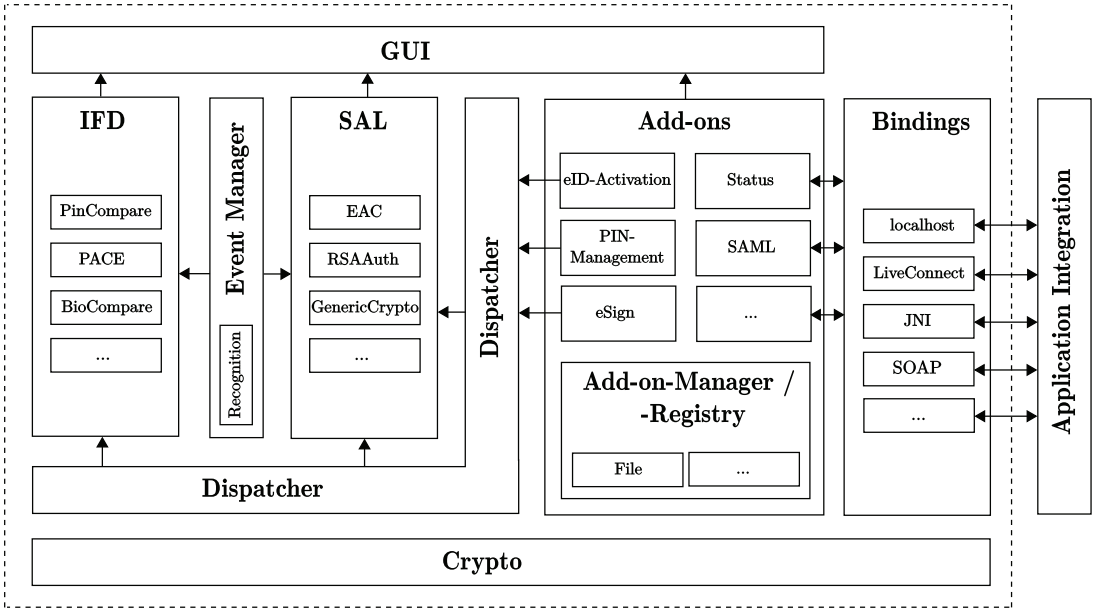

Abb. 1: Architektur des FutureID Client

3.1 IFD

Der IFD stellt eine nachrichten-orientierte Schnittstelle nach ISO/IEC 24727-4 [ISO/08] zur Verfügung um auf ISO/IEC 7816-4 [ISO/04b] basierte Geräte wie z.B. Chipkarten zuzugreifen. Die Interaktion mit den Geräten ist dabei i.d.R. durch weitere Schnittstellen geregelt. Auf PC Plattformen ist dies meist die PC/SC Schnittstelle [PC/S12], während auf Mobilgeräten NFC [ISO/04a, ISO/05] und die Open Mobile API [SIMa12] denkbar sind. Neben der standardisierten Schnittstelle enthält der IFD Erweiterungen um sichere Kanäle mit dem Gerät aufzubauen.

3.2 SAL

Ähnlich wie der IFD stellt der SAL eine nachrichten-orientierte Schnittstelle nach ISO/IEC 24727-3 [ISO/08] bereit. Logische Abstraktionen und die Möglichkeit Protokolle zu ergänzen ermöglichen eine generische Nutzung der Geräte. Ohne die Schnittstelle des SAL zu verändern können so neue Gerätetypen und Protokolle unterstützt werden.

3.3 Dispatcher

Die Zustellung der Nachrichten für SAL und IFD übernimmt der Dispatcher. Er erkennt anhand des Nachrichtentyps welche Komponente das richtige Ziel darstellt. Dies reduziert die Komplexität für Komponenten die Nachrichten von anderen Anwendungen weiterleiten müssen. Zusatzfunktionen wie Zugriffsbeschränkungen auf bestimmte Funktionen und Nachrichtentracking sind mit dem Dispatcher leicht realisierbar.

3.4 Add-ons

Durch Add-ons kann die Funktionalität der Applikation erweitert werden. Sie können dynamisch geladen werden und sind durch eine Sandbox beschränkt um die Auswirkungen bösartiger Add-ons auf die restliche Umgebung zu minimieren.

3.5 Bindings

Add-ons interagieren über Bindings mit Anwendungen von Diensteanbietern. Die Komponenten implementieren jeweils einen Transportmechanismus wie beispielsweise HTTP oder SOAP und reichen die empfangenen Nachrichten in einer abstrahierten Form an die Add-ons weiter.

