www.egiz.gv.at
E-Mail: post@egiz.gv.at
Telefon: ++43 (316) 873 5514
Fax: ++43 (316) 873 5520
Inffeldgasse 16a / 8010 Graz / Austria

Dokumentation

Kryptographie und JavaScript

Automatische Clientseitige Verschlüsselung für Web Applikationen

Version 1.0, 19.09.2012

Autor DI Thomas Lenz – tlenz@egiz.gv.at

Zusammenfassung: Web basierende Applikationen erfreuen sich immer größerer Beliebtheit. Hierbei stellt sich JavaScript immer mehr als die Programmiersprache heraus, welche für komplexe Anwendungen verwendet wird, die direkt im Browser laufen sollen. Bei sicherheitsrelevanten Anwendungen stellt sich jedoch die Frage, wie kryptographische Algorithmen integriert und etwaige Schlüssel sicher abgelegt werden können. In diesem Projekt wurde ein Prototyp eines Cloud Services mit clientseitiger Verschlüsselung implementiert. Es konnte gezeigt werden dass die clientseitige Verschlüsselung mit JavaScript und die Schlüsselverwaltung mittels HTML5 Features funktioniert. Von Seiten der Sicherheit gibt es jedoch bedenken, da es in einem realen Anwendung Szenario, mit diesen Mitteln, eigentlich nicht möglich ist die Daten vor dem Web-Service Betreiber zu schützen. Von Seiten W3C ist jedoch eine Kryptografie API für den Web Browser angedacht und diese würde einen Großteil der Sicherheitsbedenken lösen.

Inhaltsverzeichnis:

Inhaltsverzeichnis: 1
Abbildungsverzeichnis 2
Revision History 3
1 Einleitung 4
2 Anforderungen 5
3 Kryptographie mit JavaScript 7
3.2 Sicherheit 8
4 Prototyp 9
4.1 Aufbau 9
4.2 Implementierung 11
4.3 Konfiguration 13
5 Zusammenfassung und Ausblick 14
Referenzen 15

Abbildungsverzeichnis

Abbildung 1 Blockdiagramm Webanwendung.......... 5
Abbildung 2 Web Frontend.......... 6
Abbildung 3 Web Frontend des Demonstrators. 9
Abbildung 4 Lokalen Schlüsselspeicher initialisieren oder öffnen.......... 9
Abbildung 5 Web Frontend mit geöffnetem Schlüsselspeicher.......... 10
Abbildung 6 Web Frontend mit verschlüsselten Dateien. 11

Revision History

Version	Datum	Autor(en)	
1.0	18.09.2012	Thomas Lenz	Initialversion

1 Einleitung

Web basierende Applikationen, bei denen ein Teil der Funktionalität direkt im Webbrowser ausgeführt wird, erfreuen sich immer größerer Beliebtheit. Hierbei stellt sich JavaScript immer mehr als die Programmiersprache heraus, welche für komplexe Anwendungen verwendet wird, die direkt im Browser laufen sollen. Bei sicherheitsrelevanten Anwendungen stellt sich jedoch die Frage, wie kryptographische Algorithmen integriert und etwaige Schlüssel sicher abgelegt werden können.
Ziel dieses Projekts war es einen Überblick über den funktionalen Umfang von Krypto-Bibliotheken in JavaScript zu bekommen. Dabei wurden sowohl Bibliotheken für symmetrische Verschlüsselung als auch für asymmetrische Verschlüsselung untersucht. Für die sichere Ablage von Schlüsseln wurde HTML5 Offline Storage Technologien verwendet.
Im Weiteren wurde ein Prototyp entwickelt welcher es ermöglicht Daten mit einem Server verschlüsselt auszutauschen. Hierbei werden jedoch die Nutzdaten bereits clientseitig verschlüsselt bevor sie zum Server übertragen werden und clientseitig entschlüsselt nachdem sie vom Server empfangen wurden. Im Unterschied zu einer reinen Sicherung des Kommunikationskanals, wie z.B. bei SSL, werden hierbei automatisch die Nutzdaten auch am Server verschlüsselt abgelegt. Eine solche Applikation kann z.B. für Cloud Anwendungen verwendet werden um Daten nur verschlüsselt abzulegen ohne dass im Vorhinein ein zusätzliches Programm für die Verschlüsselung der Daten erforderlich ist.
Zusätzlich bot dieses Projekt die Möglichkeit zur Analyse in wie weit JavaScript für kryptografische Anwendungen einsetzbar ist.

2 Anforderungen

Im Konzept wurden folgende Anforderungen an den Prototyp gestellt.
Es soll eine Web-Anwendung erstellt werden welche Dokumente, ähnlich wie bei diversen Cloud Providern an einen Server übermittelt. Diese Dokumente sollen jedoch vor der Übermittlung clientseitig mit Hilfe von JavaScript verschlüsselt und erst anschließend an den Server übertragen werden. Abbildung 1 zeigt ein Blockdiagramm einer solchen Anwendung schemenhaft. Der Server besitzt in diesem Fall nur die Funktionalität Dokumente anzunehmen und wieder zur Verfügung zu stellen, da die, in diesem Projekt betrachteten Funktionen, clientseitig ausgeführt werden soll. Die funktionalen Einheiten, welche für die Verarbeitung und die Verschlüsselung der Daten erforderlich sind können, wie in Abbildung 1 dargestellt, in vier Teile aufgeteilt werden.

