Zentrum für sichere Informationstechnologie – Austria
Secure Information Technology Center – Austria

A-1030 Wien, Seidlgasse 22 / 9
Tel.: (+43 1) 503 19 63–0
Fax: (+43 1) 503 19 63–66

A-8010 Graz, Inffeldgasse 16a
Tel.: (+43 316) 873-5514
Fax: (+43 316) 873-5520

http://www.a-sit.at
E-Mail: office@a-sit.at

DVR: 1035461 ZVR: 948166612 UID: ATU60778947

BROWSER-PLUGIN ZUR INTERAKTION MIT SKYTRUST

Version 1.0, 12. Oktober 2015
Johannes Feichtner – johannes.feichtner@a-sit.at

Zusammenfassung: Online-Dienste zur Ablage von Daten (z.B. Dropbox, Google Drive) bieten keine Möglichkeit, die hinterlegten Daten vor dem Hochladen zu verschlüsseln. Die Geschlossenheit dieser Dienste verhindert die individuelle Addition von Verschlüsselung und würde ohnedies keine allgemein verwendbare Lösung darstellen. Als Konsequenz wurde in diesem Projekt im Rahmen eines Prototyps ein Browser-Plugin für Google Chrome umgesetzt, das ohne Umweg über tertiäre Applikationen bestehende Browser-Schnittstellen so umleitet, dass von AnwenderInnen gegebene Daten über eine sichere Plattform (SkyTrust) einer kryptographischen Operation unterzogen werden. Der Vorteil dieses Ansatzes ist, dass existierende Web-Applikationen keiner speziellen Anpassung bedürfen, damit die Daten von BenutzerInnen kryptographisch geschützt werden. Zusätzlich zur Bereitstellung dieser Funktionalität, umfasst der Fokus dieses Projekts auch die Behandlung von Sicherheitsaspekten, die dafür verantwortlich sind, unverschlüsselte Eingaben im Browser organisatorisch von verschlüsselten zu separieren, welche schließlich an einen Online-Dienst geschickt werden.

Dieses Dokument beschreibt die Ergebnisse des durchgeführten Projekts, das erarbeitete Konzept für die sichere Integration einer zentralen Verschlüsselungskomponente in Web-Browser und behandelt sicherkritische Aspekte bei der Umsetzung eines „Proof-of-Concepts“ für Google Chrome.

Inhaltsverzeichnis

1. Einleitung

Traditionell bieten Online-Dienste zur Ablage von Informationen (z.B. Dropbox, Google Drive) keine Möglichkeit, die von AnwenderInnen hinterlegten Daten vor dem Hochladen zu verschlüsseln um somit dem Anbieter eines Speicherdienstes keine Klartextinformationen zu übermitteln. Zielgerichtete Desktop-Programme von Drittherstellern sind prinzipiell in der Lage, dieser Aufgabe nachzukommen. Jedoch sind diese Applikationen teils nicht für das jeweils eingesetzte (Mobil-) Betriebssystem verfügbar oder schützen die eingesetzten kryptographischen Schlüssel nur unzureichend. Analog zur Ablage von Daten bei Online-Diensten gebrauchen Web-Applikationen vermehrt Schnittstellen, die in modernen Web-Browsern bereitgestellt werden, um benutzerspezifische Informationen lokal abzulegen. Analog zum zuvor beschriebenen Szenario mit Online-Speichern sehen auch entsprechende lokale Schnittstellen des Browsers hierbei keinen expliziten Schutz der abgelegten Daten vor.

Die Gewährleistung des angestrebten Schutzes durch die Anwendung kryptographischer Operationen bedingt wiederum grundsätzlich den korrekten Umgang mit Schlüsselmaterial. Web-Browser sind jedoch prinzipiell nicht darauf ausgerichtet, Schlüssel sicher zu verwahren. So unterstützt etwa die W3C Crypto API[1], welche prinzipiell von allen modernen Browsern für kryptographische Anwendungen bereitgestellt wird, keine Mechanismen zum Schutz von Schlüsselmaterial. Web-Applikationen können damit weder auf Smart Cards oder Secure Elements zugreifen, noch die verwendeten Schlüssel in geeignetem Format aus anderen Quellen sicher importieren. Nicht zuletzt dieser Umstand führt schließlich zur Tatsache, dass die, durch Aufruf entsprechender Schnittstellen im Browser abgelegten Daten, mittels der W3C Crypto API nicht verschlüsselt werden können und dies auch der geschützten Ablage im lokalen Browser entgegensteht.

