Narrative of Digital Signature Technology and Moving Forward

Galoh Rashidah Haron, Nor Izyani Daud, Moesfa Soeheila Mohamad
MIMOS
Technology Park Malaysia, Kuala Lumpur, Malaysia

Abstract

The paper presents the technology curves in developing a digital signature solution for the web environment. The solution enables a user to perform an electronic version of a digital signature using web extension technologies. On the quest for data integrity and authenticity, data is digitally signed, and a digital signature is generated with the assistance of software or hardware-based token on a web browser. The technology curves identified are (1) web extensions survivability and advancement, (2) web browsers compatibility, and (3) digital certificates issuance. The paper explains how these technology curves have impacted the decision on the architecture and design of the solution during the development and deployment. In the final, a digital signature ecosystem which is based on a client and server technology is successfully released, which includes a web signature script to simplify the digital signature as a service.

1. Introduction

In digital security, the solution for data integrity is a hash algorithm. For data authenticity, it is a digital signature. A user who performs digital signature, assure that the data has integrity, the data is authenticated and originated from a valid user. For data security, technologies such as digital certificates, security devices, hash, and digital signature algorithm are integrated. Today, with the maturity of technologies (20 years), the development of digital signature solution should be simple and feasible. However, based on the latest development experiences, it revealed technical and integration complexities. These complexities may further assert the study of low adoption and the lack of the digital signature application [1]. This paper explains the issues and challenges in integrating the technologies, with the introduction of technologies existence and competitive survival in the digital world.

Digital certificate technology emerged as a solution to instil trust in the internet transaction. The purpose is to verify user identity in a web site. The standard for issuing certificate is published in 1998 by International Telecommunication Union – Telecommunication (ITU-T) [2]. Security devices such as smart cards and USB tokens, allow users to store a certificate inside the devices. As per today, a user has an option to purchase a certificate and a security device, perform an application installation of the security device and securely log in to a domain site that deployed a certificate-based authentication mechanism.

For web browser technology, it must provide a connection to the security device and read the certificate from the security device. The standard is stated in 'PKCS#11: Cryptographic Token Interface Standard' by RSA Laboratories in 1995 [3]. It is a guide for defining a generic interface such as application programming interface (API), for the security device. With the standard, a Cryptographic Service Provider (CSP) library is built for Microsoft web browser. A PKCS#11 library is made ready for Mozilla web browser. These libraries provide secure access to the private key for authentication, signing and manage the handling of security devices context. For saving the certificate, web browsers must provide storage for the certificate. At present, a majority of web browsers equipped with user's and server's certificates storage. 'Certificate Manager' is a graphic user interface available in the web browsers for the import and export of the certificates. This feature is delivered as the core security functionalities of the web browser, to support mutual SSL (Secure Sockets Layer) authentication.

For cryptography technology, Microsoft provides a Crypto API (CryptoAPI) library [15] that enables the integration of cryptography and security for Microsoft-based application. Mozilla provides a Network Security Services (NSS) library [14] that responsible for all cryptography and security standard for Mozilla-based application. For the developers, it means, cryptographic functions that relate to signing, which includes connecting to secure device, read and store the certificate to the certificate manager on Mozilla Firefox requires API from NSS library. For other web browsers, the integration requires API from CryptoAPI library.

Web extension is an application that resides on the web browser. It extends functionalities of the web browser. It is a terminology, which initially referred to ActiveX and Netscape Plug-in technologies. The development of web extension provides a function to sign data on the web browser digitally. The function calls a set of API from CryptoAPI and NSS as in Figure 1. The development allows binding of security devices and secures cryptographic services in the web

as provided in the 'Cryptographic Token Provider' layer. The top two layers which are 'Web Browser' and 'Web Extension' are the applications and technologies layers for the digital signature solution. Other layers are pre-existed with the installation of the security device's application, web browsers, and Windows operating system.