4 FutureID Client – Add-on Framework

Dieser Abschnitt beschreibt die Erweiterungsmöglichkeiten des FutureID Client mit Add-ons. Unterschiedliche Add-on Typen ermöglichen die Erweiterung auf verschiedenen Ebenen. Prinzipiell gliedern sich Add-ons in Protokoll bzw. Application *Plug-ins* und Application *Extensions*. Plug-ins reagieren dabei auf an sie adressierte Daten, wohingegen Extensions explizit, i.d.R. vom Benutzer, ausgeführt werden.

Protokoll Plug-ins können für IFD und SAL entwickelt werden, um beispielsweise sichere Kanäle, Authentisierungs- und Anwendungsprotokolle einzubringen. Das in diesem Beitrag diskutierte eSign Plug-in fällt in die Kategorie der Application Plug-ins. Application Extensions sind für eigenständige Funktionalität ohne Eingaben von externen Anwendungen zuständig, wie z.B. das PIN Management von Chipkarten.

Wie bereits im vorherigen Abschnitt erwähnt, unterstützt das Add-on Framework das dynamische Laden von Add-ons. Diese Aufgabe übernimmt der in Abbildung 1 gezeigte Add-on Manager. Die zur Verfügung stehenden Add-ons werden von der Add-on Registry verwaltet, welcher die Details des jeweiligen Add-ons in Form einer Konfigurationsdatei zur Verfügung stehen. Beschrieben werden dort Metadaten, Konfigurationsoptionen, Nachrichtenformate in abstrakter Form und Einstiegspunkte für den Aufruf des Add-ons. Wird ein bestimmtes Add-on benötigt, kann die Registry konsultiert werden ob ein passendes Add-on vorhanden ist. Nachfolgend kann dieses von der Registry bezogen werden. Dabei liegt es an der konkreten Registry Implementierung von welcher Quelle das Add-on geladen wird. Neben der naheliegenden dateibasierten Verwaltung ist beispielsweise eine App-Store basierte Registry denkbar.

Die Kommunikation mit externen Anwendungen wird über die sogenannten Bindings abgewickelt. Bindings übersetzen ein wohldefiniertes Transportprotokoll in eine abstrakte Form, sodass Plug-ins nicht an ein bestimmtes Transportprotokoll gebunden sind. Angelehnt an die Struktur von HTTP Nachrichten bestehen eingehende Nachrichten aus Parametern, einem Body, sowie Anhängen. Ausgehende Nachrichten beinhalten zusätzlich einen wohldefinierten Status Code.

5 FutureID Client – eSign Plug-in

Das eSign Plug-in setzt das Konzept eines Signaturgateway um, welches auf OASIS DSS [OASI] Anfragen antwortet und somit den Prozess der Signaturerstellung vollkommen von der aufrufenden Applikation abkapselt. Dabei wird das Add-on Framework des FutureID Client genutzt um (1) die Kommunikation mit dem eSign Plug-in über das oben erwähnte Binding Framework zu abstrahieren und (2) die Signaturerstellung mit beliebigen, vom FutureID Client unterstützten, Signaturerstellungsgeräten zu ermöglichen. Abbildung 2 gibt einen Überblick über die Architektur des eSign Plug-in und stellt dessen Interfaces im Kontext des FutureID Client dar.

Durch die Kapselung der eSign Services in ein Plug-in des FutureID Client ist es möglich

