Using the W3C WebCrypto API for document signing

Nick Hofstede and Nick Van den Bleeken

Inventive Designers, Sint Bernardsesteenweg 552, 2660 Antwerp, Belgium,
https://www.inventivedesigners.com/

Abstract. This paper focusses on digitally signing documents as a specific use case for making secure hardware available to a web application. We explore the current options available to implementers and the drawbacks associated with each option. Then we look at the emerging Web Cryptography API developed by the W3C and discover missing functionality needed to implement this use case. Finally, we suggest a way to extend the API in order to support digitally signing documents using secure hardware.

Keywords: web applications, secure hardware, Web Cryptography API, W3C

1 Use Case

Converting paper processes to digital ones is an obvious trend within organisations. Documents are stored electronically and processes involving paperwork are transitioned to web applications. Not only is this more ecological, it enables a lot of opportunities to increase productivity and reduce costs. The past decade the paperless office has been inching ever closer to reality.

Not all paper documents are so easily replaced by their digital equivalent however. After digitizing the low-hanging fruit, attention now turns to processes that are more difficult to digitize. One of the common hurdles involves signatures. Often they can be replaced by auditing the web applications and using your corporate account to identify yourself, but some documents really do require a legally binding signature. Whereas on paper this involves taking a pen and making a scribble, the digital equivalent is often more complex. Web applications capable of generating documents digitally signed by the user are needed.

1.1 Signing documents

The European Union recognizes three levels of digital signatures [1]: electronic signatures like scanned in handwritten signatures or email footers, advanced electronic signatures which are created using modern cryptography standards and qualified signatures which are advanced electronic signatures satisfying additional requirements. This last level is an advanced electronic signature made

using a key which has a qualified certificate associated with it. Qualified certificates can only be issued by certified certificate authorities and are only issued using the most stringent processes and only to the most secure type of keys. In return, qualified signatures must be considered equivalent to handwritten signatures by a judge whereas other electronic signatures are open to interpretation based on the actual details. The peace of mind this obligation provides makes them a requirement often encountered.

The requirement that a key should be kept under the sole control of the owner for qualified certification is interpreted to mean the key should be embedded in a hardware device. This to make sure undetected copying is impossible. In order to create a qualified signature, a European citizen will therefore always need a smart card or key dongle. A web application creating digital signatures should therefore be able to access the secure hardware of the user.

1.2 Electronic identity cards

The requirement for secure hardware prompted several European countries to issue electronic identity cards to their citizens. Currently countries like Belgium, Germany and many others provide electronic identity cards capable of creating electronic signatures [2–10] and more countries are planning on issuing them.

These cards, many of them issued mandatory, create a large group of users with access to secure hardware and a certificate ready to create legally binding electronic signatures. Governments have begun using these cards in web applications for the retrieval of official documents like birth certificates (after authentication) or submitting online tax forms (requiring a signature). Many more use cases for government and corporate applications can be thought of, but in this paper we will be focussing on signing documents.

2 Current situation

Using these identity cards in web applications today is possible, but serious drawbacks exist.

2.1 Client-side TLS

The first applications that popped up were using client-side TLS for authentication. The user navigates to a website and is asked to authenticate herself. Most if not all of the identity cards come with middleware that either adds the certificate to the operating system's key ring allowing it to be accessed by browsers, or installs specific browser plug-ins registering the certificate for client-side use.

After the user has authenticated herself, consent with the content to be submitted to the application (like a tax form for example) is typically given by pushing a button. Proof of this consent is then carried by the auditing information logged by the web application. No real electronic signature is created and consent can only be inferred from the audit logs of the web application.

Even this simple scheme can lead to problems when one is not careful. In particular, revocation checks often need to be explicitly enabled at the SSL termination point as it is typically disabled by default. Additionally, client-side authentication suffers from a few other problems like not being able to log out of a session and somewhat bad usability due to it being needed before a connection to the server is established and a web page providing guidance can be shown. While this can be worked around using different domains, it would be preferable if the authentication process is initiated by the web page itself to allow for a richer user experience.

In any case, no actual qualified electronic signature is created, only an audit trail.

2.2 Java plugin

If a qualified electronic signature needs to be created to sign a PDF document, or to create an archive containing a XAdES signature for example, the browser needs to access the smart card. This can be done using a custom plug-in, but a better option is to use a more widely available general purpose plug-in for this task. Chances are the user already has this plug-in installed which avoids going through an additional installation procedure. Using javascript and a Java applet, data can be shipped to and signed by the secure hardware. Typically, one of two methods is used. Either the applet probes for a library installed as part of the card's middleware using JNI, or the smart card is accessed directly using APDU commands available in standard java runtimes starting from version 6 [11].