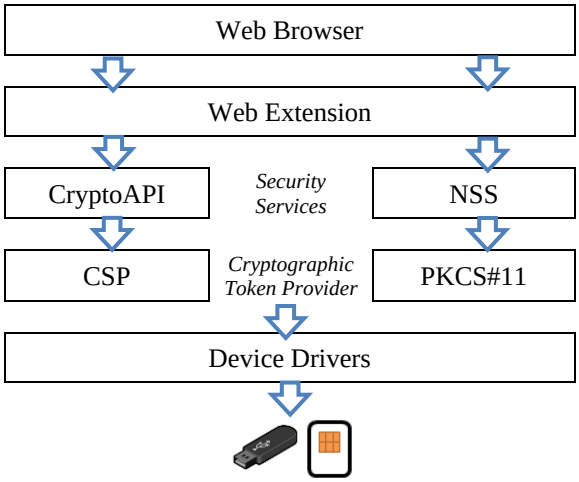

Figure 1: Architecture of Digital Signature Solution

2. Requirements

The first requirement is the digital signature must be generated using a private key with the matching of a public key in the user's certificate. The certificate must be listed in the certificate manager of the web browser. The certificate will be presented to a user prior to digital signing. The user is required to confirm the ownership of the certificate by providing a personal identification number (PIN) to access the private key. The certificate is stored inside a security device or hardware-based token (PKCS#11) or structured as a software-based token (PKCS#12). For a hardware-based token, the application of the security device will automate the import the certificate. For a software-based token, a user is required to import the certificate manually. With success of the import, the certificate will be able for viewing in the certificate manager of the web browser.

2.1 Mutual SSL Authentication

The second requirement is the digital signing must be accomplished with a mutual SSL authentication support. The SSL authentication allows the client and server, to authenticate each other at the same time. Both will provide a valid certificate for identification purpose and for the client to access the protected service. It means that user certificate must reside inside the web browser (client). The user certificate is used for both authentication and digital signing. When the digital signing service is invoked, the session of the authentication is active, and the user's certificate is loaded from the web browser.

2.2 Cross-platform Web Browser

The third design requirement is the digital signature must be available in all cross-platform web browsers. This includes Internet Explorer, Google Chrome and Mozilla Firefox. As a result, a dedicated web extension for each web browser is built. Web extension for Internet Explorer is using ActiveX technology. For Google Chrome, technology applied is a Chrome extension and native messaging. For Mozilla Firefox the technology is 'Add-on'. These web extension technologies are built using multiple languages; C++, JavaScript, and combination thereof.

2.3. Identical Core Functionalities Across all Web Browser

The functionalities of the digital signature solution are described in three phases, data before signing, data while signing and data after signing. Prior to sign, the functionalities are to present the data to be signed to a user, hash the data to sign using the SHA-256 algorithm and add a button to sign the data. While signing, the functionalities are to prompt a list of user certificates available on the web browser, view the content of the selected certificate and present a PIN dialogue for the user to access the private key and compute the digital signature using RSA 2048-bit key operation. After signing, the functionalities are to present the signature and the status of the digital signature. The format of the digital signature is a raw, PKCS#1 padded digital signature. Verification of the digital signature is assisted with a user's public key and result in a cryptographic hash value. This hash value will then be compared to the hash value of the data to sign. The signature verification is achieved without the presence of hardware-based tokens.

The prementioned core functionalities are required to be uniformly featured in all web browsers. Hence, to support cross-platform web browsers, the codes of web extension for each of web browsers, must provide identical functions, inputs, and outputs. The five main functions across all web browsers are performing sign on a hash value, get signature value, get user certificate value, get status of digital signing and get a web extension version.

3. Technology Curves

3.1 Web Extension Technology

Web extension has the ability to call native binary code through scripting code in the web browser. For

the digital signature solution, JavaScript is calling either a native binary or scripting codes that connect to CryptoAPI and NSS libraries. In the last two decades, the web extension technology has evolved and curved its own survival and compatible lifecycle. Technologies such as ActiveX, Java Applet, Adobe Flash and Netscape Plugin Application Programming Interface (NPAPI) are among the pioneers of web extension. The challenge of the web extension is the development is in silos due to the technology and framework dependencies to the web browser.

For Internet Explorer, the web extension is called ActiveX technology. It is first released in 1996 **Error! Reference source not found.** and designated for Internet Explorer. Microsoft has provided a digital signature solution called as CAPICOM **Error! Reference source not found.**. It is a module to sign data and to verify a digital signature. With few lines of scripting codes in .NET framework, a user can access private key stored by CryptoAPI and perform generation and verification of PKCS#7 based digital signature. In 2011, Microsoft decided to discontinue the CAPICOM. This leaves developers with no option, other than to develop its own proprietary digital signature solution.