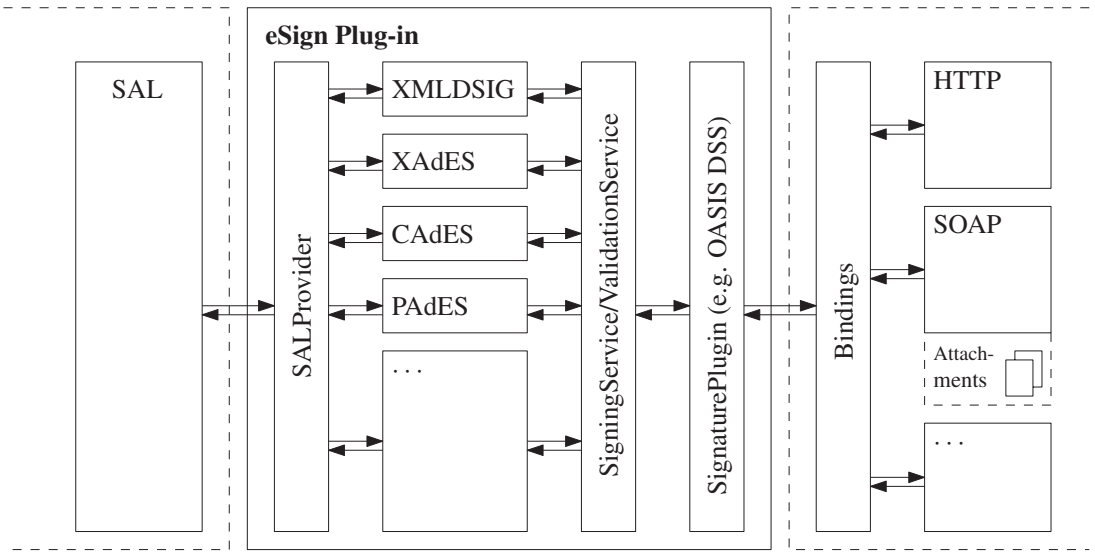

Abb. 2: Architektur des eSign Plug-in

beliebige Bindings zu verwenden ohne die eSign Services selbst anzupassen. Des weiteren wird auch die Berechnung des Signaturwertes über ein zentrales Interface abgewickelt (siehe SAL bzw. SALProvider in Abbildung 2), was es einfach ermöglicht alternative Wege der Berechnung des Signaturwertes zur Verfügung zu stellen. So können alternativ zum SALProvider beliebige andere Java Cryptographic Provider implementiert werden.

Intern kann die Signaturerstellung mit den eSign Services wie folgt dargestellt werden: Die eingehende OASIS DSS [OASI] Anfrage wird über einen sogenannten `RequestTransformer` in ein internes Containerformat transformiert, was prinzipiell eine Austauschbarkeit des Anfrageformates erleichtert. Danach wird die Anfrage an einen `SigningService`- bzw. `ValidationService`-Proxy weitergeleitet der die Anfrage in weiterer Folge an eine konkrete `Signing-/ValidationService` Implementierung delegiert. Dadurch können sowohl `Signing`- als auch `ValidationService` beliebig ausgetauscht werden. `Signing`- bzw. `ValidationServices` greifen wiederum auf `SignatureFormat` Implementierungen (XAdES [XML10], CAdES [CAdES11], PAdES [PDF10]) zurück, wodurch die einfache Integration von beliebigen Signaturformaten sichergestellt wird. Sobald die Signatur erstellt wurde, wird das Resultat im internen Containerformat bis zum `ResponseTransformer` durchgereicht, welcher dann eine Antwort in beliebigen Formaten erstellt (hier OASIS DSS). In diesem Schritt können auch Transformationen wie die Kapselung aller relevanten Daten in ein Archivformat wie z.B. ASiC [ASC12] durchgeführt werden. Im Fehlerfall wird die Antwort von einem sogenannten `ExceptionTransformer` erstellt, der interne Fehlermeldungen in das jeweilige Antwortformat transformiert.

5.1 Abstraktion der Signaturerstellung

Wie bereits zuvor erwähnt wird die Signaturerstellung intern über ein zentrales Interface abgewickelt. Damit ist es einerseits möglich die Berechnung des Signaturwertes über den SAL zu abstrahieren, andererseits können dadurch auch beliebige andere Wege der Signaturerstellung abgebildet werden. Abbildung 3 zeigt verschiedene Szenarien der Signaturerstellung die wir im Folgenden diskutieren werden.

