Developer's Guide to JavaScript and Web Cryptography

Kevin Hakanson

mdc2013

Important Info

#MDC13

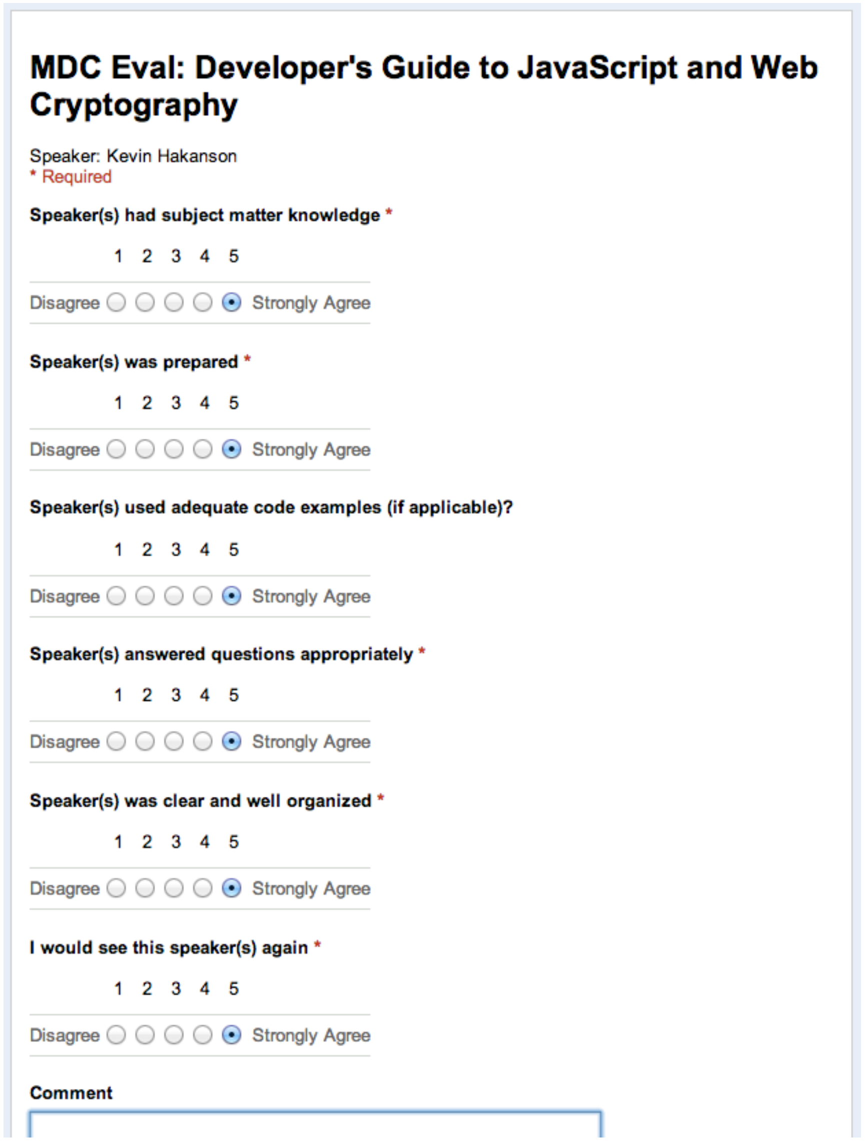
Speaker Evaluations

http://mdc.ilmservice.com/eval

Presentation Downloads

http://mdc.ilmservice.com/download

Happy Hour



MDC Eval: Developer's Guide to JavaScript and Web Cryptography

Speaker: Kevin Hakanson
* Required

Speaker(s) had subject matter knowledge *

	1	2	3	4	5	
Disagree	○	○	○	○	●	Strongly Agree

Speaker(s) was prepared *

	1	2	3	4	5	
Disagree	○	○	○	○	●	Strongly Agree

Speaker(s) used adequate code examples (if applicable)?

