# W3C Web Crypto API Update

Ryan Sleevi

# Why are we talking about the W3C in the IETF?

- The W3C Web Crypto WG charter calls for active liaising with the IETF JOSE WG
- Protocols matter to application developers. RFCs do not exist in a vacuum.
- The W3C is actively soliciting feedback from the broader Internet community, particularly those skilled in applications and cryptography - like many participants in the IETF.

# Background

- W3C Working Group chartered at the end of 2011
  - Result of several community groups and workshops on identity in the browser
- Looking at the set of problems facing web developers today regarding crypto & identity, and how browsers/user agents can solve them
  - This includes the "drive-by" web, but also other use cases such as the W3C SysApps WG, with very different security models

# State of Web Crypto

- Support for TLS client certs
- (Mixed) support for <keygen> tag
  - Lets you generate an RSA key and provide POP
  - Not widely implemented
- Variety of vendor-specific extensions (CAPICOM/XEnroll, generateCRMFRequest) and plugins

# State of JS Crypto

- JSBN
- Stanford Javascript Crypto Library (SJCL)
- CryptoJS
- Closure Library ("goog.crypt")
- OpenPGP.js
- Forge
- Qooxdoo
- CryptoCat
- Mega
- ... and no doubt more

# If you can do crypto in JS, why a browser API?

# Problems with JS Crypto

- Constant time is hard
  - JS engines are constantly changing and tuning how they optimize code, leading to a variety of ever changing expectations for how code will actually run
- Entropy is hard
  - Math.random() is not secure
- Correctness is hard
  - Many JS implementations by skilled JS authors, unskilled cryptographers
  - You explicitly don't want everyone re-implementing AES from the spec. As shown repeatedly, it's quite hard to get right (c.f. constant time)

# Problems with JS Crypto

- Performance is hard
  - In JS, all numbers are 64-bit floating point
  - BigNum math in JS relies on "elegant" hacks like using only 28 bits of the 64-bits, since that's what "works"
  - Until Typed Array specification for WebGL, no good way to represent arbitrary binary data in JS. Everything was a UTF-16 character array (DOMString)

# If JS Crypto is so hard, why do people bother?

# "Web" is a broad term

- Can mean the "drive-by" web - the (untrusted) sites and origins you visit in your browser
- Can mean the extensions you install in your browser
- Can mean the Web Apps/Sys Apps you "install" in your browser
- Can mean the "apps" you install on devices like your smart TV
- Can mean apps that use the JS engines developed for/used by browsers (eg: node.js)
- And for some, can refer to just the combination of HTML + JS, with a variety of different security guarantees and problems

# What are people actually writing

- SSH (RFC 4253) clients
- S/MIME (RFC 3851) clients
- OpenPGP (RFC 4880) clients
- Chat applications (eg: CryptoCat)
- Authentication frameworks (eg: BrowserID/Persona)

# What can browsers / user agents do to make it better?

# Browser Crypto Stacks

- Browsers today already ship with support for full crypto stacks, by virtue of SSL/TLS support
- Features such as WebRTC/Rtcweb further support with DTLS/SRTP
- ... Can we expose that to JS?

# Making it better

- Provide quality entropy
  - Can provide cryptographically-strong RNGs, beyond just Math.random()
- Provide quality implementations
  - Potentially FIPS-validated implementations
  - Constant time (hopefully...)
- Provide better performance
  - Can leverages the underlying hardware (eg: AESNI, 64-bit BigNums)

# Current Approach

# Provide access to low-level algorithms

- Primitives such as AES (CTR, CBC, CMAC, GCM, CFB), SHA-1/SHA-2, RSA, DH, ECDH, ECDSA
- KDFs such as Concat (NIST SP 800-56A) and HKDF (RFC 5869), as well as password-based such as PBKDF2 (RFC 2898)

# Isn't that... dangerous?

**Yes.**

# Alternatives

- Only provide a high-level API
  - Cryptography is hard - don't encourage more of it.
  - Developers should just have simple 'encrypt' and 'decrypt' APIs - let browsers/the W3C define the secure construction.

But what does that mean?

  - Based on JOSE?
  - Based on NaCl?
  - Based on KeyCzar?
  - Based on SJCL?
  - Based on... ?

# Alternatives

- Only provide a high-level API
  - Doesn't solve the need of real developers and real applications today.
  - Security will remain a hard problem - cryptography is only part of that.
  - Developers who have no idea what they're doing will continue to do what they've always done. Most will continue using libraries and code written by people who do know what they're doing (eg: jQuery, Dojo, YUI, Prototype, MooTools, etc)
  - Poor story for the non-drive-by-web (eg: Sys Apps, extensions)

# Alternatives

- Provide a high-level API for SSH / PGP / S/MIME
  - ... Does **every user on the Internet** really want their browser to be an SSH client?

# Alternatives

- Fix Javascript/ECMAScript
  - Problem is even more broad and ill-defined as a language feature.
  - Lots of proposals for TC39 already for even broader issues than that of cryptography.
  - Possible place for a BigNum API (Performance, Constant Time)

# What about JOSE?

# JOSE

- W3C WG currently lacks consensus on how much or how little of JOSE should be used.
- W3C WG is focused on an API. JOSE is about a protocol/exchange format.
  - Very different problems and use cases.
  - High-level vs Low-level
- However, JOSE-led specs **are** being explored for:
  - Key representation/transport (import/export, wrap/unwrap)
  - May or may not make sense in the long run

# JOSE

- W3C Web Crypto WG is still discussing how the use of JWK/JWE for (wrapped) key affects other aspects of the API
  - Algorithm naming (eg: "A128-CBC" vs "AES-CBC" with a 128-bit key)
  - JOSE's MTI requirements
  - Handling key provisioning (importing integrity-protected attributes - protection of keys against application using the keys)

# Questions?