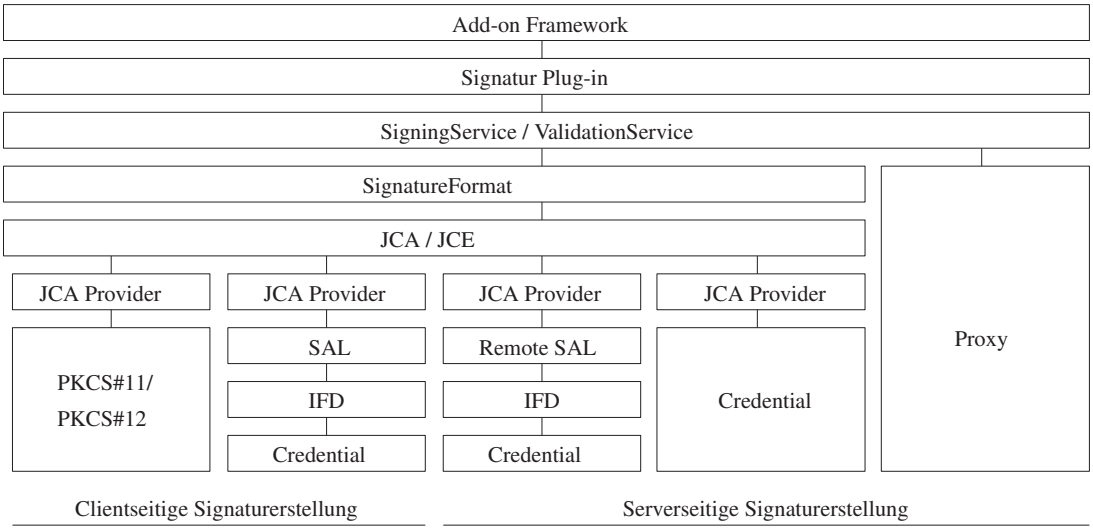

Abb. 3: Abstraktion der Signaturerstellung

Prinzipiell kann man zwischen client- und serverseitiger Signaturerstellung differenzieren, wobei in beiden Fällen wieder mehrere mögliche Szenarien unterschieden werden können. Wir konzentrieren uns im folgenden auf die lokale Signaturerstellung mit Berechnung des Signaturwertes über den SAL bzw. auf die serverseitige Signaturerstellung über die österreichische Handy-Signatur.

Wie in Abbildung 3 angedeutet, hat man die Möglichkeit die Signaturerstellung auf verschiedenen Ebenen zu abstrahieren. Einerseits könnte man z.B. die gesamte Signaturerstellung, bzw. -verifikation an ein Webservice delegieren (Proxy). Dabei würde die Client Implementierung nur das Weiterleiten der Anfragen/Antworten übernehmen. Andererseits könnte man einen Java Cryptographic Provider (JCP) zur Verfügung stellen, der nur die Berechnung des Signaturwertes übernimmt, während die Erstellung des Containers im jeweiligen Signaturformat von der `SigningService` bzw. `SignatureFormat` Implementierung übernommen wird. Die Berechnung des Signaturwertes mittels Provider wird dabei in zwei abstrakte Schritte geteilt:

Schlüsselmaterial abrufen: Dieser Schritt wird durch einen sogenannten `ProviderInitializationProxy` gekapselt, welcher die prinzipielle Aufgabe hat sowohl den privaten als auch den öffentlichen Signaturschlüssel in einer implementierungsunabhängigen Art und Weise zur Verfügung zu stellen. Dabei können verschiedene Szenarien unterstützt werden: Im einfachsten Fall wird direkt ein Schlüsselpaar erzeugt und zur Verfügung gestellt. Allerdings könnte man den öffentlichen Signaturschlüssel eines so erzeugten Schlüsselpaares in diesem Schritt auch direkt von einem Zertifizierungsanbieter zertifizieren lassen. Des weiteren können verschiedenste Signaturerstellungsgeräte wie z.B. Smartcards unterstützt werden, wobei in diesen Fällen lediglich eine Referenz auf den privaten Schlüssel bereitgestellt wird.

Signaturwert berechnen: In diesem Schritt wird unter Verwendung der zuvor abgerufenen Schlüssel ein Signaturwert berechnet. Dies passiert über eine Implementierung des von der JCA zur Verfügung gestellten `Signature` Interface. Dabei ist zu beachten, dass die `Signature` Implementierung auch mit den verschiedensten vom jeweiligen `ProviderInitializationProxy` erzeugten Schlüsseln kompatibel ist.

Clientseitige Signaturerstellung (SAL Provider)

Bei der clientseitigen Signaturerstellung mit Berechnung des Signaturwertes über den SAL Provider wird beim Abrufen des Schlüsselmaterials der öffentliche Signaturschlüssel vom jeweiligen Signaturerstellungsgerät geladen. Viele Signaturerstellungsgeräte verlangen dabei eine Authentifizierung (z.B. PIN) was ebenfalls auf abstrakte Art und Weise vom SAL unterstützt wird. Anstatt den privaten Signaturschlüssel direkt zu laden (in den meisten Fällen nicht möglich) wird hier nur eine Referenz auf diesen zurück geliefert. Weiters wird eine Authentifizierung durchgeführt, falls dies nicht beim Laden des öffentlichen Schlüssels erfolgte. Die Implementierung des JCA `Signature` Interface besteht letztlich nur mehr aus dem Erstellen einer Signaturanfrage für den SAL und dem Weiterreichen der Antwort. Die eben beschriebenen Schritte zur Berechnung des Signaturwertes werden dann von beliebigen `SignatureFormat` Implementierungen verwendet.