	1	2	3	4	5	
Disagree	○	○	○	○	●	Strongly Agree

Speaker(s) answered questions appropriately *

	1	2	3	4	5	
Disagree	○	○	○	○	●	Strongly Agree

Speaker(s) was clear and well organized *

	1	2	3	4	5	
Disagree	○	○	○	○	●	Strongly Agree

I would see this speaker(s) again *

	1	2	3	4	5	
Disagree	○	○	○	○	●	Strongly Agree

Comment

Abstract

The increasing capabilities and performance of the web platform allow for more feature-rich user experiences. How can JavaScript based applications utilize information security and cryptography principles? This session will explore the current state of JavaScript and Web Cryptography. We will review some basic concepts and definitions, discuss the role of TLS/SSL, show some working examples that apply cryptography to real-world use cases and take a peek at the upcoming W3C WebCryptoAPI. Code samples will use CryptoJS in the browser and the Node.js Crypto module on the server. An extended example will secure the popular TodoMVC project using PBKDF2 for key generation, HMAC for data integrity and AES for encryption.

(Less) Abstract

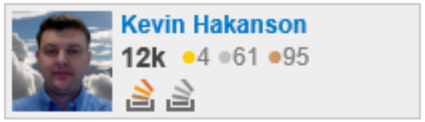
The increasing capabilities and performance of the web platform allow for more feature-rich user experiences. How can JavaScript based applications utilize information security and cryptography principles? This session will explore the current state of JavaScript and Web Cryptography. We will review some basic concepts and definitions, discuss the role of TLS/SSL, show some working examples that apply cryptography to real-world use cases and take a peek at the upcoming W3C WebCryptoAPI. Code samples will use CryptoJS in the browser and the Node.js Crypto module on the server. An extended example will secure the popular TodoMVC project using PBKDF2 for key generation, HMAC for data integrity and AES for encryption.

Kevin Hakanson

@hakanson

kevin.hakanson@gmail.com

github.com/hakanson



Kevin Hakanson
12k ●4 ●61 ●95

stackoverflow.com/users/22514/kevin-hakanson

Bio

Kevin Hakanson is an application architect for Thomson Reuters where he is focused on highly scalable web applications, especially the JavaScript and security aspects. His background includes both .NET and Java, but he is most nostalgic about Lotus Notes. He has been developing professionally since 1994 and holds a Master's degree in Software Engineering. When not staring at a computer screen, he is probably staring at another screen, either watching TV or playing video games with his family.

What to Expect

- Concepts and Definitions
  - (slides with lots of words I won't read aloud)
- Real World Examples
  - (including proper attribution and hyperlinks)
- JavaScript Code Samples
  - (some of them "forked" from the internet)
- Attempts at Humor
  - (I will laugh at my own jokes)
- Questions
  - (OK during presentation)

Technology Logos (NASCAR Style)

WS WebStorm

ANGULARJS by Google

Windows Azure

Qu QUnit js unit testing

Google

node JS™

chrome

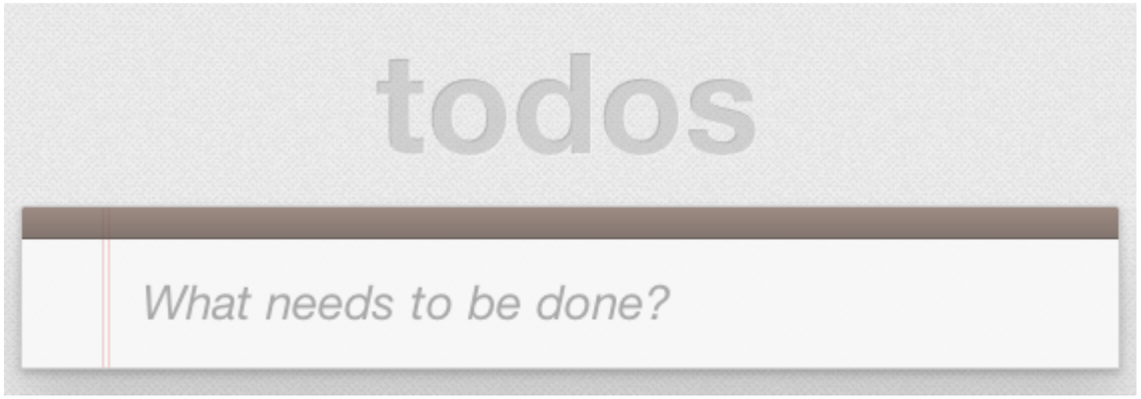
OpenSSL™ Cryptography and SSL/TLS Toolkit

TodoMVC

stackoverflow

TodoMVC

Project which offers the same Todo application implemented using MV* concepts in most of the popular JavaScript MV* frameworks of today.