In einem anderen Projekt wurde am Beispiel der zentralisierten Plattform für kryptographische Operation namens SkyTrust bereits gezeigt, wie über eine W3C Crypto API kompatible Schnittstelle im Browser Dateien ver- und entschlüsselt werden können. Die entsprechenden Schlüssel wurden dabei stets durch flexible Authentifizierungsmaßnahmen geschützt.

Das Ziel dieses Projekts bestand darin, eine Erweiterung für einen Browser zu entwickeln, welche die W3C Crypto API für SkyTrust, einem zentralen Schlüsselspeicher, adaptiert und anhand der daraus resultierenden Anwendungsmöglichkeit ein praktischer Mehrwert für bestehende Web-Applikationen entsteht. Gemäß der eingangs erwähnten Defizite bei der sicheren Ablage von Informationen im Browser, definiert sich der Nutzen der Browser-Erweiterung anhand folgender Ziele:

- Transparente Integration eines zentralen Schlüsseldienstes über bestehende Schnittstellen im Browser. Dies umfasst zum einen eine Modifikation der W3C Crypto API hinsichtlich der Bereitstellung von Schlüsselmaterial über einen externen Dienst. Zum anderen ist auch eine Anpassung weiterer Schnittstellen wie HTML5 WebStorage[2] und dem damit verwandten SessionStorage vonnöten, damit darüber abgelegte Daten mittels der W3C Crypto API verschlüsselt werden.
- Organisatorische Trennung von sicherheitskritischer Funktionalität (SkyTrust) von den Inhalten einer Webseite, welche eine Schnittstelle zur Datenablage verwendet. In der Praxis bedeutet das, dass eine aufgerufene Webseite keine Möglichkeit haben darf, auf einen Ver- bzw. Entschlüsselungsprozess Einfluss zu nehmen. Ähnliches gilt für die Authentifizierung eines Benutzers bei SkyTrust, etwa mittels Benutzername und Passwort, die von einer Webseite durch entsprechende Maßnahmen insofern getrennt sein muss, dass die Web-Applikation auf die verwendeten Anmeldedaten keinesfalls zugreifen kann.

Im Folgenden wird das erarbeitete Konzept vorgestellt, um die angeführten Ziele umzusetzen. Am Beispiel von Google Chrome wird dabei aufgezeigt, wie die vom Browser bereitgestellten

[1] http://www.w3.org/TR/WebCryptoAPI/
[2] http://www.w3.org/TR/webstorage/

Sicherheitsfunktionen eingesetzt werden können, um eine Interaktion zwischen Web-Applikation und Browser-Erweiterung unter Wahrung sicherheitskritischer Aspekte zu realisieren.

2. Entwicklungsrahmen

Um die in der Einleitung angeführten Ziele zu erreichen, führt die nachstehende Aufstellung mehrere Anforderungen auf und stellt zugleich einen Leitfaden dar, um die Browser-Erweiterung umsetzen zu können:

- Um Uploads, die bei Anbietern von Online-Storage getätigt werden, zu erkennen, muss das Browser-Plugin zunächst die Browser-Schnittstellen (APIs) umleiten, die von diesen Diensten verwendet werden. Im ersten Schritt galt es also, dahingehend Know-How aufzubauen, ob das Überschreiben von Browser-Schnittstellen überhaupt so möglich ist, dass sich eine generell einsetzbare Lösung daraus ableiten lässt.

- Die Unterbrechung und Umleitung von Anfragen, die an Server geschickt werden, stellt eine ins Auge zu fassende Lösungsalternative zur Abänderung von Schnittstellen dar. Da es allerdings gegenwärtig nur bei Mozilla Firefox möglich ist, im Rahmen einer Erweiterung Anfragen an Server abzubrechen, zu modifizieren oder umzulenken, wurde favorisiert, Browser-Schnittstellen so zu ändern, dass Anfragen bereits umgelenkt werden, bevor sie tatsächlich gesendet werden. Dieser Ansatz lässt sich außerdem auf Offline-Speicher wie HTML5 WebStorage bzw. SessionStorage übertragen.