The digital signature solution developed for Internet Explorer is using ActiveX technology with C++ as its primary language. The development involves CryptoAPI library and requires code sign to secure the distribution. The installation of the solution is as a series of steps, advised by an offline installer. It is automated which includes the secure registration of the ActiveX. However, for security purpose, a user manual intervention is required to approve the usage of the ActiveX by clicking the "OK" button on the site. A user requires to set the site as a trusted site with a medium security level. Missing these steps lead to the inability of a user to proceed with the digital signing and may result in a usability issue. A similar experience is observed when users engage with ActiveX technology **Error! Reference source not found.**. In the study, users tend to agree on either secure or non-secure ActiveX installation blindly, and it creates a bad computing practice and web accessibility crisis.

In Google Chrome, the first pipeline of web extension being introduced to developers is a Netscape Plugin Application Programming Interface (NPAPI) **Error! Reference source not found.**. In 2015, for Chrome version 45, Google decided to remove all NPAPI support plugin permanently. Google introduced Chrome Extension to access the operating system features. The Chrome Extension for the digital signature solution is using C++ language and JavaScript. The Chrome extension is the first development experience in understanding the architecture of web extension with native messaging. The web extension calls CryptoAPI library and shares user certificate information with Microsoft certificate manager. With one time of importing certificate, the certificate information is available in both Google Chrome and Microsoft web browser. For installation and management of chrome extension, Google provides chrome web store to ensure the trustworthy of chrome extensions.

For Mozilla Firefox, the web extension is first based on Netscape Plugin Application Programming Interface (NPAPI). It is an application programming interface (API) that allows a plugin development in C++ language. It first developed for Netscape browsers, starting in 1995 with Netscape Navigator 2.0. In 2015, add-on technology was introduced for Mozilla Firefox. It provides a set of simple API that allows developers to enhance the functionalities of the web browser. Mozilla provides an add-on SDK which allows calls of NSS API from JavaScript. NSS API calls the native NSS library to access the private key in the hardware token for signing purposes. Since the development is in JavaScript, the deployment of the add-on version of the digital signature solution is preferable than the ActiveX and Chrome extension. There is no offline installer as the add-on is listed online as a trusted add-on in the Mozilla Add-ons (AMO).

Implementation of the digital signature solution, albeit using different technologies for multiple web browsers, each produces an average of 1255 lines of codes. In details, for ActiveX with implementation in C++ language, produces a total of 1175 lines of codes. For Chrome extension written with C++ and JavaScript languages, produces a total of 1484 lines of code. For Add-On written in JavaScript, the total line of codes is 1105.

3.2 Web Browser Technology

A new version of web browser leads to possibilities of a new version of web extension and new policies for the web browser. For example, in the development span of two years, Mozilla Firefox has started with version 38 and ended with version 57. The version releases considered as rapid development and based on the bugs fixed in the Mozilla web browser; the web extension has completed five iterations of a new release. The new policies have a direct impact on the architecture and the design of the web extension. The following is the real cases derived from the new release and the new policy of web browser.

3.2.1. New Release. Mozilla released a compatibility issue bug number 1241646 for Mozilla Firefox version 47. In this issue, Mozilla removed unused token arguments from 'nsIX509CertDB' function which is used to list the user certificates. The issue leads to halt the add-on when it is executed. This issue required codes changes and resulted in a new version of the add-on. Mozilla released a compatibility issue

bug number 1284946 for Mozilla Firefox version 50. In this issue, Mozilla has removed three functionalities in the NSS library, which are 'nsIX509Cert.getUsagesArray', 'requestUsagesArrayAsync', and 'getUsagesString', which is used to view the content and key usage of the certificate. The issue leads to halt the add-on when it is executed. The content 'Certificate Key Usage' as in Figure 2, is removed. This issue required codes changes and resulted in a new version of the add-on.

Mozilla released a compatibility issue bug numbered 857627 for Mozilla Firefox version 53. In this issue, Mozilla has advised the developer, not to expose the NSS certificate nickname API in the Personal Security Manager (PSM) interfaces. Based on the bug, the certificate nickname as shown in Figure 2 has been updated to "Sarah Othman" which is a common name from the user's certificate, as shown in Figure 3. This issue required codes changes and resulted in a new version of the add-on.