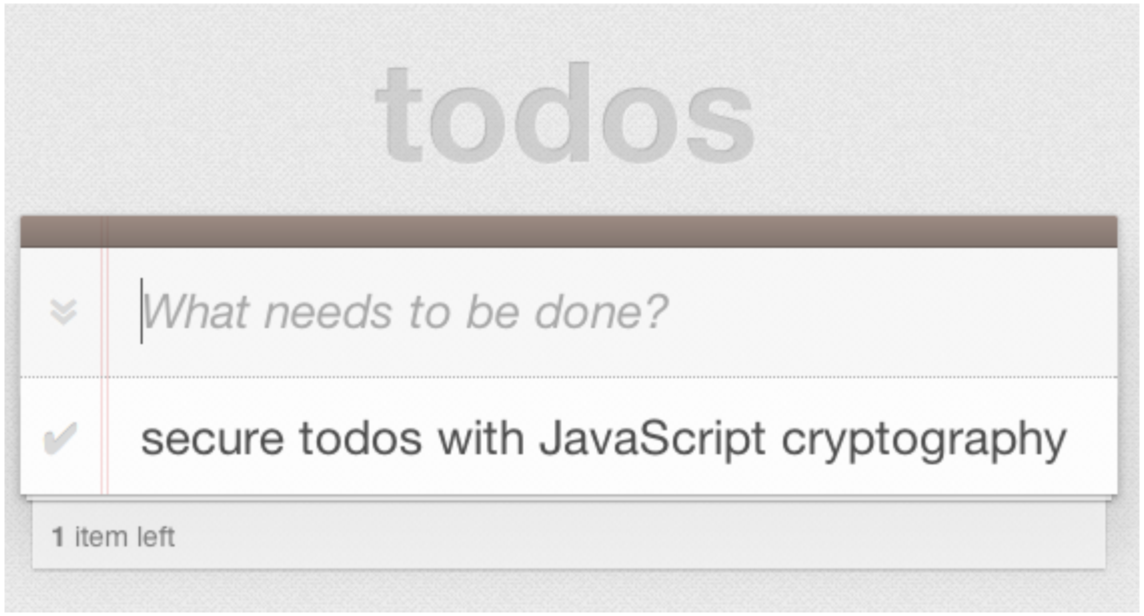


http://todomvc.com/

https://github.com/addyosmani/todomvc

Today's Session "todos'

- Review the relevant cryptography concepts.
- Use JavaScript and Web technologies to apply these cryptography concepts.



secure todos with JavaScript cryptography

1 item left

Why? TodoMVC Uses `localStorage`

Chrome keeps `localStorage` in an SQLite file:

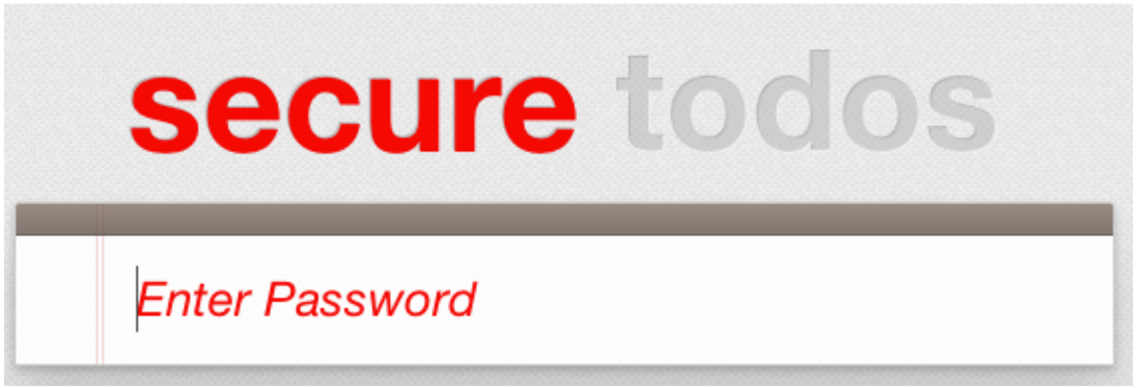
OS X: `~/Library/Application Support/Google/Chrome/Default/Local Storage`