- Um den Login-Prozess (z.B. mittels Username und Passwort) seitens SkyTrust durchzuführen, muss eine entsprechende Login-Maske bereitgestellt werden. Selbiges gilt für die Auswahl eines kryptographischen Schlüssels.

  Bei Firefox[3] kann dies etwa im Rahmen eines Panels passieren. Bei Chrome muss ein neues Browser-Fenster geöffnet werden[4] oder ein Popup[5]. Um dabei mit der Erweiterung zu interagieren, kann bei Firefox die port API[6] verwendet werden. Unter Chrome wird jedes Browserfenster durch einen eigenen Prozess abgebildet, weswegen bislang ungeklärt ist, ob eine Interprozesskommunikation prinzipiell realisierbar ist. Alternativ kann der Login-Prozess umgesetzt werden, indem das *Document Object Model (DOM)* modifiziert wird. Eine direkte Injektion des Logins in die Seite kann unter Umständen jedoch implizieren, dass Angriffe durch Cross-Site-Request-Forgery (CSRF) oder Cross-Site-Scripting (XSS) begünstigt werden.

- Die Anzeige des Ver- und Entschlüsselungsprozess sollte über den gleichen Mechanismus umgesetzt werden, der auch für die zuvor genannte Funktionalität eingesetzt wird. Da dabei aber sensiblen Informationen verarbeitet werden, könnte dies auch durch eine Injektion von Objekten in den DOM realisiert werden. Konkret heißt das, dass Browser-Schnittstellen bereits zu dem Zeitpunkt geändert werden müssen, bevor eine Ablage im lokalen Browser-Speicher vorgenommen wird oder in den Upload- bzw. Downloaddialog eines Online-Ablage-Dienste von SkyTrust verschlüsselte Daten eingefügt werden können.

Aus den angeführten Punkten lässt sich schließen, dass die Erweiterung prinzipiell für Firefox und/oder Chrome umgesetzt werden kann. Um die gewünschte Integration einer transparenten Datenverschlüsselung umzusetzen, gilt es also 1) Möglichkeiten zu eruieren, um Browser-Schnittstellen zu überschreiben und 2) eine Interaktion mit SkyTrust vom Zugriff einer Web-Applikation so zu separieren, dass mögliche Sicherheitsprobleme wie CSRF und XSS im Kontext der Browser-Erweiterung nicht ausgeführt werden können.

---

[3] https://developer.mozilla.org/en-US/Add-ons/SDK/High-Level_APIs/panel
[4] https://developer.chrome.com/extensions/windows#method-create
[5] https://developer.chrome.com/extensions/pageAction
[6] https://developer.mozilla.org/en-US/Add-ons/SDK/Guides/Content_Scripts/using_port

3. Sicherheitskonzepte für Erweiterungen im Browser

Im ersten Schritt zur Realisierung einer Browser-Erweiterung wurden die für Entwickler zur Verfügung gestellten Schnittstellen mehrerer Browser (Mozilla Firefox, Google Chrome, Microsoft Internet Explorer bzw. Edge, Opera und Safari) hinsichtlich der beschriebenen Anforderungen evaluiert. Darauf aufbauend wurden die Mechanismen identifiziert, die in der Lage sind, die angestrebte Funktionalität bereitzustellen.

Dabei hat sich herauskristallisiert, dass ein Plugin üblicherweise nur dann ohne größeren Aufwand für mehrere Browser gleichzeitig entwickelt werden kann, wenn es ausschließlich mit dem *Document Object Model (DOM)* interagiert. Greift ein Browser also ausschließlich auf den Inhalt der aktuell angezeigten Seite zu (z.B. um etwas einzufügen), kann über JavaScript + CSS ein Plugin realisiert werden, das weitgehend mit der gleichen Codebasis in Chrome, Firefox, Safari und Opera lauffähig ist.