Zusätzlich zu den im Konzept spezifizierten Anforderungen musste der Prototyp um eine Anmeldefunktion erweitert werden. Diese Anmeldfunktion ist erforderlich um die Dokumente

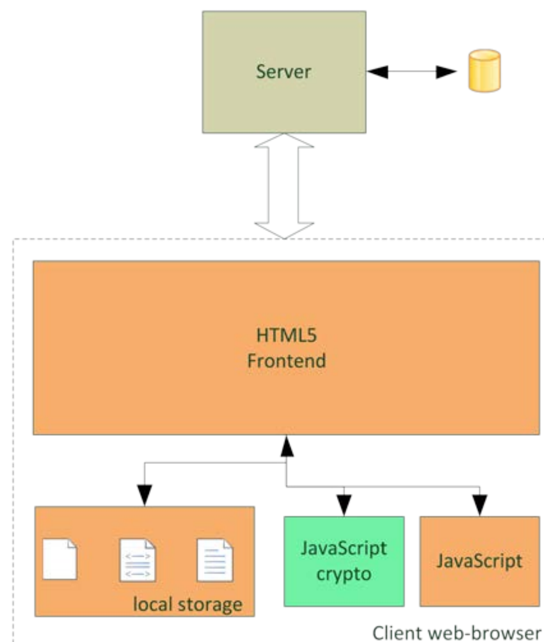

Abbildung 1 Blockdiagramm Webanwendung

den einzelnen Benutzern zuordnen zu können. So kann gewährleistet werden dass jeder Benutzer die vollständige Kontrolle über seine Daten behält. Dokumente können vom Benutzer verschlüsselt an den Server überragen und empfangen werden. Zusätzlich ist es dem/der Benutzer/in möglich die Daten wieder vom Server zu löschen. Sollten die Daten nicht durch den/die Benutzer/in gelöscht werden, werden diese nach einer gewissen Zeit automatisch vom System gelöscht.

Das HTML Frontend ist in Abbildung 2 dargestellt. Dieses besteht aus zwei großen Bereichen. Einem Drag&Drop Bereich für die Dokumente und dem Abschnitt für die Schlüsselverwaltung. Der Schlüsselspeicher ist mit einem Pin geschützt. Somit ist es erforderlich den Schlüsselspeicher vor dem Einfügen von Nutzdaten zu öffnen. Der Schlüsselspeicher besteht aus zwei Bereichen.

- Eigene private Schlüssel
- Öffentlichen Schlüssel von Empfängern

Die Schlüssel können mittels Drag&Drop im jeweiligen Bereich abgelegt werden und stehen danach für Verschlüsselungsoperationen zur Verfügung.

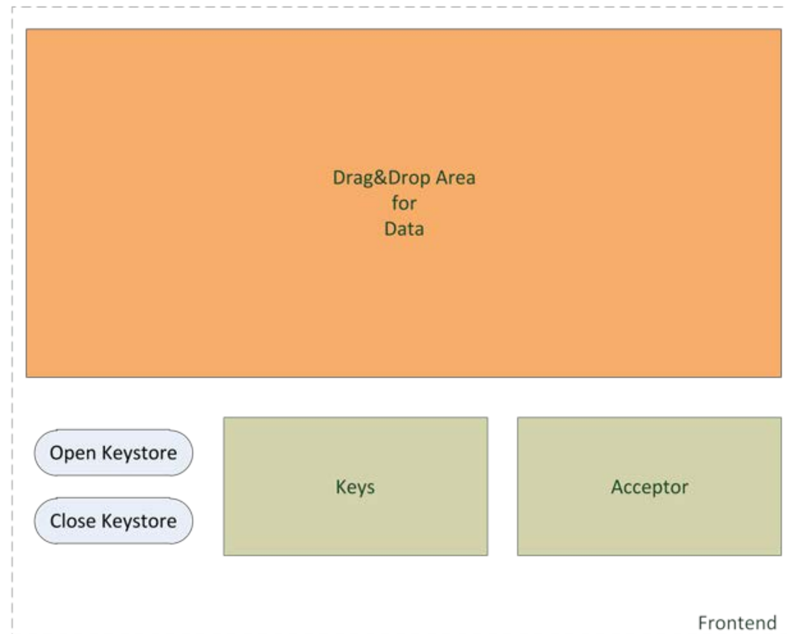

Abbildung 2 Web Frontend

Die Dokumente werden in einem zweistufigen Verfahren verschlüsselt. Zuerst wird ein symmetrischer Schlüssel erstellt mit welchem die Daten verschlüsselt werden und anschließend kann der symmetrische Schlüssel mit Hilfe eines asymmetrischen Verschlüsselungsverfahren für den jeweiligen Empfänger verschlüsselt werden.