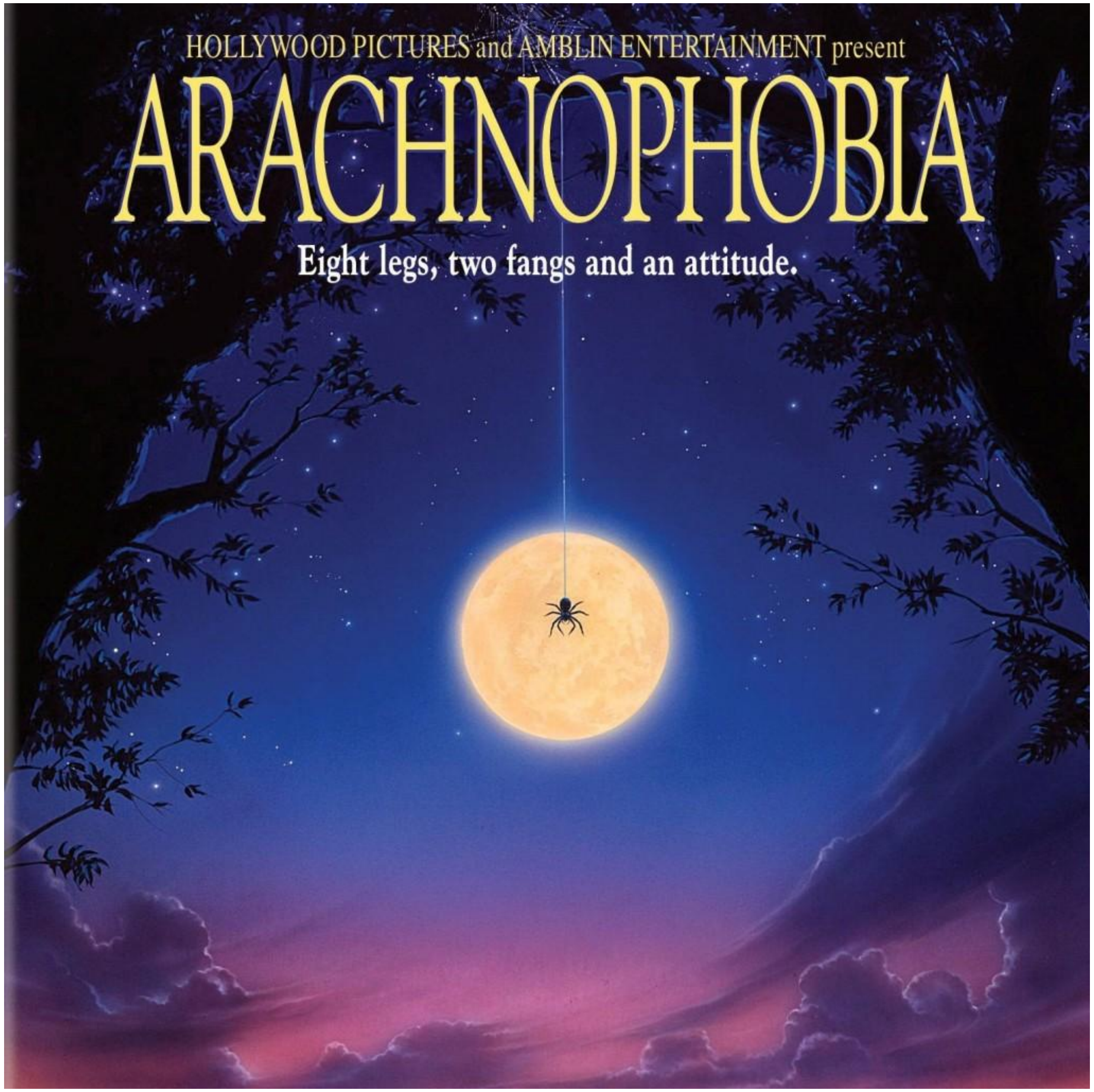
Windows: `%HOMEPATH%\AppData\Local\Google\Chrome\User Data\Default\Local Storage`

```
$ sqlite3 http_todomvc.com_0.localstorage
SQLite version 3.7.5
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> select * from ItemTable;
todos-jquery|[{"id":"b6731fa4-2e52-43c1-ae2f-
4d431f9bef75","title":"secure todos with
JavaScript cryptography","completed":false}]
```

This Is What I Want

- Enter Password before access to "todos"
- Encrypted "at rest" in `localStorage`




HOLLYWOOD PICTURES and AMBLIN ENTERTAINMENT present
ARACHNOPHOBIA
Eight legs, two fangs and an attitude.

