A browser-based digital signing solution over the web

Fotis Loukos[1], Charalampos Tsipizidis[2], Dimitris Daskopoulos[3]

[1]Aristotle University of Thessaloniki IT Center, PO Box 888, EL 541 24, fotisl@it.auth.gr
[2]Aristotle University of Thessaloniki IT Center, PO Box 888, EL 541 24, tsipizic@it.auth.gr
[3]Aristotle University of Thessaloniki IT Center, PO Box 888, EL 541 24, dimitris@it.auth.gr

Keywords
Digital Signing, Document and Grade Signing, Certificates, Web Extensions, Native Messaging API.

1. INTRODUCTION

The ability to digitally sign documents or pieces of information for submission over the web is an area of work that concerns all major organizations and governments. Educational institutions have the typical needs of document signing, as well as assuring that the grade submission process and long term archival of grades is not exposed to risk. Typical solutions so far have involved hardware cryptographic devices that guarantee a higher level of security for safekeeping of the signer's personal certificate. This makes it harder for web browser applications to access these non-standard devices for signing and most existing solutions resort to Java applets loading proprietary libraries. With Java being plagued by so many vulnerabilities, maintaining a working, let alone secure, setup on end-user machines has become a major challenge for organizations. Furthermore, Oracle has already announced that the Java Plugin for browsers will be phased out in the near future (Oracle, 2016).

2. PROPOSED SOLUTION

In order to overcome the aforementioned problems, a pure Javascript solution should be deployed. Currently, Javascript supports the Web Crypto API (Sleevi & Watson, 2014) which allows cryptographic operations such as hashing, encrypting and signing. However, to ensure the privacy of the end users, the Web Crypto API does not support cryptographic operations using cryptographic devices which can be used to identify the user. Thus, a different approach is required.

Google Chrome Native Messaging (Google, 2016) is a technology which allows developers to create native applications, called *Native Messaging Hosts*, which can be executed by the Google Chrome browser, and communicate using a message passing interface with Google Chrome extensions and apps. Hence, a native application which uses the PKCS11 API (RSA Laboratories, 2004) can be used to sign any message in cooperation with an extension that implements the communications with the browser.

2.1. Architecture

The proposed solution consists of two major components, the Native Messaging Host and the Native Messaging Application. The Native Messaging Host is the OS native program that uses the cryptographic interface to communicate with a hardware cryptographic device and digitally sign the payload. On the other hand, the Native Messaging Application is a pure Javascript implementation running in the web browser having the form of a Web Extension Plugin. It initiates the signing process, by passing the payload to the Native Messaging Host and retrieves the results.

2.2. The Native Messaging Host

The Native Messaging Host is launched by the Web Extension Plugin and receives a JSON formatted message at its standard input. The message consists of a payload to be signed and a number of different options. Currently, the host can accept messages in various encodings, such as base64, which allows users to sign arbitrary binary data, and can apply a hash function to the input message. By loading any known PKCS11 libraries, it searches for attached hardware devices and signs the payload when it finds an appropriate certificate.

With multiplatform support being a prerequisite, the program should be written in a portable scripting language, and Python 3 was the authors' programming language of choice.

In order to interface with PKCS11 libraries, the PyKCS11 Python module (Amato & Rousseau, 2016) was used. It is based on the SWIG framework and implements a Python wrapper around any standard PKCS11 library. It has been successfully tested with the Gemalto Classic Client and the Safenet eToken Client under Windows (x86 and x64 architectures), Linux (x86 and x64 architectures) and Mac OS X.

2.3. The Native Messaging Application (Browser Plugin)

The Native Messaging Application initiates the signing procedure when the user triggers a DOM element inside a specific webpage. The extension then crafts a JSON message containing parts of the webpage to be signed and forwards it to the Native Messaging Host. The extension also registers a callback function, triggered when the Native Messaging Host responds, to embed the signed data back into the webpage and inform the user of the successful signing. After the end of the signing process the data can be sent to the web server for storage.

The Google Chrome extension is available for installation through the Google Chrome Web store. Users can be guided to either preinstall the Plugin, or the webpage can prompt installation by clicking at an element in the related website (inline installation). The latter makes the procedure more transparent.

3. CONCLUSION

We have implemented a low maintenance solution for digital signing based on personal certificates stored on hardware devices. So far, this solution is only available to Google Chrome users. Similar functionality will be made available to Mozilla users with the WebExtensions API which is expected to be released in Q2 2016. The installation procedure is simple and is not affected by the security updates of Java or other third-party software. It allows organizations with a large employee basis to function on wide scale digital signing projects with the maximum of security, that is personally stored digital certs.

4. REFERENCES

Amato, G. & Rousseau, L., *PyKCS11 - PKCS#11 Wrapper for Python*. Retrieved February 18 2016, from: https://bitbucket.org/PyKCS11/pykcs11.

Google (2016). *Native* Messaging. Retrieved February 18 2016, from: https://developer.chrome.com/extensions/nativeMessaging.

Oracle (2016). *Moving to a Plugin-Free Web*. Retrieved February 18 2016, from: https://blogs.oracle.com/java-platform-group/entry/moving_to_a_plugin_free.

RSA Laboratories (2004), *PKCS #11 v2.20: Cryptographic Token Interface Standard*. Public Key Cryptography Standards PKCS#11-v2.20.

Sleevi, R., & Watson M. (2014), *Web Cryptography API*, W3C.

5. AUTHORS' BIOGRAPHIES

Dr Fotis Loukos is working for the Computing Infrastructure and Systems Administration Department of the IT Center of Aristotle University of Thessaloniki where he focuses on Network and Information Security. He is the PKI Administrator of the Hellenic Academic and Research Institutions Certification Authority (HARICA), a Certification Services Provider included in most major Root CA programs (Mozilla, Microsoft, Apple) and in the EU TSL. He is also an external contractor for the European Network Information Security Agency (ENISA). He holds a Doctor of Philosophy in Computer Science from the Department of Informatics of the Aristotle University of

Thessaloniki. His interests include Network and Information Security, Cryptography and Public Key Infrastructures.
https://gr.linkedin.com/in/fotisloukos

Charalampos Tsipizidis works as a software developer at the IT Center of the Aristotle University of Thessaloniki. His work experience and interests lie in web conferencing systems (both Flash and WebRTC based), web development with numerous PHP frameworks, Linux system administration, load balancing and information security.

https://gr.linkedin.com/pub/charalampos-tsipizidis/3a/822/2b7

Dimitris Daskopoulos holds a Master's degree in Computer Engineering from Penn State University. He works at the IT Center of the Aristotle University of Thessaloniki, as Head of the Services Unit, a line of work he still finds immersive after 20 years. His interests are in web development, authentication/authorization, and DevOps practices for university applications.