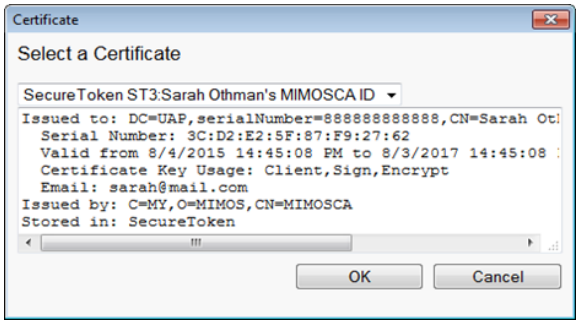

Figure 2 : Certificate Key Usage

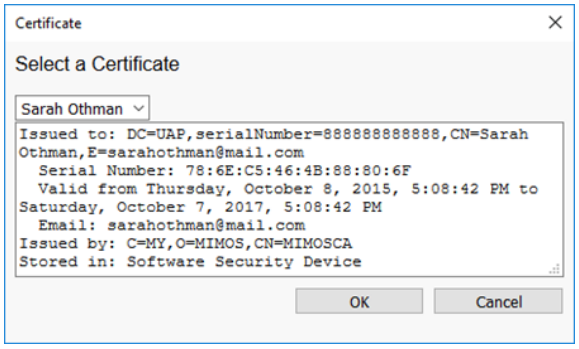

Figure 3 : Certificate Nickname

Every new version of add-on requires signing by Mozilla. The signing process requires the add-on to be uploaded. The signing time is varied and dependent on the number of add-ons in a queue. Thus, as new developers, it is essential to prepare the stability of the digital signature solution for every new release of the web browser by leveraging a web development platform which is Mozilla Firefox Developer Edition.

3.2.2. New Policy. The availability of the web extension technology is dependent on policies set for the web browser. For Internet Explorer, the ActiveX technology has been standing for 22 years. In July 2015, the first version of the digital signature solution (ActiveX) was first released. For the last two years of its released, minimal changes are performed, since there is no rapid development performed by Microsoft. ActiveX is the longest standing web extension technology. However, ActiveX is not supported in Microsoft Edge. In 2017, Microsoft announced the alternative to ActiveX, which are Microsoft Edge extension with native messaging **Error! Reference source not found.**. The announcement shall open a new development phase for developing the digital signature, dedicated to Microsoft Edge.

For Mozilla Firefox, the policy set has impacted the architecture and development of the digital signature solution. In July 2015, the first version of the digital signature solution (add-on) was distributed using an offline installer. Mozilla states a new policy where it is required for the add-on to be signed by Mozilla and to be distributed through a listing at the Mozilla Add-ons (AMO). This policy is to protect end-users, where a security review is required to determine the safety level of the add-ons **Error! Reference source not found.**. The impact of this policy is, for every new version release, a dedicated slot of time is required for Mozilla to sign the add-on. Based on the experience, signing time is varied, and it can stretch until two to three weeks' time.

In June 2016, Mozilla introduced multi-process Firefox (e10s). Add-on that is not compatible with e10s is considered as a legacy, and it will be no longer supported in the latest version of Mozilla Firefox. This required efforts to undergo a learning development and to figure out how to support multi-process operation in Mozilla Firefox. In April 2017, Mozilla decided to phase out the add-on technology. Mozilla introduced web extension technology with native messaging. In elapsed of two years, Mozilla team is actively changing the web extension technology to better performance and security, which result in to shift of architecture, design and user interface of the digital signature solution, writing from a purely JavaScript to a hybrid of C++ and JavaScript language.

3.3 Digital Certificate Technology

A user obtains a certificate from a trusted certificate authority (CA). The certificate binds user information such as name and email address for usage of digital identity. For an optimal solution, the digital signing requires to support any certificate generated by the multiple certificate authorities. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (RFC5280) is the standard for generating user certificate **Error! Reference source not found.**. With the user

certificate conforms to the standard of RFC5280, the assumption is user certificate, and the digital signature solution will be universal compatibility. However, based on the development experience, there are cases that showed adverse effects.