3 Kryptographie mit JavaScript

Dieses Kapitel behandelt die Umsetzung von kryptografischen Funktionen mittels JavaScript. Der erste Teil beschäftigt sich mit Krypto-Bibliotheken für JavaScript und deren Einsatz. Im zweiten Teil werden einige Sicherheitsbedenken, die den Einsatz von JavaScript für kryptografische Anwendungen betreffen, näher beleuchtet.

3.1.1 Krypto-Bibliotheken

Es gibt sowohl bestehende fertige Krypto-Bibliotheken für JavaScript als auch die Möglichkeit Java Code mittels Google Web Toolkit (GWT) [gwt] in JavaScript zu übersetzten. Zusätzlich werden auch fertige Kryptografie -Bibliotheken für GWT angeboten. Fast alle fertigen Bibliotheken haben eine symmetrische Verschlüsselung implementiert. Die Funktionalität für asymmetrischer Verschlüsselung, sicherer Zufallszahlengenerierung und Passwort passierende Schlüsselgenerierung variiert jedoch sehr stark. Die in den Bibliotheken implementierten Algorithmen unterscheiden sich zum Teil sehr stark, was einen Vergleich der einzelnen Bibliotheken, bezüglich Performance erschwert.

Ein paar Beispiele für fertige JavaScript Krypto-Bibliotheken sind:

- Stanford JavaScript Crypto Library [stanford]
- Jscryptolib: A JavaScript Cryptography Library [jscrypto]
- JavaScript Cryptography Toolkit [oka]
- RSA and ECC in JavaScript [jsbn]
- jsrsasign – JavaScript implementation of PKCS#1 v2.1 [jspkcs1]

Zusätzlich gibt es die Möglichkeit JavaScript Applikationen mit Hilfe von GWT zu erzeugen. Auch für GWT existieren fertige Bibliotheken welche kryptografische Funktionen implementieren. Diese weisen in etwa denselben Funktionsumfang und dieselben Schwierigkeiten wie die fertigen JavaScript Bibliotheken auf. Somit wurde der Prototyp im Rahmen dieses Projekt ohne Zuhilfenahme von GWT erstellt.

3.1.2 Offline Speicher

Für kryptografische Anwendungen ist die Ablage und Speicherung von kryptografischen Schlüsseln erforderlich. Die Speicherung von kryptografischen Schlüsseln über einen längeren Zeitraum oder über mehrere Sessions kann mit Hilfe von HTML5 Offline Storage Technologien, siehe [KrönerP] gelöst werden. Hierbei bieten sich in HTML5 drei Technologien an.

- DOM Storage
- WEB SQL
- Indexed DB

Bei allen drei Varianten können beliebige Daten abgelegt werden, wobei DOM Storage die beste Browserunterstützung aufweist. Somit wurde diese Technologie für den Prototyp verwendet, wobei DOM Storage in zwei Varianten zur Verfügung steht.

- SessionStorage
- LocalStorage

Diese beiden Varianten weisen dasselbe Interface auf, die Daten werden jedoch unterschiedlich lange gehalten. Im Gegensatz zu SessionStorage, wo die gespeicherten Daten nur für eine Session zur Verfügung stehen und danach automatisch verfallen, bleiben die Daten bei LocalStorage auch nach Beendigung der Session erhalten. Beide Varianten werden durch die Same-Origin-Policy des Browsers geschützt. Somit sollte es für andere Webseiten nicht möglich sein auf die, im DOM Storage abgelegten, Daten zuzugreifen. Ein zusätzlicher Zugriffsschutz ist jedoch nicht vorhanden. Für kryptografische Anwendungen

können beide Varianten eingesetzt werden. Mit Hilfe des *LocalStorage* Interface können Schlüsselspeicher für öffentliche und private Schlüssel erzeugt werden, welche auch über mehrere Sessions zur Verfügung stehen. Der Zugriff ist jedoch nur über die Same-Origin-Policy des Browsers geschützt. Somit empfiehlt es sich die im *LocalStorage* abgelegten Daten zu verschlüsseln. Der hierfür verwende Schlüssel kann mit Hilfe einer Password-Based Key Derivation Funktion aus einem Passwort erzeugt werden. Dieser damit erzeugt Schlüssel muss jedoch nur für eine Session verfügbar sein und soll in der nächsten Session wieder neu erzeugt werden. Für die temporare Speicherung dieses Schlüssel bietet sich das *SessionStorage* Interface an.