The Java plug-in needs to be installed and the applet needs to be granted additional privileges. This is cumbersome, but until recently it was a reasonable approach. After a series of vulnerabilities in Java however, browsers either disable the plug-in, or only allow it to run when it is the latest version and has been given explicit permission. This degrades the user experience so much that using an applet is no longer a viable option.

3 Evaluation of the Web Cryptography API

Mainly driven by mobile applications being implemented as web applications, more and more functionality, including access to hardware like GPS sensors [12] and cameras [13], has been made available through javascript API's. When we first heard about a Web Cryptography API [14] being under development we looked at it to remove the dependency on a Java applet to access the secure hardware device and do the signing. With a use case like "The ability to select credentials and sign statements can be necessary to perform high-value transactions such as those involved in finance, corporate security, and identity-related claims about personal data." in the working group's charter [15] as a goal this did not seem far-fetched at all.

3.1 Design

In order to keep the scope of the API limited, the Working group has defined a very narrow scope for its main document. In order to stay away from concepts that are not portable between operating systems, cryptographic libraries or user agent implementations, provisioning operations or the discovery of cryptographic modules is considered out of scope.

While this might seem like a severe restriction, by supporting key generation functions it still allows for many cryptographic use cases. Indeed, as long as the keys are generated by the api, the provided operations (`encrypt`, `decrypt`, `sign`, `verify`, `digest`, `deriveKey`, `importKey` and `exportKey`) allow for a wide array of applications like secure messaging, data integrity protection of cached data and cloud storage.

On the more practical side of designing the API, the working group follows JavaScript best practices and tries to make every call that might take some time, or that might conceivably require user interaction to be asynchronous. This allows for the program flow of the javascript application to continue while waiting for the user the grant permissions, enter a pin number or calculations to be completed.

3.2 Signing

The key types and sign operations supported by the API are suited for the use case we have in mind. The issue has been raised whether "broken" cryptography algorithms like SHA1 and PKCS#1 v1.15 should be included, but for the sake of backwards compatibility and integration with server-side software it has been decided they remain. While we hope the world quickly moves on to the more modern alternatives which are supported as well, given the number of deployed smartcards implementing only these older algorithms, excluding them would seriously limit the applicability of the API today.

3.3 Key discovery

Recognizing that not all use cases work with keys generated by the application itself, the group started work on key discovery in a separate working draft [16]. In order to limit the scope and driven by a use case focussing on trusted platform modules (TPMs) and digital rights management (DRM), the specification currently limits itself to "discovering named, origin-specific pre-provisioned cryptographic keys for use with the Web Cryptography API". While this allows for user agents to make keys stored on secure elements like TPMs available to web applications originating from a given domain under a known name, none of those adjectives are a good match for the signing use case.

While it might be possible to assign names to the keys provided by the secure hardware, in all naming schemes we could come up with it would be impossible for the web application to guess what that name might be. Additionally, the keys contained in the secure hardware wouldn't be origin-specific either. A user shall

want to use her smart card to authenticate herself to different web applications. Finally, the keys are not pre-provisioned. A way to prompt the user to (re)insert her smart card or USB dongle would be needed.

Key discovery as currently conceived by the WebCrypto Key Discovery draft isn't useful for discovering keys that reside on users' smart cards.

3.4 Certificate based discovery

Instead of name-based key discovery we believe attribute-based discovery of keys based on the key's algorithm or the accompanying certificate is necessary. By limiting the keys known to the browser by algorithm, issuing certificate authority, intended usage and other relevant properties, a short list could be presented to the user where she can pick the key needed to complete the action she started.

While this operation closely resembles selecting the appropriate key to use when setting up client-side TLS, it should be noted that this operation can be made asynchronous and can be initiated after a page has been presented to the user. It is conceivable that the user prompt takes the same form as other requests for access to specific resources like your location or camera. The main difference might be that this wouldn't have an "Allow" and "Disallow" button, but a "Select..." one popping up a dialog requiring further interaction.

4 Proposed extension

Despite its current shortcomings, we believe the Web Cryptography API forms a solid basis. We joined the working group to help shape the API and make the use case outlined in this paper possible. We're currently working on a proposal that will extend the current API to enable the creation of web applications that use secure hardware for the creation of digital signatures.