Acronymphobia?

AES CA CBC CSRF DES
DSA ECC FIPS HMAC
ISO IV MD5 MITM NIST
OCB OWASP PBKDF2
PKCS PII PKI PRNG RC4
RSA SHA1 SSL TLS XSS

Glossary

Glossary of Key Information Security Terms (Draft)

- http://csrc.nist.gov/publications/drafts/ir-7298-rev2/nistir7298_r2_draft.pdf

- common security terms has been extracted from NIST Federal Information Processing Standards (FIPS), the Special Publication (SP) 800 series, NIST Interagency Reports (NISTIRs), and from the Committee for National Security Systems Instruction 4009 (CNSSI-4009)

NIST and FIPS

NIST - National Institute of Standards and Technology

http://www.nist.gov/index.html

FIPS - Federal Information Processing Standard

http://www.nist.gov/itl/fips.cfm

FIPS PUB - FIPS Publication

http://csrc.nist.gov/publications/PubsFIPS.html

Cryptography

The discipline that embodies principles, means, and methods for providing information security, including confidentiality, data integrity, non-repudiation, and authenticity.

SOURCE: SP 800-21

Cipher, Plaintext and Ciphertext

Cipher: Series of transformations that converts plaintext to ciphertext using the Cipher Key.

Plaintext: Data input to the Cipher or output from the Inverse Cipher.

Ciphertext: Data output from the Cipher or input to the Inverse Cipher.

SOURCE: FIPS 197

Cryptographic Key

A parameter used in conjunction with a cryptographic algorithm that determines

- the transformation of plaintext data into ciphertext data,
- the transformation of ciphertext data into plaintext data,
- a digital signature computed from data,
- the verification of a digital signature computed from data,
- an authentication code computed from data, or
- an exchange agreement of a shared secret.

SOURCE: FIPS 140-2

Secure Socket Layer (SSL)

A protocol used for protecting private information during transmission via the Internet.

Note: SSL works by using a public key to encrypt data that's transferred over the SSL connection. Most Web browsers support SSL, and many Web sites use the protocol to obtain confidential user information, such as credit card numbers. By convention, URLs that require an SSL connection start with “https:” instead of “http:.”

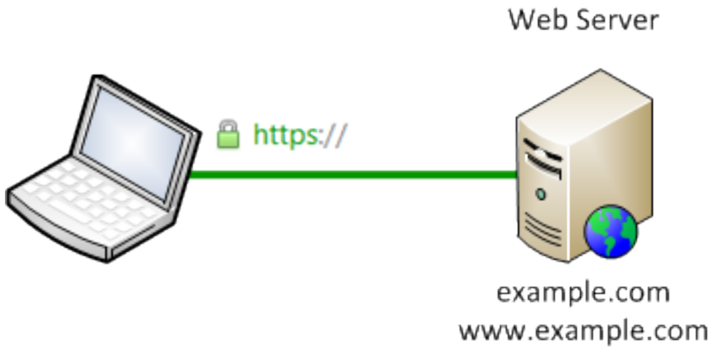
SOURCE: CNSSI-4009

TLS (not SSL)

theY MiGHt BE GIANtS
When You
PLAY IT,
SAY IT!
ISTANBUL
NOT
CONSTANTINOPLE
ISTANBUL (NOT CONSTANTINOPLE) (2:34)
(J. Kennedy, N. Simon; CHAPPELL & CO., ASCAP)
PRODUCED BY CLIVE LANGER & ALAN WINSTANLEY
LICENSED FOR PROMOTION ONLY. NOT FOR SALE.
From the Elektra album "Flood" #60907-1/2/4
Elektra
PR 8182-2
DAD
ELEKTRA ENTERTAINMENT, A Division of Warner Communications Inc., ®75 Rockefeller Plaza, New York, New York 10019, 345 North Maple Drive, Beverly Hills, California 90210
℗© 1990 Elektra Entertainment for the United States and WEA International Inc. for the world outside of the United States. All Rights Reserved. Printed in U.S.A.
Warning: Unauthorized reproduction of this recording is prohibited by Federal law and subject to criminal prosecution.