3.2 Sicherheit

Beim Thema „Sicherheit bei Kryptografie mit JavaScript“ muss als erstes die Frage gestellt werden vor *‚wem‘* die Daten geschützt werden sollen, da sich für einzelne Akteure durchaus verschiedene Antworten geben
Wie bereits in einem Artikel von Matasano Security [matasono] beschrieben gibt es bei Kryptografie mit JavaScript einige grundlegende Sicherheitsbedenken. Diese betreffen einerseits die sichere Auslieferung des JavaScript Codes vom Server zum Browser und andererseits die clientseitige Inspektion des Sourcecode. Der Sourcecode ist zwar prinzipiell ohne Probleme verfügbar und der Programmablauf beliebig unterbrechbar, eine genaue Inspektion der implementierten Algorithmen ist jedoch in vielen Fällen trotz allem nur mit großem Aufwand möglich. Dieser scheinbare Widerspruch ergibt sich aus der Optimierung der JavaScript Bibliotheken bezüglich der Dateigröße, wodurch meist alle Funktions- und Variablennamen auf ein Minimum gekürzt werden und somit der algorithmische Ablauf nur mehr schwer nachvollziehbar und kontrollierbar ist. Somit ergeben sich zwei scheinbar widersprüchliche Sicherheitsprobleme, denn der Programmablauf kann im Detail beliebig verändert und manipuliert werden und gleichzeitig ist es jedoch für den Benutzer nur schwer möglich den algorithmischen Ablauf zu inspizieren.
Durch diese obengenannten Probleme ist es somit in einem praktischen Anwendungsszenario, wie z.B. das im Prototyp implementierte, nicht möglich die Daten, trotz lokaler clientseitiger Verschlüsselung, vor dem Webservicebetreiber zu schützen. Somit muss dem Infrastrukturbetreiber prinzipiell vertraut werden, da es für ihn recht einfach möglich ist die Verschlüsselung durch eine Änderung der Algorithmen zu umgehen oder den lokalen Schlüsselspeicher an den Server zu übertragen ohne das es vom Anwender sofort erkannt werden kann.
Sollen die Daten jedoch gegenüber Dritten geschützt werden, stellt sich die Situation anders dar. Das Sicherheitsproblem mit der Auslieferung des JavaScripts Codes vom Server zum Browser lässt sich durch Verwendung einer verschlüsselten Verbindung auf ein Minimum reduzieren. Unter der Annahme dass der Webservicebetreiber vertrauenswürdig ist, sichere Algorithmen verwendet werden und die kryptographischen Algorithmen sicher implementiert sind, sollten die Daten am Server gegenüber Zugriff von Dritten geschützt sein. Auch der lokale Schlüsselspeicher weißt einen gewissen Schutz gegenüber Zugriffe von Dritten auf, solange diese nicht direkten Zugriff auf das Endgerät des Benutzer haben. Einerseits ist der Schlüsselspeicher durch die Same-Origin-Policy des Browsers geschützt, wodurch andere Webanwendungen keinen Zugriff auf den *LocalStorage* haben. Zusätzlich werden die Schlüssel im *LocalStorage* mit einem PIN geschützt abgelegt, wodurch ein Zugriff durch Dritte weiter erschwert wird. Dieser Schutz ist jedoch nur aufrecht, so lange ein etwaiger Dritter keinen direkten Zugriff auf das Endgerät hat, denn in diesem Fall würden sich der PIN, und somit auch der Schlüsselspeicher, problemlos auslesen lassen.

4 Prototyp

Der Prototyp wurde wie im Konzept vorgestellt umgesetzt. Um den Prototyp als Demonstrator Online stellen zu können, musste dieser jedoch um eine MOA-ID Anmeldung erweitert werden. Um die Serverinfrastruktur nicht über Kapazität zu belasten wurde der Demonstrator zusätzlich um eine Säuberungsfunktion erweitert, welche in regelmäßigen Abständen nicht mehr verwendete Daten löscht.

4.1 Aufbau

Abbildung 3 zeigt die Startseite des Demonstrators nach erfolgreicher Anmeldung.

Abbildung 3 Web Frontend des Demonstrators.

Nach der Anmeldung muss im ersten Schritt ein neuer Schlüsselspeicher angelegt oder ein bestehender Schlüsselspeicher geöffnet werden. Die im Schlüsselspeicher abgelegten öffentlichen und privaten Schlüssel können danach für Ver- und Endschlüsselungsoperationen verwendet werden. Über die Schaltfläche *‚OpenKeyStore‘* kann ein Schlüsselspeicher initialisiert oder geöffnet werden.

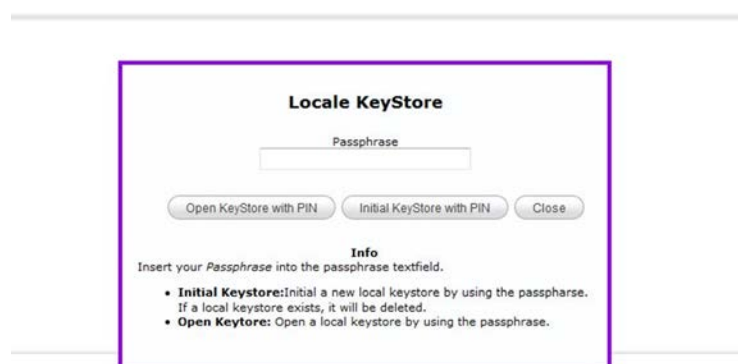

Abbildung 4 Lokalen Schlüsselspeicher initialisieren oder öffnen.