3.3.1. Missing Netscape Certificate Type for Digital Signing. In a deployment phase, a production certificate is reported to be successful in digital signing on Google Chrome but failed on Mozilla Firefox. The debug effort is focused on the certificate as in Figure 4. In the analysis, the certificate has an additional new field that is 'Netscape Cert Type' with a value of SSL Client Authentication (80). Next is to investigate the impact of 'Netscape Cert Type' field to Mozilla Firefox. The NSS technical note **Error! Reference source not found.** states, for digital signing, there is two type of certificates which are 'certUsageSSLClient' and 'CertUsageEmailSigner'. The first is responsible for SSL client authentication and the second is responsible in verifying S/MIME email signatures. The current digital signature solution is to support both SSL authentication and data signing. Thus, the usage should embed both functions, which are 'certUsageSSLClient' and 'CertUsageEmailSigner'. Missing the second type causes errors in performing the signing.

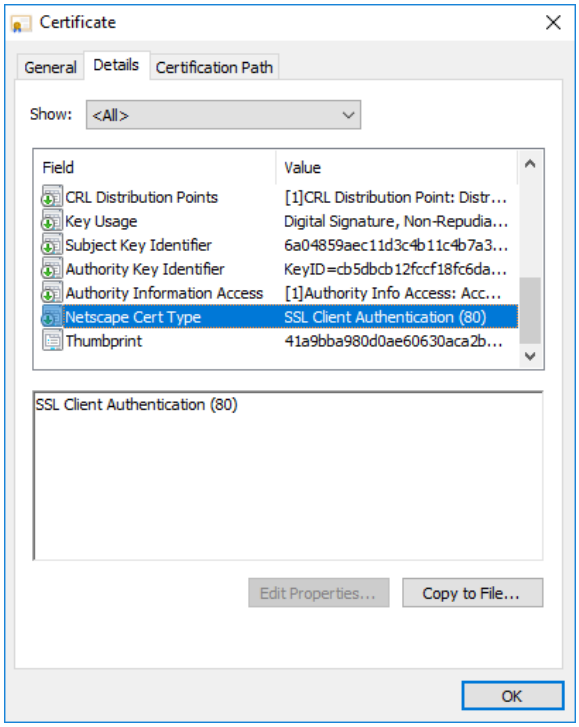

Figure 4: Certificate with Mismanage of Key Usage

In addition, based on NSS codes as in Figure 5, 'Netscape Cert Type' field is translated as 'requiredCertType'. For the digital signature purpose, the 'requiredCertType' is set as NS_CERT_TYPE_EMAIL with 'requiredKeyUsage' is set as non-repudiation key.

```
1156        switch ( usage ) {
1157          case certUsageSSLClient:
1158            /*
1159             * RFC 5280 lists digitalSignature and keyAgreement for
1160             * id-kp-clientAuth.  NSS does not support the *_fixed_dh and
1161             * *_fixed_ecdh client certificate types.
1162             */
1163            requiredKeyUsage = KU_DIGITAL_SIGNATURE;
1164            requiredCertType = NS_CERT_TYPE_SSL_CLIENT;
1165            break;
                .
                .
                .
1179          case certUsageEmailSigner:
1180            requiredKeyUsage = KU_DIGITAL_SIGNATURE_OR_NON_REPUDIATION;
1181            requiredCertType = NS_CERT_TYPE_EMAIL;
1182            break;
```

Figure 5 : NSS Code (nss/lib/certdb/certdb.c, 1115)

The certificate has 'Netscape Cert Type' for NS_CERT_TYPE_SSL_CLIENT, which purposes of performing authentication and not for digital signature. Thus, the solution is, to either generate a new batch certificate with additional 'Netscape Cert Type' as NS_CERT_TYPE_EMAIL or to remove the Netscape Certificate Type field. The latter solution is preferred, as the Netscape Certificate Type field is deprecated and a non-standard extension, and being replaced by 'basicConstraints', 'keyUsage' and 'extended key usage' extensions **Error! Reference source not found.**.