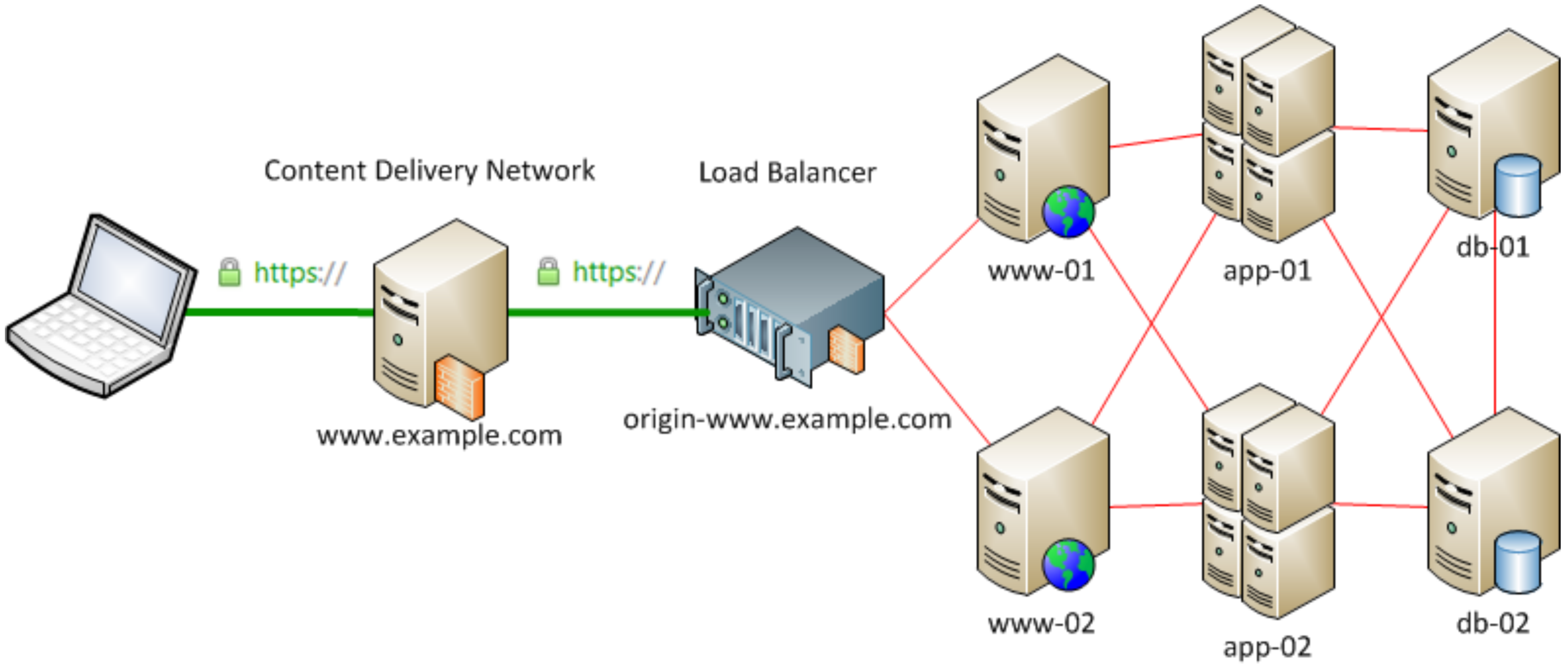
SSL "In Theory"

Client makes HTTPS connection to server.



SSL "In Practice"

Separate HTTPS connections for client/CDN and CDN/origin server; internal networks don't use HTTPS.



Web Standard?

W3C Web Cryptography API

This specification describes a JavaScript API for performing basic cryptographic operations in web applications, such as hashing, signature generation and verification, and encryption and decryption.

http://www.w3.org/TR/WebCryptoAPI/

WebCryptoAPI Charter

Primary API Features in scope are: key generation, encryption, decryption, deletion, digital signature generation and verification, hash/message authentication codes, key transport/agreement, strong random number generation, key derivation functions, and key storage and control beyond the lifetime of a single session.

http://www.w3.org/2011/11/webcryptography-charter.html

WebCryptoAPI Use Cases

- Multi-factor Authentication
- Protected Document Exchange
- Cloud Storage
- Document Signing
- Data Integrity Protection
- Secure Messaging
- Javascript Object Signing and Encryption (JOSE)

http://www.w3.org/TR/WebCryptoAPI/#use-cases

Perfect!