Da im vorliegenden Fall Daten von AnwenderInnen verschlüsselt werden sollen, die im DOM einer geladenen Web-Applikation verfügbar sind und nur standardisierte Schnittstellen (W3C CryptoAPI, WebStorage) in den Browsern verwendet werden sollen, korreliert die Sicherheit der Daten von BenutzerInnen wesentlich mit dem Sicherheitsmodell, das die jeweiligen Browser für Erweiterungen zugrunde legen. Ein Vergleich der eingangs angeführten Browser-Hersteller hat dabei aufgezeigt, dass etwa Firefox und Internet Explorer eine Erweiterung prinzipiell im gleichen DOM laden wie die geladene Webseite. In praktischer Hinsicht impliziert dies, dass sich sicherheitskritische Funktionalität grundsätzlich nicht vom geladenen Inhalt einer Webseite trennen lässt. Im konkreten Anwendungsfall hätte also eine geladene Webseite die theoretische Möglichkeit, eine über ein Formular durchgeführte Authentifizierung bei SkyTrust zu modifizieren bzw. Logindaten zu stehlen. Um dem entgegen zu wirken, bietet etwa Firefox an, dass Erweiterungen eigene Fenster öffnen können (sog. Panels), welche schließlich in einem eigenen DOM ausgeführt werden. Bei Google Chrome ist ein separater DOM für die Inhalte einer Browser-Erweiterung von vorne herein vorgesehen. Wie in Abbildung 1 dargestellt, läuft eine Chrome Extension prinzipiell getrennt von einer Web-Applikation. Um dennoch auf die Inhalte bzw. den DOM zuzugreifen, können sowohl sog. „Background Scripts“ oder „Content Scripts“ eingesetzt werden. Der wesentliche Unterschied ist, dass „Background Scripts“ unabhängig von einer Webseite nur einmal geladen werden. „Content Scripts“ im Gegensatz dazu werden für jede geladene Webseite neuerlich geladen und im DOM der Webseite ausgeführt.

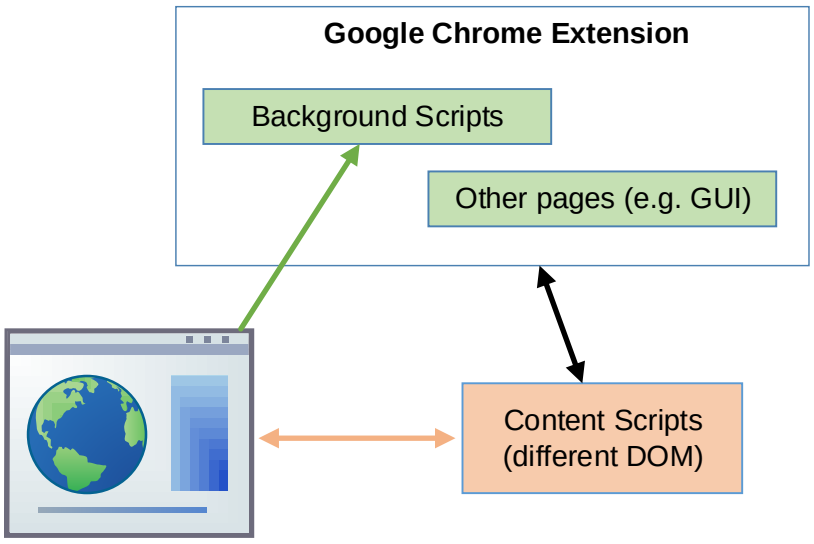

Abbildung 1.Struktur von Browser-Erweiterungen in Google Chrome

Durch die klar definierte Trennung der DOMs von Web-Applikation und Browser-Erweiterung, sowie der Möglichkeit, über „Content Scripts“ auf den DOM der jeweiligen Webseite zuzugreifen, wurde

somit die Erarbeitung eines Browser-Plugins für Chrome lanciert. Das bei Chrome gegebene Sicherheitsmodell für Erweiterungen erfüllt die gewünschten Bedingungen, wonach SkyTrust bzw. kryptographisches Schlüsselmaterial organisatorisch von einer geladenen Webseite vollständig getrennt werden kann und sich über „Content Scripts“ ein einseitiger Zugriff vom Plugin auf die Webseite sicherstellen lässt.

4. Browser-Erweiterung

In diesem Kapitel werden die Funktionalität und gewählten Lösungsansätze bei der Implementierung der zuvor angeführten Fragestellungen erläutert.

