This specification is hereby put in the public domain

Combining PKI and Smart Cards with W3C's WebCrypto

Background: In many parts of the world proprietary browser "plugins" have been deployed in order to provide missing browser PKI-functionality like the ability to sign transactions etc. Although indeed working, these plugins share a number of weaknesses like:

- Highly platform-dependent making them expensive to roll out to consumers
- Typically run with the same privileges as local applications
- Requiring users to perform explicit installs which may not be allowed on enterprise-managed computers

The W3C WebCrypto [http://www.w3.org/TR/WebCryptoAPI] WG has (among numerous of other things) taken on this as a target.

The following specification outlines a foundation for creating lightweight JS/HTML5-based "plugins" having similar functionality as the proprietary plugin schemes, but without the mentioned disadvantages using an X.509-based extension mechanism.

> *This specification is essentially a souped-up version of a scheme originally proposed by Samuel Erdtman of NexusSafe.*

Preconditions

This specification depends on that the user has one or more X.509 certificates supplied in a smart card, "soft token", or in an embedded SE (Security Element). One of the many possibilities that have been mentioned is that the user in some way grants unknown code (or sites) access to specific keys. This is of course technically feasible but confronts users with questions and decisions they are less likely to understand the consequences of.

To facilitate a more *manageable* access control model, this specification builds on the current WebCrypto API specification which *indirectly* mandates that keys residing in platform-wide keystores are *inaccessible* from WebCrypto, while introducing a mechanism that enables such keys to *optionally "transcend"* through the use of a *dedicated X.509 extension holding a target domain* (or a set of domains).

Users should (as with "native" WebCrypto keys), not need to explicitly grant privileges to code or keys, but rather be able aborting the entire operation which should be a part of any well-designed application.

Of course nothing stops a browser-vendor introducing an opt-in dialog possibly including a "don't ask me again" option.

Out of Scope – Key Initialization

Provisioning keys in smart cards etc. is something entirely different to "using" keys and is out of scope for this specification.

Cross Origin Operation

A core WebCrypto feature is the reliance on SOP (Same Origin Policy) for protecting keys from unauthorized access. To facilitate cross-origin-operations **`postMessage()`** operations can be used like already is the case for native WebCrypto keys.

X.509 Domain Indicator Extension

The exact format is yet to be defined but it does only have to provide UTF8 string(s) of the target domain(s), like **`example.com`**.

Extended API

On the next page there a short description of the additional API methods required by this specification.

API Examples

The following section contains a few examples on how the WebCrypto API *could* be augmented to support the described extension scheme. Here supplied in Java-notation rather than Web-IDL.

Finding Keys

Assuming that the keys we are interested in reside in the browser/platform/system keystore *we need a platform-independent way of finding them*.

```
import window.crypto.subtle.*;

Key[] our_keys = KeyStore.enumerateKeys(KeyStore.PLATFORM);
if (our_keys.length == 0)
  {
    fail and exit…
  }
Key key = our_keys[0];     // Select a key to use, here just the first one
```

One could imagine supporting a discriminating argument like SMARTCARD or a bit-field with attributes like CONNECTED, EMBEDDED, HARDWARE etc.

The call to `enumerateKeys()` will only return keys having an X.509 domain indicator extension matching the invoking domain.

Using Keys

Now we are ready using the "regular" WebCrypto API:

```
crypto.subtle.sign(AlgorithmIdentifier, key.privateKey, Data2Sign).then (function (etc….
```

A WebCrypto implementation must distinguish between keys associated with `KeyStore` *and keys having "web-storage" since access control must be enforced for every call*

Key Attributes

Many "plugin" applications presumably need to find out various and usually *composite* key attributes like associated X.509 certificates, supported algorithms, etc. Something like the following should be appropriate:

```
KeyAttribute key_attr = key.getKeyAttribute(AttributeTypeURI);
```

Key Authorization

Some keys will require authorization by a PIN before they can be used. PINs may be gathered by the invoking application. When the user clicks OK or similar, the authorization data would be transferred to the target key (*before* invoking any standard WebCrypto operations), through the use of an additional method implied by this specification:

```
key.authorize(AuthorizationData);
```

If an application provides no explicit authorization data for a key that needs authorization, the browser should automatically request the user for key authorization through a system-specific pop-up dialog launched immediately before the actual crypto-operation is to be performed.

Privacy and Security Considerations

The described scheme doesn't appear to significantly depart privacy- or security-wise from the original WebCrypto API: *Nothing prevents malicious issuers from subjecting their clients to various attacks or misfortunes*.

Author

Anders Rundgren, anders.rundgren.net@gmail.