Nach dem betätigen der *‚OpenKeyStore‘* Schaltfläche wird das in Abbildung 4 gezeigte Fenster eingeblendet. Im Textfeld ‚Passphrase‘ muss ein PIN für den lokalen Schlüsselspeicher eingegeben werden. Im Anschluss kann entweder ein bestehender Schlüsselspeicher geöffnet ‚*OpenKeyStore with PIN*‘ oder ein neuer Schlüsselspeicher *‚Initial KeyStore with PIN‘* angelegt werden. Wenn die Initialisierung oder das Öffnen des Schlüsselspeichers erfolgreich war schließt das Fenster automatisch und die Anwendung kehrt zu der in Abbildung 5 dargestellten Ansicht zurück.

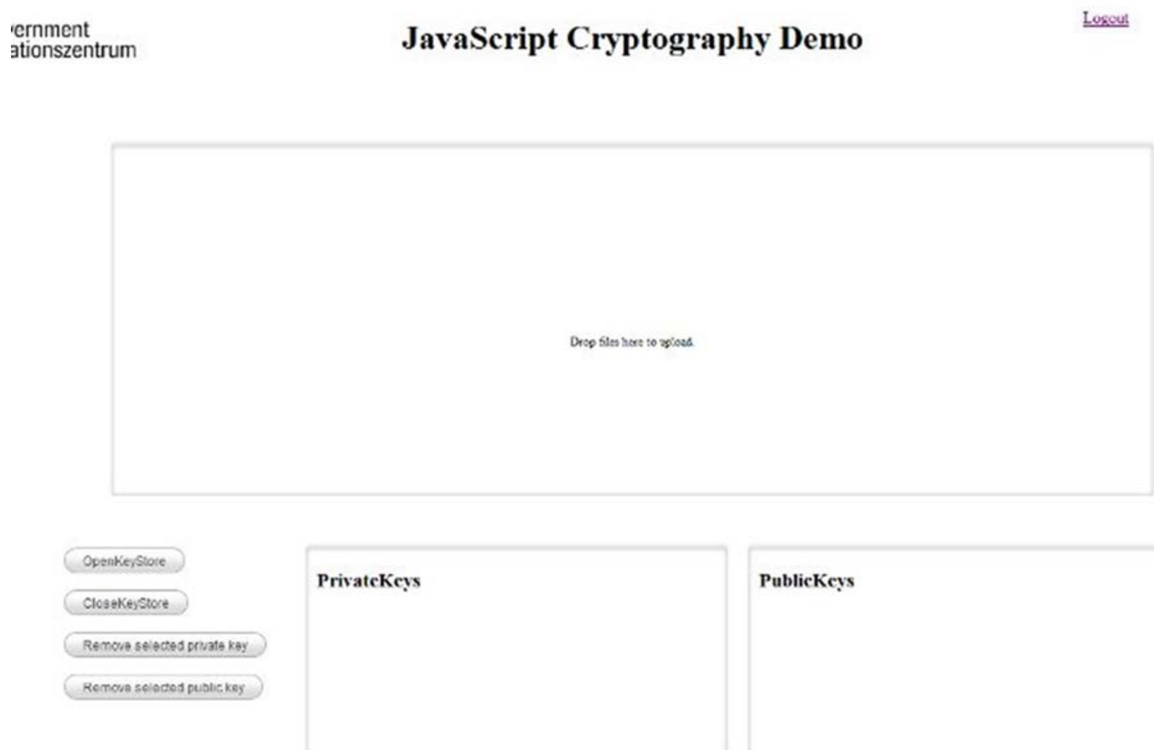

Abbildung 5 Web Frontend mit geöffnetem Schlüsselspeicher.

Bevor Daten verschlüsselt an den Server übertagen werden können muss zuerst der öffentliche Schlüssel des Empfängers eingefügt werden. Hierfür kann ein beliebiges PKCS#1 PublicKey Zertifikat, per Drag&Drop in den *‚PublicKeys‘* Bereich gezogen werden um den Schlüssel in den Schlüsselspeicher einzufügen. Im Bereich *‚PrivateKeys‘* können per Drag&Drop private Schlüssel im Schlüsselspeicher abgelegt werden. Es können beliebig viele öffentliche und private Schlüssel eingefügt werden. Der aktuell verwendete Schlüssel ist blau hinterlegt und kann jederzeit durch einen Mausklick auf einen anderen Schlüsseleintrag geändert werden. Über die Schaltflächen auf der linken Seite in Abbildung 5 kann der Schlüsselspeicher geschlossen oder Schlüssel aus dem Schlüsselspeicher entfernt werden. Nachdem ein öffentlicher Schlüssel ausgewählt wurde können beliebige Daten per Drag&Drop in den *‚FileUpload‘* Bereich gezogen werden. Aktuell ist die Dateigröße mit 20 MB beschränkt und es können maximal 5 Dateien gleichzeitig verarbeitet werden. Abbildung 6 zeigt die Oberfläche nachdem ein öffentlicher Schlüssel ausgewählt und zwei Dateien an den Server übertragen wurden. Wenn nun ein privater Schlüssel ausgewählt wird, kann der Download der verschlüsselten Datei durch einen Doppelklick auf die dementsprechende Schaltfläche ausgelöst werden. Die Datei wird im Hintergrund an den Browser übertragen und dann lokal entschlüsselt.
Zum aktuellen Zeitpunkt ist ein vollständiger Download mittels JavaScript nur in Google Chrome Version größer 14.06 möglich. Alle anderen Browser benötigen aktuell noch ein Adobe Flash Programm um die entschlüsselte Datei aus dem Browser wieder in das lokale Dateisystem zu schreiben. Hierfür wird nach erfolgreicher Entschlüsselung eine *‚SaveAs‘*