4.1. Verschlüsselung von Daten vor Online-Upload

Die nachfolgende Grafik illustriert den Aktionsablauf, für den Fall, dass Dokumente bei einem Online-Dienst hochgeladen werden sollen. Zuerst muss die Browser-Erweiterung den regulären Upload-Vorgang, je nach verwendetem Online-Speicherdienst, erkennen und abbrechen bzw. aufhalten. AnwenderInnen werden in weiterer Folge dazu angeleitet, sich bei SkyTrust anzumelden. Diese Anmeldung wird im Rahmen eines Anmeldedialogs vollzogen, in das Username und Passwort einzugeben sind. Sofern von SkyTrust nicht bereits vorgegeben, müssen AnwenderInnen die zu verwendenden SkyTrust-Schlüssel auswählen. Nach erfolgter Verschlüsselung eines Dokuments wird ein CMS-Container zurückgegeben, welcher schließlich beim Online-Speicherdienst abgelegt werden kann. Beim Downloadvorgang werden die Schritte analog in umgekehrter Richtung ausgeführt.

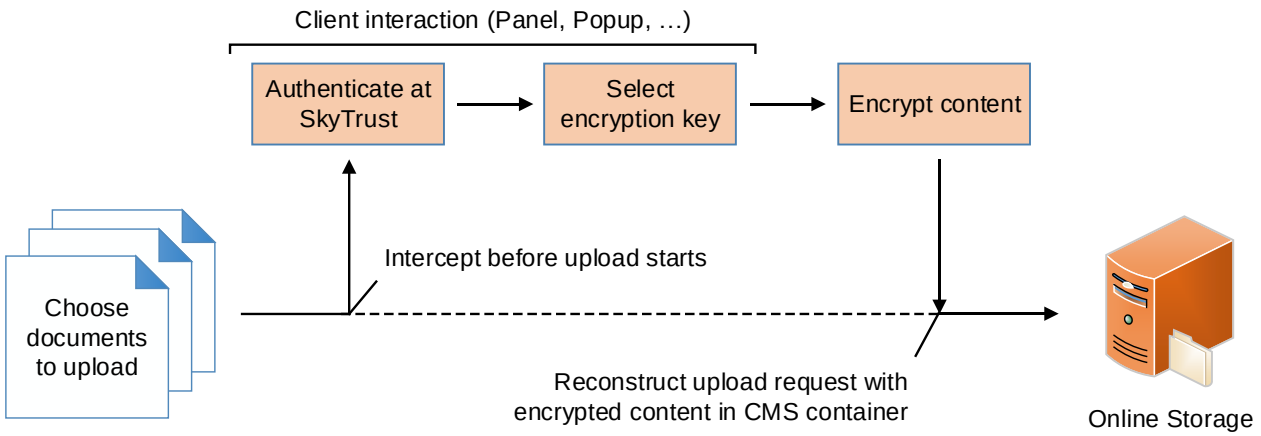

Abbildung 2. Online-Upload mit Umleitung über SkyTrust

Um den beschriebenen Ansatz umzusetzen, wurde zunächst analysiert, wie populäre Online-Speicherdienste wie Google Drive und Dropbox von AnwenderInnen bereitgestellte Daten hochladen. Eine Studie des Quellcodes der beiden Dienste hat dabei gezeigt, dass Uploads bei beiden Diensten über die Schnittstelle XMLHttpRequest[7] initiiert werden. Folglich galt es, eine Möglichkeit zu finden, die getätigten AJAX-Requests zu unterbinden und die Daten anstelle dessen an SkyTrust zu senden.

Die einzige Möglichkeit um über die XMLHttpRequest API gesendete Anfragen umzuleiten, findet sich in der Verwendung der neuen Webtechnologie Reflect[8], welche Teil des JavaScript-Standards ECMAScript 6 ist. Unter Zuhilfenahme dieser Technologie kann beliebigen JavaScript-Operationen eine neue Funktionalität zugewiesen werden. Durch Überschreibung wesentlicher Methoden in der XMLHttpRequest API ließen sich schließlich die URL herausfinden, an welche Daten gesendet werden, sowie die eigentlichen Daten. Da die Anfragen der Online-Dienste synchron gestellt wurden (mit Erwartung eines Ergebnisses vom Server) konnten in der Zwischenzeit die Daten an SkyTrust übermittelt werden, von welchem ein verschlüsselter CMS-Container zurückgeliefert wurde. Durch

[7] https://developer.mozilla.org/de/docs/Web/API/XMLHttpRequest
[8] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Reflect

Konstruktion einer neuen XMLHttpRequest-Anfrage an den originalen Speicherdienst wurden schließlich verschlüsselte Daten an den Speicherdienst übermittelt.