3.3.2. Missing CKA Label. It is reported that the certificate has failed in performing digital signature and authentication for Mozilla Firefox. The debug effort started with the ability for the certificate to be displayed in the Certificate Manager for Mozilla Firefox. The analysis shows that the certificate appears on the "Others" tab instead of "Your Certificate" tab as in Figure 6. In order for the certificate to be successful in digital signing and authentication, it requires for the certificate to be appeared in "Your Certificate" tab.

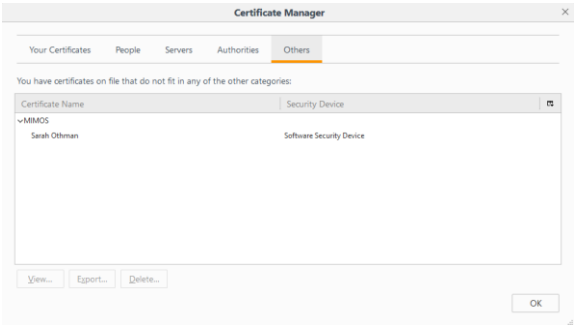

Figure 6 : Missing CKA_Label in Certificate Manager

Based on the analysis of the certificate structure, if the certificate contains an empty CKA_LABEL, then the certificate nickname will be empty. Thus, the NSS library will treat the certificate as an unknown

certificate and show up in the "Others" tab rather than "Your Certificate" tab. The certificate will not be accepted as a personal certificate. The information of CKA_LABEL is extracted from NSS FAQ **Error! Reference source not found.**, it describes CKA_LABEL as an attribute to identify user certificates and present label to the user. User certificate must have a label associated with to user. Each token label should be unique and meaningful to the user, and that each certificate label should be unique to the token.

3.3.3. Other Issues Based on the development and deployment experience, a small number of faulty certificates produced by the Certificate Authorities (CA). Examples, URL for the certificate revocation list (CRL) is expired or unavailable, and generation of public and private keys for a certificate is a mismatch. The certificate structures are different by CA, which means the software to generate certificates is written by multiple software products. However, in one case, the CA claim the current certificates is not defective, since it can perform signing by using a third-party application which is written in Java libraries and integrated with Java Applet technology. Since it is in Java, it has no compatibility issue with web browsers. Nevertheless, Java Applet is not preferable since it is no longer be supported in the new web browsers. In fact, it breaks the cycle of mutual SSL authentication, and it is not leveraging the functionalities of the certificate manager and cryptographic libraries that existed in the web browser.

4. New Architecture and Design for Mozilla Firefox

Mozilla Firefox has enabled multi-process operations and phased out the add-on technology. Mozilla recommends adopting the new web extension and native messaging for the replacement of the add-on technology. The first drafting of the architecture is to connect the new web extension and native messaging to the NSS library. However, there are user certificates which released in the production with the issues of missing 'Netscape Cert Type' field. These user certificates are having difficulties in calling the NSS libraries for signing function. In contrast, the same user certificates function well with a non-Mozilla web browser, i.e. Internet Explorer and Google Chrome which opted for CryptoAPI and CSP libraries.

The reason being is, the field of 'Netscape Cert Type' is not a sensitive field to CryptoAPI library. With the prementioned factors, the decision is to revamp the design of digital signature solution from add-on technology to extension and native messaging technology. It involved the decision to drop the NSS dan PKCS#11 libraries and to replace with the CryptoAPI and CSP libraries. Thus, in order to support Firefox 58 onwards, the new architecture is as shown in Figure 7.

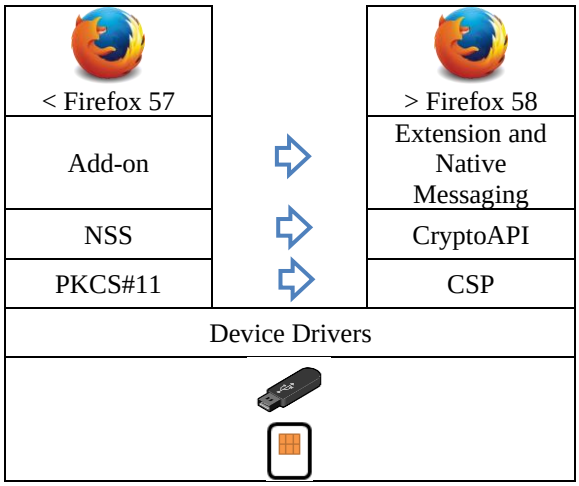

Figure 7: New Web Extension Architecture for Mozilla Firefox

5. Digital Signature Ecosystem

With web extension technologies are built in silos, the integration of web application to the digital signature as a service demands for simplicity. The first problem is the digital signature operation on multiple web extensions technologies generates an unstructured format of data and digital signature. The integration requires a web application to provide functionalities to receive unstructured data from multiple web browsers and to format the unstructured data to a structured data that is independent on the type of web browser. The web application may be a third-party software vendor who requires to provide a digital signature as a service. However, requesting for the third-party software vendor to manage the integration code may impede the delivery of the digital signature solution.