Serverseitige Signaturerstellung (Österreichische Handy-Signatur)

Wie in Abschnitt 2 erwähnt, ist die österreichische Handy-Signatur in der Lage, qualifizierte elektronische Signaturen zu erstellen. Daher muss die Möglichkeit gegeben sein, das zu signierende Dokument in der SSCD für den Unterzeichner darzustellen, was in Folge bedeutet, dass das Dokument vollständig auf den Server übertragen und die Signatur auch dort erzeugt werden muss. Die clientseitige Erstellung des Signaturcontainers, z.B. XAdES [XML10], und die Abstraktion der Berechnung des Signaturwertes mittels eines JCA Providers ist folglich nicht möglich. Daher wurde der Proxy-Ansatz gewählt und die Signaturerstellung vollständig an den Handy-Signaturserver delegiert. Die Hauptaufgabe der Proxy Implementierung ist dabei die Transformation der OASIS DSS Anfragen in entsprechende Security-Layer Anfragen wie sie für die österreichische Bürgerkarte benötigt werden. In diesem Zusammenhang sind die Befehle `CreateXMLSignatureRequest` zum Erstellen von XAdES Signaturen und `CreateCMSSignatureRequest` für CAdES und PAdES Signaturen relevant.

Die Interaktion mit dem Unterzeichner, also Eingabe von Mobilnummer und Passwort und danach TAN, die im Allgemeinen über einen Browser erfolgt, wird über den FutureID Client, d.h. die Signatur-Proxy Komponente, abgewickelt. Damit wird erreicht, dass das bestehende Look and Feel des FutureID Clients erhalten bleibt, selbst wenn die eigentliche Signaturerstellung an ein externes Signaturservice weitergereicht wird.

5.2 Abstraktion der Signaturverifikation

Grundsätzlich kann die Signaturverifikation durch belibige `ValidationService` Implementierungen durchgeführt werden. In unserer Implementierung verfolgen wir den Proxyansatz, bei dem die Verifikationsanfrage an ein Validierungsservice weitergeleitet wird. Dies hat den Vorteil, dass beispielsweise Entscheidungen über den Vertrauensstatus von Zertifikaten nicht vom Nutzer selbst getroffen werden müssen und somit die Usability erhöht wird. Durch die flexible Architektur gibt es allerdings auch die Möglichkeit ein solches `ValidationService` lokal zu betreiben und somit (unter anderem) Entscheidungen über den Vertrauensstatus von Zertifikaten selbst zu treffen.

Vom Blickwinkel der Implementierung wird für die Verifikation des Signaturwertes eine Implementierung der Java Cryptography Extensions wie z.B. [BrHH13] verwendet. Daneben führt das Validierungsservice eine Signaturvalidierung nach [SVPP12] durch, was prinzipiell auch

Signaturen abdeckt die mit konventionellen Methoden nicht mehr validiert werden könnten (Langzeitsignaturvalidierung).

Nachdem der Fokus in diesem Artikel auf der Signaturerstellung liegt, soll hier nicht näher auf die Verifikation eingegangen werden. Für weitere Details verweisen wir den Leser an dieser Stelle auf [FID].

6 Zusammenfassung

In diesem Artikel wird ein flexibles, auf dem FutureID Client aufbauendes, Signaturgateway vorgestellt. Durch den modularen Aufbau, sowie das standardisierte Anfrage und Antwortformat wird die einfache Erweiterbarkeit, sowie die einfache Integration in bestehende Applikationen sichergestellt. Vorteil für bestehende Applikationen ist dabei die vollständige Abstraktion der Signaturerstellung, was solche Applikationen unabhängig von Änderungen an Signaturkomponenten bzw. -formaten und Regulierungen macht. Daneben bietet der FutureID Client die Möglichkeit auf abstrakte Art und Weise auf beliebige Signaturerstellungsgeräte zuzugreifen, was es, im Gegensatz zu existierenden Ansätzen, ermöglicht eine einzelne Signaturapplikation für eine Vielzahl von Signaturerstellungsgeräten zur Verfügung zu stellen. Durch die geplante Unterstützung einer möglichst großen Zahl europäischer Signaturerstellungsgeräte im Rahmen von FutureID bietet ein solcher Client die ideale Basis für eine breite Nutzerakzeptanz. Ein weiterer wichtiger Faktor für die Nutzerakzeptanz ist das einheitliche Look and Feel der Signaturerstellung, unabhängig vom verwendeten Signaturerstellungsgerät, wie am Beispiel der österreichischen Handy-Signatur gezeigt wurde.