Bei der Erarbeitung dieser Umsetzung hat sich herausgestellt, dass der gewählte Ansatz in der Praxis nicht weitläufiger einsetzbar ist. Die unter Versuchsbedingungen gestellten Anfragen bestanden jeweils nur aus einer geschlossenen Anfrage. Das bedeutet, dass eine Datei mit einer Anfrage als Ganzes hochgeladen wurde. Die zuvor erwähnten Dienste verwenden für ihre Upload-Funktionalität jedoch mehrere kontinuierliche Anfragen, die Daten in mehreren, kleinen Teilen (Chunks) zum Online-Speicherdienst senden und dort wieder zusammensetzen. Skytrust hätte hier also keine Möglichkeit, Dateien im Ganzen zu verschlüsseln, sondern jeweils nur Teile daraus.

Wenngleich das Umleiten aller Verwendungen der XMLHttpRequest-API entsprechende Uploads an Online-Dienste erkennen kann, ist damit keine allgemein einsetzbare Lösung geschaffen. Für letztere müssten vielmehr all jene Schnittstellen überschrieben werden, die für einen Daten-Upload zum Speicherdienst eingesetzt werden könnten. Selbstverständlich bedingt dies, dass es dafür nicht reicht, JavaScript-Operationen umzuleiten, da eine Datei im einfachsten Fall auch über einen POST-Request (ohne JavaScript) an einen Server gesendet werden kann.

4.2. Verschlüsselung von Daten vor Offline-Speicherung

Modernen Web-Browsern stehen zur lokalen Ablage von Daten von AnwenderInnen prinzipiell die mit HTML5 spezifierte Schnittstellen LocalStorage[9] (WebStorage) bzw. SessionStorage[10] zur Verfügung. Während die erstgenannte Schnittstelle die Ablage von Daten auch über längere Zeiträume ermöglicht, beschränkt sich die Verfügbarkeit von Daten im SessionStorage auf die Dauer einer Browser-Sitzung. In der Praxis werden diese neuen Technologien von Webseiten eingesetzt, um Dateien oder Informationen im Allgemeinen im Speicher des Browsers abzulegen und bei Bedarf wieder abzurufen. Ein besonderer Schutzmechanismus um diese Daten vor illegitimen Zugriff, etwa auf Dateisystemebene, zu schützen, wird dabei nicht angewandt. Konzeptionell betrachtet, dient der Browser-Speicher somit nicht zur Ablage sensibler Daten.

Um die von einer Webseite im lokalen Speicher des Browsers abzulegenden Daten zu verschlüsseln, injiziert die Browser-Erweiterung eigene Implementierungen der genannten Schnittstellen in jede aufgerufene Webseite. Der Effekt dieser Maßnahme ist, dass jeder von einer Webseite getätigte Aufruf der Schnittstellen nicht zu den originalen, vom Browser bereitgestellten, geht sondern zu selbst entwickelten Wrapper-Konstrukten, die vor die eigentliche Ablage im Speicher des Browsers geschaltet werden.