Schaltfläche eingeblendet mit deren Hilfe die Datei in das lokale Dateisystem geschrieben werden kann.

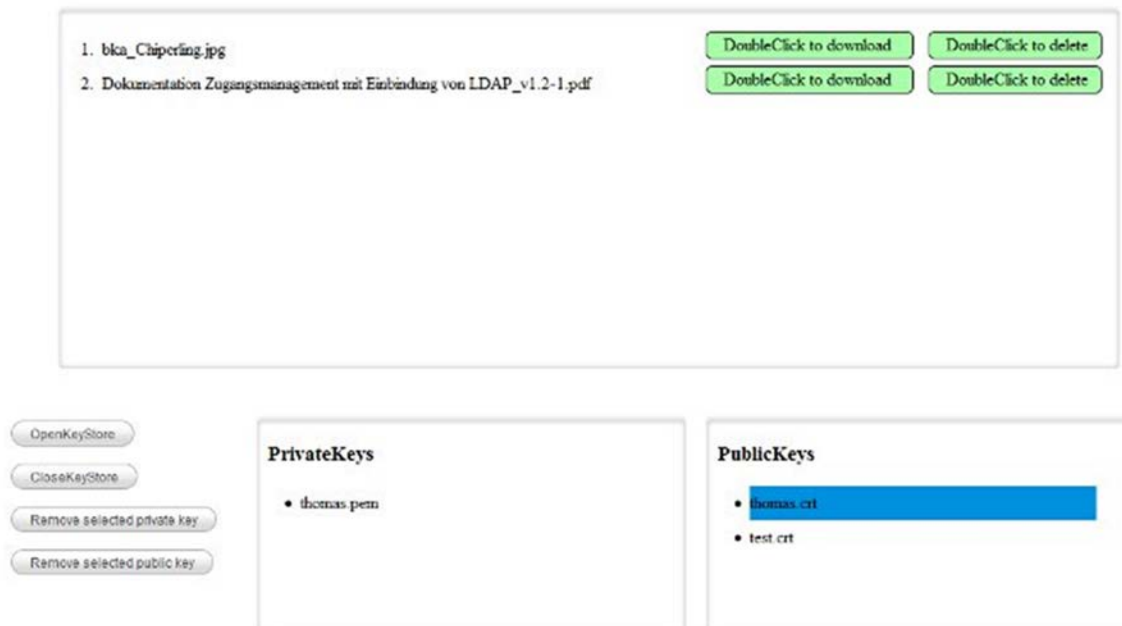

Abbildung 6 Web Frontend mit verschlüsselten Dateien.

Zusätzlich bietet das Web Frontend die Möglichkeit Datei die an den Server übertragen wurden manuell wieder zu löschen. Sollten die an den Server übertragenen Dateien nicht durch den Benutzer gelöscht werden, werden diese sechs Stunden nach der letzten Anmeldung am System automatisch gelöscht. Bei einer Anmeldung innerhalb dieser Zeitspanne stehen die verschlüsselten Dateien jedoch wieder zur Verfügung und können durch den Benutzer verwaltet werden.

4.2 Implementierung

Der Serverteil des Demonstrators ist in JAVA implementiert und beinhaltet die Verwaltung der verschlüsselten Dokumente. Zur Kommunikation mit dem Web Frontend wird Asynchronous JavaScript and XML (AJAX) verwendet, wobei das serverseitige http Interface mittels Struts2 [struts2] umgesetzt wurde.
Das Web Frontend, welches die kryptografische Funktionalität beinhaltet ist in HTML5 und JavaScript implementiert. Hierfür wurden folgende JavaScript Bibliotheken verwendet.

- **jQuery [jquery]:**
  jQuery ist eine freie JavaScript Bibliothek. Diese implementiert Funktionen mit deren Hilfe DOM-Manipulationen und AJAX http Anfragen einfach umgesetzt werden können. Der Vorteil in der Verwendung solcher Bibliotheken liegt an dem Umstand dass nach wie vor nicht alle Browser JavaScript im selben Umfang unterstützen und dadurch bietet diese Bibliothek, in weiten Bereichen, eine browserunabhängige Abstraktionsebene.
- **jQuery filedrop Plugin [jqfiledrop]:**
  jQuery filedrop Plugin ist eine Erweiterung zu jQuery und bieten ein Framework für HTML5 Drag&Drop Operationen.
- **Stanford JavaScript Crypto Library [standford]**
  Die Stanford JavaScript Crypto Library implementiert kryptografische Algorithmen in JavaScript. Im Demonstrator werden folgende Algorithmen aus dieser Bibliothek verwendet