Literatur

[ASC12] Associated Signature Containers (ASiC), v1.2.1. ETSI TS 102 918 (2012).

[ATru] A-Trust: Handy-Signatur. http://www.handy-signatur.at.

[BrHH13] D. Bratko, C. Hanser, B. Haas: IAIK-JCE 5.2 SDK (2013).

[BSI12] BSI: eCard-API-Framework. Technische Richtlinie BSI-TR-03112, Version 1.1.2, Part 1-7 (2012).

[CAdES11] CMS Advanced Electronic Signatures (CAdES), v1.8.3. ETSI TS 101 733 (2011).

[Corn] F. Cornelis: eID Digital Signature Service Project. https://code.google.com/p/eid-dss/.

[Euro00] Europäische Kommission: Richtlinie 1999/93/EG des Europäischen Parlaments und des Rates vom 13. Dezember 1999 über gemeinschaftliche Rahmenbedingungen für elektronische Signaturen (2000).

[Euro14] Europäische Kommission: Verordnung des europäischen Parlaments und des Rates über die elektronische Identifizierung und Vertrauensdienste für elektronische Transaktionen im Binnenmarkt (2014).

[FID] FutureID - Shaping the future of electronic identity. http://www.futureid.eu.

[HDRH+14] M. Horsch, D. Derler, C. Rath, H.-M. Haase, T. Wich: Open Source für europäische Signaturen. In: *Datenschutz und Datensicherheit - DuD*, 38, 4 (2014), 237–241.

[ISO/04a] ISO/IEC: Information technology – Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1). International Standard, ISO/IEC 18092 (2004).

[ISO/04b] ISO/IEC: ISO/IEC 7816: Identification cards - Integrated Circuit cards - Part 1-15 (2004).

[ISO/05] ISO/IEC: Information technology – Telecommunications and information exchange between systems – Near Field Communication Interface and Protocol -2 (NFCIP-2). International Standard, ISO/IEC 21481 (2005).

[ISO/08] ISO/IEC: ISO/IEC 24727: Identification cards – Integrated circuit cards programming interfaces – Part 1-6 (2008).

[KuVe] A. Kuehne, H. Veit: Sirius Sign Server Project. http://sourceforge.net/projects/sirius-sign/.

[Midd] Middleware do Cartao de Cidadao. http://svn.gov.pt/projects/ccidadao.

[MOC] MOCCA: Modular Open Citizen Card Architecture Project.

[Nara] D. Naramski: Digital Signature Service. https://joinup.ec.europa.eu/software/sd-dss/description.

[OASI] OASIS Digital SignatureServices TC: Digital Signature Services DSS Core Protocols, Elements, and Bindings v1.0. 2007.

[OeC] Open eCard. http://www.openecard.org.

[PDF10] PDF Advanced Electronic Signature Profiles. ETSI TS 102 778 (2010).

[Palj] M. Paljak: OpenSC Project – tools and libraries for smart card. https://github.com/OpenSC/OpenSC.

[PC/S12] PC/SC Workgroup: PC/SC Workgroup Specifications, Version 2.01.11, Part 1-10 (2012).

[SEC13] Die Applikationsschnittstelle Security-Layer zur österreichischen Bürgerkarte, v1.2.7. http://www.buergerkarte.at/konzept/securitylayer/spezifikation/20140114/core/core.html (2013).

[SigG10] SigG: Bundesgesetz über elektronische Signaturen (2010), http://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=10003685.

[SIMa12] SIMalliance: Open Mobile API specification, Version 2.03 (2012).

[SVPP12] Signature verification procedures and policies, v1.1.1. ETSI TS 102 853 (2012).

[WHPS+13] T. Wich, M. Horsch, D. Petrautzki, J. Schmölz, D. Hühnlein, T. Wieland, S. Potzernheim: An extensible client platform for eID, signatures and more. *In: Open Identity Summit*, GI e.V. (2013), *LNI*, Bd. 233, 55–68.

[XML10] XML Advanced Electronic Signatures (XAdES), v1.4.2. ETSI TS 101 903 (2010).