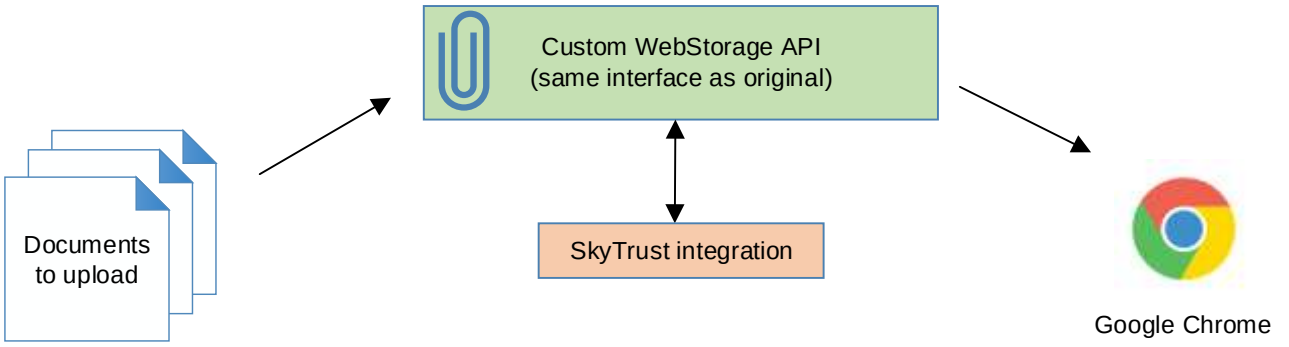

Abbildung 3. Offline-Speicherung mit Umleitung über SkyTrust

Damit der beschriebene Mechanismus zur Umleitung funktioniert, müssen die eigenen Implementierungen der WebStorage API hierarchisch über jenen liegen, die vom Browser nativ bereitgestellt werden. Konkret muss die Browser-Erweiterung somit sicherstellen, dass bei Verwendung der WebStorage API durch eine Webseite zwingend eine Verschlüsselung durch SkyTrust passiert. Um dies zu gewährleisten, müssen die eigenen Implementierungen in jede Webseite injiziert werden, die geladen wird. An dieser Stelle nimmt jedoch das Sicherheitsmodell

[9] https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API
[10] https://developer.mozilla.org/de/docs/Web/API/Window/sessionStorage

der Erweiterung unter Google Chrome eine entscheidende Rolle ein. Über „Content Scripts“ ist es möglich, eigene Inhalte in geladene Webseiten einzufügen. Einer Browser-Erweiterung ist es daraufhin aber nicht(!) möglich, auf den geänderten DOM einer geladenen Seite zuzugreifen. Praktisch wirkt sich das dahingehend aus, dass das „Content Script“ nicht imstande wäre, die Ablage einer Datei zu erkennen, da es zwar in die geladene Webseite eingefügt wurde, jedoch in einem anderen DOM läuft. Durch die Sicherheitsbeschränkungen des Browsers hat also eine Implementierung der WebStorage API, die als „Content Script“ in eine Webseite injiziert ist, keine Auswirkung auf die in der Webseite verfügbare API. Praktisch bedeutet das, dass das „Content Script“ zwar in eine beliebige Webseite injiziert werden kann, jedoch in sich geschlossen steht und von AnwenderInnen im Browser abgelegte Daten nicht entgegennehmen kann.

Um diesem Missstand dennoch beizukommen und sicherzustellen, dass in jedem Fall die eigenen Implementierungen der WebStorage APIs geladen werden, behilft sich die Browser-Erweiterung eines alternativen Weges. Anstatt die WebStorage APIs über „Content Scripts“ zu injizieren, werden die eigenen Implementierungen direkt als JavaScript-Blöcke in eine ladende Webseite eingefügt. Dieser Prozess passiert zu einem Zeitpunkt, wo der DOM der Webseite noch nicht vollständig geladen ist bzw. der Browser das Event „DOMContentLoaded“ noch nicht gesendet hat. Das direkte Hinzufügen der Implementierungen beim head-HTML-Element führt außerdem dazu, dass die eigenen APIs dem JavaScript-Code einer Webseite unmittelbar zur Verfügung stehen und nicht zuerst auf die API des Browsers zugegriffen wird.

Eine breite Verwendbarkeit dieser Lösung wird dadurch gewährleistet, dass die WebStorage APIs in allen modernen Browsern lauffähig sind und momentan (abgesehen von Cookies) die einzigen verfügbaren Mechanismen sind, um Daten lokal im Browser abzulegen. Durch die Bereitstellung einer eigenen Implementierung dieser Schnittstellen, versehen mit Verschlüsselungs- bzw. Entschlüsselungsfunktionalität beim Schreib- bzw. Lesezugriff, wird eine transparente Integration in bestehende Web-Anwendungen ermöglicht.

4.3. Anmeldung bei SkyTrust

Im Rahmen dieses Projekts sollte ferner festgestellt werden, ob Authentifizierungs-Informationen wie Benutzername und Passwort, die für den Zugriff auf SkyTrust von der Browser-Erweiterung benötigt werden, von einer geladenen Webseite hinreichend getrennt werden können.