4.1 Additional API

We propose a new `X509Certificate` class, and two new asynchronous methods. A first one to search for certificates and related keys, and a second one to enable exporting certificates to a byte array.

The `X509CertificateSelector` will take a dictionary containing filters. Filters include things like issuing certificate authority, usage flags, key algorithm, validity dates and others.

Invoking the `X509CertificateSelector` will create a subset of all known certificates known to the browser and initiate a selection procedure by the user agent. This procedure can start with a subtle banner to request access like the location or media capture API's. When clicked, the subset can be presented as a list to the user. Confronted with the list, the user will pick the certificate appropriate for the action she started and grant the web application access to the associated keys. This grant is origin-specific. When the keys are used an additional dialog window prompting for a pin code may be shown.

By keeping the user in the loop and requiring her to explicitly allow access to the certificate and keys stored in the operating system's store, this API can't be used to fingerprint users or glance information from unrelated certificates stored in the store.

The export functionality is necessary because the certificate (or certificates as you probably need the entire chain) used will likely have to be embedded in or associated with the digital signature that is being created.

4.2 Prototype

We are implementing enough of the WebCrypto API and the proposed extensions as a browser plug-in to validate this proposal [20]. The code is available under an Apache license on github [21].

Note This is an initial proposal and not all issues have been discovered or indeed resolved. We welcome comments, insights and code contributions you may have or want to share.

References

1. European Parliament and Council. Directive 1999/93/EC on a Community framework for electronic signatures. 13 December 1999. Retrieved from http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31999L0093:EN:NOT
2. Austria. The Austrian electronic ID Card "Bürgerkarte". Retrieved from http://www.buergerkarte.at/
3. Belgium. The Belgian electronic ID Card. Retrieved from http://eid.belgium.be/
4. Estonia. The Estonian eID Card, "EstEID". Retrieved from http://www.id.ee/
5. Finland. The Finnish eID Card, "'FINEID'. Retrieved from http://www.fineid.fi/
6. Germany. The German electronic ID Card, "Personalausweis". Retrieved from http://www.personalausweisportal.de/
7. Italy. Carta di Identit Elettronica (C.I.E), the Italian electronic ID Card. Retrieved from http://www.halnet.it/cie/
8. Portugal. The Portuguese electronic ID Card, "Carto de Cidado". Retrieved from http://www.cartaodecidadao.pt/
9. Spain. The Spanish eID Card. Retrieved from http://www.dnielectronico.es/
10. Sweden. Fakta om nationellt id-kort (Facts about the national ID card). Retrieved from http://www.polisen.se/inter/nodeid=33378&pageversion=1.jsp
11. Oracle. Java Smart Card I/O API. Javadoc, package specification. Retrieved from http://docs.oracle.com/javase/6/docs/jre/api/security/smartcardio/spec/
12. Geolocation API Specification W3C Proposed Recommendation 10 May 2012 Retrieved from http://www.w3.org/TR/geolocation-API/
13. HTML Media Capture W3C Last Call Working Draft 26 March 2013 Retrieved from http://dev.w3.org/2009/dap/camera/
14. World Wide Web Consortium. Web Cryptography API. W3C Working Draft 8 January 2013. Retrieved from http://www.w3.org/TR/2013/WD-WebCryptoAPI-20130108/
15. World Wide Web Consortium. Web Cryptography Working Group Charter. 3 April 2013. Retrieved from http://www.w3.org/2011/11/webcryptography-charter.html

16. World Wide Web Consortium. WebCrypto Key Discovery. W3C Working Draft 08 January 2013. Retrieved from http://www.w3.org/TR/2013/WD-webcrypto-key-discovery-20130108/
17. Cryptographic Token Interface Standard. Version 2.30, Editor Simon McMahon with Robert Griffin of RSA as project coordinator. RSA PSS mechanism parameters. Retrieved from http://www.cryptsoft.com/pkcs11doc/v230/
18. Mac Developer Library. kSecInputIsDigest constant definition. Retrieved from https://developer.apple.com/library/mac/search/?q=kSecInputIsDigest
19. Windows Dev Center. CryptSignHash function documentation. Retrieved from http://msdn.microsoft.com/en-us/library/windows/desktop/aa380280
20. Inventive Designers API proposal, version 79f54d9 Retrieved from https://github.com/InventiveDesigners/webcrypto-key-certificate-discovery-js/wiki/API
21. GitHub, Inc. Inventive Designers' webcrypto-key-certificate-discovery-js repository. Retrieved from https://github.com/InventiveDesigners/webcrypto-key-certificate-discovery-js