

W3C Workshop on Identity in the Browser

The Need for a Web Security API

Submitters

Sean Turner, IETF Security Area Director
Stephen Farrell, IETF Security Area Director
Peter Saint-Andre, IETF Applications Area Director

Abstract

Today, web applications are constructed from a combination of server-side code and dynamically-downloaded client-side code (primarily HTML and JavaScript). The programming environment available to client-side JavaScript developers is provided by web browser implementations and various popular JavaScript libraries. Unfortunately, the "standard" JavaScript Application Programming Interfaces (APIs) do not include cryptographic functions. This position paper advocates creation of a native, in-browser API that will give developers access to cryptographic algorithms and other security methods already present in today's web browsers, similar to what is available to application programmers developing directly on common operating systems.

Motivations

More and more applications are moving to the "web" (i.e., HTTP, HTML, JavaScript, and the like). Developers are working within the confines of various web browsers to secure these applications, and most use Secure Sockets Layer (SSL)/Transport Security Layer (TLS) to do so. This reliance is sub-optimal for applications whose architectures are not strictly client-server (e.g., IM and VoIP). For example, for some applications there is a need to apply data-origin message-level authentication and possibly encryption to objects exchanged between the browser and other network entities. As a workaround, developers are investigating the use of JavaScript Object Notation (JSON) as a format for signed and encrypted objects. They are also working to implement various cryptographic functions directly in JavaScript libraries (which are typically stored at well-known web addresses and fetched as needed by any web application needing them). Although these approaches make some sense in an application layer security protocol, it makes less sense for web developers to roll (and deliver) their own cryptographic algorithms. Not only is this practice wasteful, it's also dangerous when the browser's security "goodies" (i.e., the cryptographic algorithms) are just an API away.

Downloading cryptographic algorithms is wasteful in terms of bandwidth used. Application and browser developers are both very interested in ensuring their applications are speedy in the eyes of users; nobody wants to lose a speed war on

CNET. If web developers end up rolling their own cryptographic algorithms and libraries to support a JSON application layer security protocol, the code may end up being downloaded during application initialization. Such cryptographic code could include message digest/hash algorithms, digital signature algorithms, content encryption algorithms, key wrap algorithms, and keyed-Hash Message Authentication Code (HMAC) algorithms. This kind of code is typically not small because of the significant math involved in producing strong security.

However, the greatest danger here is not a waste of bandwidth, but possible security breaches. Obviously, downloading cryptographic algorithms is an easy attack vector if not done over SSL/TLS. But the real challenge is that security is hard. As Steve Bellovin pointed out in RFC 5406, the design of security protocols is a subtle and difficult art. In fact, coding security protocols is even more subtle and difficult than designing security protocols. There is no doubt that some developers will get it right the first time, but there is also no doubt that some will get it wrong. Given that cryptographic functions are already implemented in browsers (and that some of them have already been evaluated by the U.S. National Institute of Standards and Technology (NIST) for compliance with Federal Information Processing Publication (FIPS PUB) 140), it seems unnecessarily risky to not make use of the cryptographic algorithms already present in the browser.

The development of such an API should involve web developers (who often desire simplicity) as well as browser security experts (who often desire resistance to attack). Although it can be difficult to balance these goals, a consistent API for access to cryptographic algorithms and related security functions would provide a strong foundation for securing the web.

Goals

We propose that a web security API would support the following algorithms and security functions:

- o Hash/message digest algorithms (e.g., SHA-256)
- o Digital signatures algorithms (e.g., RSA PKCS#1 v1.5, ECDSA)
- o Confidentiality algorithms (e.g., AES)
- o Key transport/agreement algorithms (e.g., RSA PKCS#1 v1.5, ECDH)
- o HMAC algorithms (e.g., HMAC-SHA1, HMAC-SHA256)
- o Extracting keys from TLS sessions (e.g., using RFC 5705)
- o PKI path validation (e.g., input/output of validation base64 certificate/CRL blobs and providing/receiving validation algorithm inputs/outputs)
- o Generation and processing of Cryptographic Message Syntax (CMS)
- o Generation of public/private key pairs
- o Establishment of TLS channel bindings (e.g., using RFC 5056 and RFC 5929)
- o Use of local resources to generate random (or pseudo-random) numbers of cryptographic strength