Diese Fragestellung ließ sich durch eine geeignete Strukturierung des Plugins, entsprechend dem Chrome-Sicherheitsmodell, beantworten. Konkret heißt das, dass der Modaldialog, der einen Benutzernamen und Passwort bei SkyTrust von AnwenderInnen entgegen nimmt, nicht im gleichen DOM ausgeführt werden darf, wie etwa die eigenen Implementierungen der WebStorage APIs. Wird hier anstelle dessen ein Dialogfenster als „Content Script“ injiziert, besitzt es einen eigenen Browser-DOM und es kann von einer geladenen Webseite nicht einseitig zugegriffen werden. Mögliche Angriffsvektoren wie Cross-Site-Scripting (XSS) oder Cross-Site-Request-Forgery (CSRF) verfehlen somit ihre Wirkung. Eine Bereitstellung des SkyTrust-Logins über ein „Background script“ würde im Hinblick auf die genannten Sicherheitsaspekte ebenso probate Alternative darstellen. Jedoch würde der Login dann nur einmal – beim initialen Laden der Erweiterung im Browser – aufgerufen werden. Da der Zugriff auf WebStorage APIs von unterschiedlichen Webseiten seitens SkyTrust jedoch unterschiedliche Authentifizierungsmechanismen benötigen könnten, verbleibt das Einfügen als „Content Script“ als einziger praxistauglicher Mechanismus.

Ebenso wurde ursprünglich ins Auge gefasst, die SkyTrust-Authentifizierung in ein „Panel“ auszulagern, anstelle eines Modaldialogs. Bei Google Chrome bedeutet dies lediglich einen visuellen Unterschied: ein Panel befindet sich in der Symbolleiste des Browsers und muss vom Benutzer aktiv geöffnet werden um eine Anzeige von Informationen zu veranlassen. Da der Login-Dialog zu SkyTrust jedoch immer angezeigt werden sollte, wenn eine Authentifizierung von AnwenderInnen vonnöten ist, war dies keine Alternative.

Insgesamt wird die Anmeldung vor illegitimen Zugriff also insofern geschützt, indem die Erweiterung strikt dem vom Chrome bereits geeigneten Sicherheitsmodell entspricht und der Anmeldedialog nicht in den gleichen DOM eingefügt wird, den eine geladene Webseite verwendet. Die tatschliche

Sicherheit des Logins somit abhängig von der In-sich-Geschlossenheit des DOMs, welche wiederum von der Ausführungs-Sandbox des Browsers sichergestellt wird.

5. Fazit

Im Zuge dieses Projekts wurde ein Konzept erarbeitet und prototypisch umgesetzt, anhand dessen eine Browser-Erweiterung für Google Chrome die Daten von AnwenderInnen verschlüsselt, ohne, dass eine bestehende Web-Applikation dazu geändert werden müsste.

Neben dem Aufbau von Wissen über das Sicherheitsmodell von Erweiterungen in Google Chrome hat sich auch gezeigt, welche Auswirkungen die Verwendung von unterschiedlichen Document Object Models (DOM) bei der Verknüpfung bestehender Web-Applikationen mit zusätzlicher API-Funktionalität hat.

Darüber hinaus wurde im Zuge dieses Projekts festgestellt, dass sich Browser-Schnittstellen durch eigene Implementierungen überschreiben lassen um damit eigene Ziele zu verfolgen. Unter Einhaltung des gebotenen Sicherheitsmodells der Erweiterung wurde dabei mit Sorgfalt eine Trennung von sensiblen Login-Daten und Plaintext-Informationen von verschlüsselten nach außen gesandten Daten erreicht.

Trotz der Verwendung neuer Schnittstellungen (Reflect API) hat sich herausgestellt, dass es gegenwärtig nicht möglich ist, eine allgemein verwendbare Lösung umzusetzen, sodass sämtliche Anfragen an Online-Speicherdienste vorzeitig umgeleitet werden und durch eine Plattform verschlüsselt werden können. Nichts desto trotz ist es gelungen, nativ ausgelieferte WebStorage APIs des Browsers so zu überschreiben, dass das gesetzte Ziel zumindest für jene Daten erreicht wird, die im Offline-Speicher des Browsers abgelegt werden. Diese Implementierungen können von existierenden Web-Applikationen eingesetzt werden, ohne dass dabei am Quellcode der Webseiten etwas angepasst werden müsste.

Bei diesem Projekt wurde der Fokus ausschließlich auf die eingangs gesetzten Ziele gelegt. Weitere Faktoren wie die Performance der durchgeführten Verschlüsselung oder die Kommunikation mit SkyTrust standen nicht im Vordergrund. Für einen produktiven Einsatz der vorgestellten Lösung müssten natürlich auch diese und womöglich weitere Aspekte in Betracht gezogen werden.