    - AES
    - PBKDF2
    - RandomNumberGenerator
- **jspn RSA and ECC in JavaScript [jspn]**
  Diese JavaScript Bibliothek implementiert kryptografische Algorithmen für asymmetrische Kryptografie. Die darin enthaltenden Algorithmen für Verschlüsselung und Entschlüsselung mittels RSA werden im Demonstrator verwendet.
- **jsrsasign – JavaScript implementation of PKCS#1 v2.1 [jspkcs1]**
  Diese JavaScript Bibliothek implementiert Algorithmen um PKCS#1 Container zu verarbeiten. Diese PKCS#1 Container werden für die asymmetrische Verschlüsselung benötigt.
- **Downloadify - Client Side File Creation [dlify]:**
  Diese Bibliothek verwendet JavaScript und Adobe Flash und implementiert eine Funktion mit deren Hilfe eine Datei aus dem Browser in das lokale Dateisystem geschrieben werden kann. Dieser Umweg ist erforderlich, da zurzeit nur ein Browser (Google Chrome Version 14.06 und höher) im vollen Umfang eine API für diese Funktionalität bereitstellt. Die in Mozilla Firefox implementierte API stellt zwar prinzipiell denselben Funktionsumfang zur Verfügung, es ist jedoch in der aktuellen Implementierung nicht möglich den Dateinamen manuell festzulegen, stattessen wird ein automatisch generierter Dateiname verwendet. Eine Änderung dieser Implementierung ist in der nächsten Zeit nicht zu erwarten, da diese Funktionalität von Mozilla als Sicherheitsrisiko eingestuft wird. [1]

4.2.1 Verschlüsselung

Jede Datei wird in einem zweistufigen Verschlüsselungsverfahren verschlüsselt. Zuerst wird ein symmetrischer Schlüssel generiert mit dem die Datei mittels AES verschlüsselt wird. In einem zweiten Schritt wird der symmetrische Schlüssel mittels RSA und dem gewählten öffentlichen Schlüssel verschlüsselt. In der aktuellen Implementierung wird jede Datei nur für einen speziellen Empfänger verschlüsselt. Somit kann jede Datei nur für einem Empfänger (öffentlichen Schlüssel) verschlüsselt werden..
Die verschlüsselte Datei und der verschlüsselte symmetrische Schlüssel wird zu einem JavaScript Object Notation (JSON) Objekt zusammengefügt und anschließend mit einem http POST Request an den Server übertragen. Die nachfolgende Liste zeigt die einzelnen Schritte dieses Verschlüsselungsvorgangs.

1. Datei hinzufügen mittels HTML5 Drag&Drop Operation
2. Generierung eines symmetrischen Schlüssels
3. AES Verschlüsselung der Base64 codierten Datei
4. RSA Verschlüsselung des symmetrischen Schlüssels
5. Generierung des verschlüsselten Containers bestehend aus Datei und Schlüssel
6. Übertragung des Containers mittels eines AJAX http POST Request

4.2.2 Entschlüsselung

Die Entschlüsselung einer Datei erfolgt in umgekehrter Reihenfolge wie in Kapitel 4.2.1 beschrieben.

1. Übertragung des Containers mittels einer AJAX http POST Response
2. RSA Entschlüsselung des symmetrischen Schlüssels aus dem Container
3. AES Entschlüsselung der Datei mit dem symmetrischen Schlüssel
4. Download der Datei aus dem Browser mittels JavaScript. Wie im Anfang von Kapitel 4.2 beschrieben ist eine reine JavaScript Implementierung in diesem Abschnitt jedoch nur mit Google Chrome in der Version 14.06 oder höher möglich.

[1] https://bugzilla.mozilla.org/show_bug.cgi?id=676619

4.2.3 Server Interfaces

Für die Kommunikation mit dem Server wurden folgende vier Interfaces implementiert, welche vom Web Frontend mittels AJAX verwendet werden. Alle vier Interfaces benutzen für die Strukturierung der Nutzdaten JSON. Als Schutzmaßnahme gegen Cross-Site-Scripting (XSS) werden alle Dateinamen und internen Identifier vor der Verarbeitung vor ihren Inhalt geprüft. Sollte sich in diesen Objekten JavaScript Code befinden wird dieser automatisch entfernt.

- **/ajax/load_filelist**
  Liefert in der Response ein JSON Objekt zurück das alle Daten des aktuellen Benutzer am Server beinhaltet. Dieses JSON Objekt besteht aus den Dateinamen und den internen Identifier.
- **/ajax/upload_file**
  Dieses Interface erwartet einen http POST MultiPartRequest welcher die zu speichernde Datei beinhaltet. Als Response werden der Dateiname und der interne Identifier der Datei zurückgegeben.
- **/ajax/download_file**
  Dieses Interface erwartet im Request ein JSON Objekt mit dem internen Identifier. Als Response wird der verschlüsselte JSON Container der angeforderten Datei zurückgegeben.
- **/ajax/delete_file**
  Dieses Interface erwartet im Request ein JSON Objekt mit dem internen Identifier. Als Response wird eine aktualisierte Dateiliste zurückgegeben. Hierbei ist es möglich dass sich die internen Identifier der Dateien ändern.