The second problem is inevitable code changes in web extension, forces code changes in the web application that provides the digital signature as a service. For example, if there is a new request to support time stamping features and leads to the upgrade of the web extension version, the web application requires to add the new time stamping parameter, manage the time stamping data and supports the latest web extension version. The code changes occur in multiple places due to the support of multiple web browsers. If multiple software vendors adopt the digital signature solution, the codes changes must reflect on all code of software vendors immediately.

The third problem is the chaos of web extension installation in the client machine. Each of the web extension has its installation policy. Each of the web

extension has its installation files; C++ and JavaScript. For web security purposes, the web extension should be selected from the web browser extension store and manually install by users. In a recent development, Google Chrome browser version 73 introduces a new group policy that enables an administrator of an organisation to configure and control instances of Google Chrome Browser [16]. This policy results in a seamless installation of the browser extension, where it is configured to automate the installation of browser extension to the user. However, it seems to be unethical to install a web extension without user's permission.

In solving the prementioned issues, a general JavaScript called 'Web Signature Script' lays in between the web browser and web extension layers as illustrated in Figure 8 is created. The purpose is to simplify the digital signature integration. The script is responsible for structuring data and the signature in JSON format. The JSON format provides the content secured messages with information of algorithm and digital signatures. The JSON format acts as a container to present the data to sign and the digital signature and built-in JavaScript language.

The JSON format contains four important fields. The first field is the **payload** which describes as the value of the original message to be signed. The second field is the **protected** which contains the algorithm used to generate the signature. The supported signing algorithms are RSA with hash algorithm SHA256, and ECDSA with hash algorithm SHA256. The protected fields contain the expiry of the data and the signature timestamp. The third field is the **header** which contains user certificate signing and the certificate chain. The last field is the **signature** which contain the signature of the digital signing. All information is in Base64 format. Despite multiple selection of web browsers, the web application will receive the JSON signature data format upon user successfully perform digital signing.

The script manages the data format of each specific web extension for each of the web browser. Any additional of new features which requires new parameter will be added in the script. The script provides an abstraction API for the integration of third-party software vendor to perform digital signing and verifying digital signatures based on the JSON format. The API provides uniform signing and verification functions for all platforms and browsers. The third-party software vendors will only require including the web signature script in the web application as JavaScript and perform either signing or verification of signature with additional two lines of codes. In case of a new updated version for the web extension, the only requirement for the web application developers is to replace the web signature script with the latest version. Thus, the script should offer minimal changes or effort for web application developers.

For the digital signature solution, to support the installation of web extensions in multiple web browser, a standalone installer is created. The installer is set to cater to the automatic installation process, supports for manual installation of the web extensions from the browser store and requires a user manual agreement to set required security policy prior for installation.

In summary, the digital signature ecosystem is built based on client and server technology. The client is a cross-platform web browser extension and a minified Web Signature Script, which performs signing of data, generates and verify the digital signature on the web browser. In the future, the extension and the native messaging framework will be implemented for all web browser as in Figure 8. The cryptographic libraries will be CryptoAPI and CSP. The server is a Java library, which performs online verification of the digital signature and the certificate.

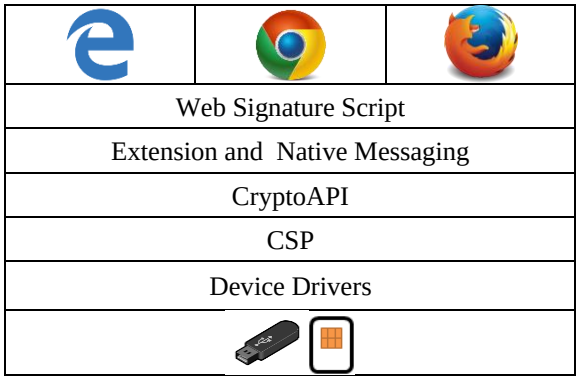

Figure 8: Future Architecture of Digital Signature Solution

6. Conclusion

The development of the digital signature solution for web environment is completed. The solution allows users to perform a digital signing on any data, to preserve the integrity of the data and to identify the authors of the data. It supports SHA-256 hash algorithm. It supports software and hardware cryptographic tokens. The development effort is made by understanding the signing mechanism and the integrated technologies that built the ecosystem. The signing mechanism is embraced through direct calls of application programming interface provided in NSS and CryptoAPI libraries, and the knowledge of web extension technology. The integrated technologies which are web extensions, web browsers and the issuance of the digital certificate are the underlying triggering factors that have impacted the architecture and design of the final solution.

7. References

[1] H. Saripan and Z. Hamin, "Digital signature as a 'blue ocean'?: An analysis of the application of its law in Internet banking," 2010 International Conference on Science and Social Research (CSSR 2010), Kuala Lumpur, Malaysia, 2010

[2] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of ITU-T Recommendation X.509, "Information Technology – Open Systems Interconnection - The Directory: Authentication Framework," 1988.

[3] "PKCS#11: Cryptographic Token Interface Standard", 1995. [Online]. Available: https://www.cryptsoft.com/pkcs11doc/STANDARD/pkcs-11.pdf. [Accessed: 08 Sep. 2018].

[4] "RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", 2008. [Online]. Available: https://tools.ietf.org/html/rfc5280. [Accessed: 08 Sep. 2018].

[5] A. Guha, M. Fredrikson, B. Livshits and N. Swamy, "Verified Security for Browser Extensions," 2011 IEEE Symposium on Security and Privacy, Berkeley, CA, 2011, pp. 115-130.

[6] "NSS Tech Note3", 2002. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/NSS_Tech_Notes/nss_tech_note3. [Accessed: 21 Aug. 2018].

[7] OpenSSL Foundation Inc., "x509v3_config", 1999. [Online]. Available: https://www.openssl.org/docs/man-master/man5/x509v3_config.html. [Accessed: 05 Sep. 2018].

[8] "PKCS11 FAQ", 2014. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/PKCS11/FAQ. [Accessed: 03 Sep. 2018].

[9] "ActiveX Controls", 2018. [Online]. Available: https://docs.microsoft.com/en-us/cpp/mfc/activex-controls?view=vs-2017. [Accessed: 19 Aug. 2018].

[10] "Extensions - Native messaging - Microsoft Edge Development", 2017. [Online]. Available: https://docs.microsoft.com/en-us/microsoft-dge/extensions/guides/native-messaging. [Accessed: 22 Aug. 2018].

[11] Michael Howard and David E. Leblanc. 2002. Writing Secure Code (2nd ed.). Microsoft Press, Redmond, WA, USA.

[12] H. M. Park, "The Web Accessibility Crisis of the Korea's Electronic Government: Fatal Consequences of the Digital Signature Law and Public Key Certificate," 2012 45th Hawaii International Conference on System Sciences, Maui, HI, 2012

[13] "NPAPI deprecation: developer guide - The Chromium Projects", 2014. [Online]. Available: http://www.chromium.org/developers/npapi-deprecation. [Accessed: 16 Aug. 2018].

[14] "Network Security Services", 2005. [Online]. Available: http://www.mozilla.org/projects/security/pki/nss/. [Accessed: 10 Jul. 2018].

[15] "Using the Microsoft CryptoAPI", 2006. [Online]. Available:http://msdn2.microsoft.com/en-us/library/aa266944.aspx. [Accessed: 19 Sep. 2018]

[16] "The Chromium Projects", 2019. [Online]. Available: https://www.chromium.org/administrators/policy-list-3. [Accessed: 10 April. 2019]

8. Acknowledgements

We acknowledge the support provided by the Ministry of Energy, Science, Technology, Environment and Climate Change (MESTECC) in funding security project through Eleventh Malaysia Plan (11MP). The completion of the project allowed implementation of the digital signature solution as one of the technologies in gearing security initiatives.