4.3 Konfiguration

Für die Installation des Demonstrators ist zusätzlich eine Konfiguration des Serverteils erforderlich. Die Konfiguration befindet sich im WEBAPPS Ordner der jeweiligen Tomcat Instanz unter folgendem Pfad abgelegt.
./WEB-INF/classes/at/gv/egiz/jscryptodemo/config/configuration.xml.
Darin sind folgende Einstellungen möglich.

Parameter	Wert	Beschreibung
file.store.path	D:/teststore	Pfad unter dem die hochgeladenen Dateien abgelegt werden sollen.
File.database	Database.xml	Name der XML Datenbank zur Verwaltung der hochgeladenen Dateien.
Cleanup.delay	6	Intervall in Stunden. Nach dieser Zeit werden Benutzerdaten wieder vom System gelöscht.
MOAID.URL	https://localhost:8846/moa-id-auth	URL zur MOA-ID Instanz die für die Anmeldung am System verwendet werden soll

5 Zusammenfassung und Ausblick

Browserseitige Verschlüsselung von Dokumenten mittels HTML5 Features und JavaScript ist von funktionaler Seite in modernen Browsern möglich. Das betrifft sowohl die Ablage und Speicherung von Schlüsseln mittels HTML5 Lokal Storage Technologien als auch den eigentlichen Verschlüsselungsvorgang mittels symmetrischer und asymmetrischer Verschlüsselung. Einzig und allein das Zurückschreiben der entschlüsselten Daten auf die Festplatte stellt bei reiner JavaScript Verwendung noch ein Problem dar, da nicht alle Browser diese Funktionalität unterstützen.
Aus sicherheitstechnischer Sichtweise gibt es jedoch wie in Kapitel 3.2 beschrieben einige Bedenken an eine reine JavaScript Lösung. Die Daten können zwar mit Hilfe von JavaScript Kryptographie vor Dritten, aber eigentlich nicht vor dem Web-Service Betreiber geschützt werden. Somit stellt sich in diesem Bereich aktuell die Frage in wie weit die Verschlüsselung der Daten zweckmäßig ist.
Die in Kapitel 3.2 beschriebenen Sicherheitsbedenken bezüglich der kryptographischen Funktionen, welche vom Web-Service Betreiber zur Verfügung gestellt werden, könnten zukünftig jedoch gelöst werden. Aktuell gibt es von der W3C den Vorschlag für eine browserseitige Kryptographie API [wcryAPI]. Ein Working Draft für eine solche API wurde Mitte September 2012 von der W3C veröffentlicht. Der Funktionsumfang dieser API entspricht dem Funktionsumfang der in diesem Demonstrator verwendeten JavaScript Bibliotheken. Somit könnte zukünftig auf, die vom Web-Service zur Verfügung gestellten Kryptografie Bibliotheken, verzichtet werden. Das würde die in Kapitel 3.2 beschriebenen Probleme stark eindämmen würde.

Referenzen

[dlify]	Downloadify: Client Side File Creation; https://github.com/dcneiner/Downloadify
[guerreroD]	Base64 Binary Decoding in JavaScript http://blog.danguer.com/2011/10/24/base64-binary-decoding-in-javascript/
[gwt]	Google Web Toolkit; Productivity for developers, performance for users2.5 Release Candidate, with major performance improvements for your application, and powerful new libraries for building user interfaces. https://developers.google.com/web-toolkit/
[KrönerP]	Kröner Peter; HTML5 Webseiten innovative und zukunftssicher 2.Auflage;opensource PRESS München; 2011
[jquery]	jQuery JavaScript Library http://jquery.com/
[jqfiledrop]	jQuery filedrop plugin - html5 drag desktop files into browser https://github.com/weixiyen/jquery-filedrop
[jscrypto]	jscrypto: A JavaScript Cryptography Library http://code.google.com/p/jscryptolib/
[jspkcs1]	Jsrsasign – JavaScript implementation of PKCS#1 v2.1 http://kjur.github.com/jsrsasign/
[jspn]	RSA and ECC in JavaScript http://www-cs-students.stanford.edu/~tjw/jsbn/
[matasano]	Javascript Cryptography Considered Harmful; Matasano Security http://www.matasano.com/articles/javascript-cryptography/
[oka]	JavaScript Cryptography Toolkit http://ats.oka.nu/titaniumcore/js/crypto/readme.txt
[stanford]	Stanford Javascript Crypto Library http://crypto.stanford.edu/sjcl/
[struts2]	Apache Struts 2; http://struts.apache.org/2.x/
[wcryAPI]	Web Cryptography API; W3C Working Draft 13 September 2012; http://www.w3.org/TR/2012/WD-WebCryptoAPI-20120913/