Web Cryptography Working Group

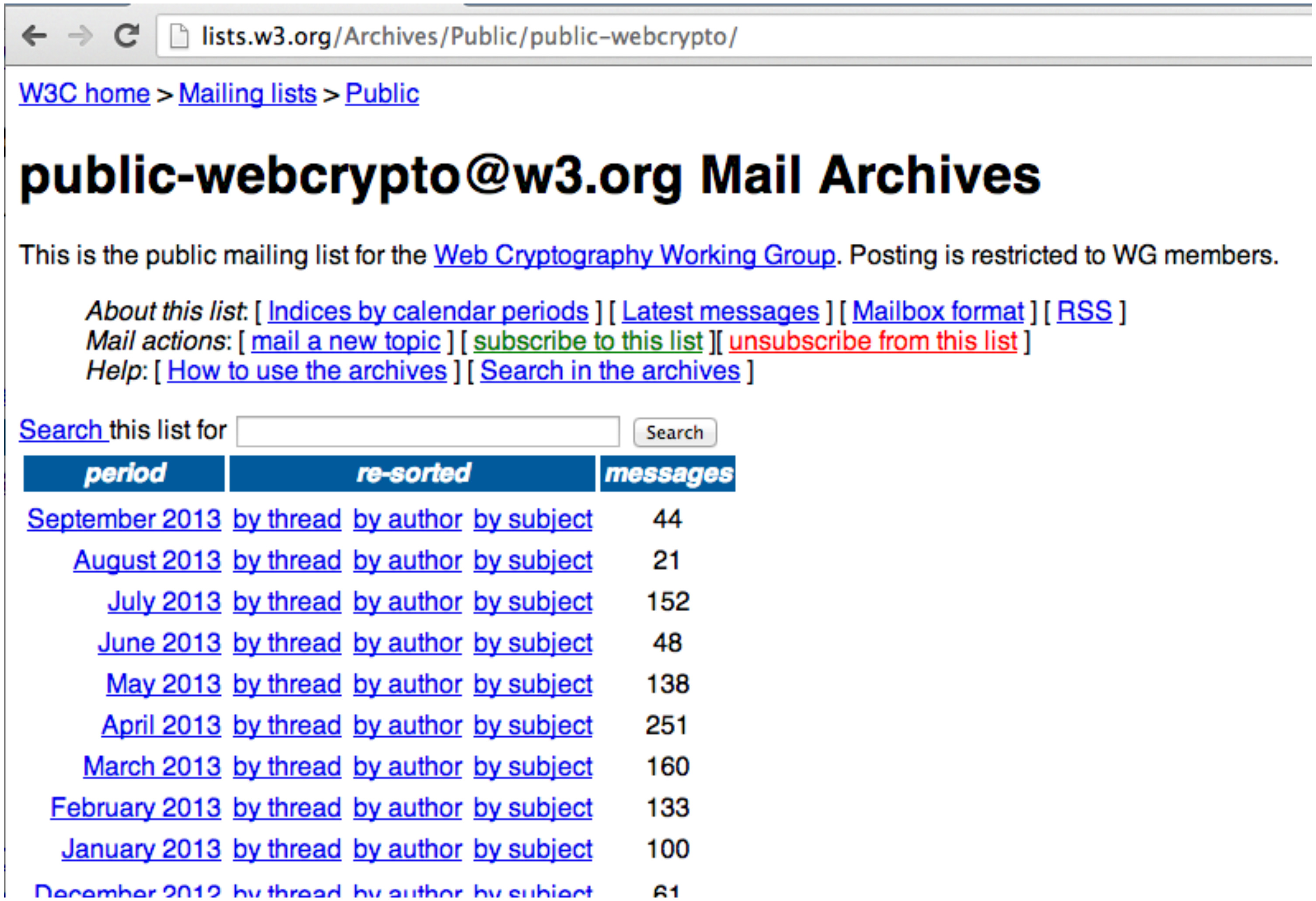
Schedule of Deliverables

- April 2012: Group Formation
- June 2012: Expected first public Working Draft of Web Cryptography API spec
- October 2013: Expected Last Call
- June 2014: Expected Candidate Recommendation
- September 2014: Expected Proposed Recommendation
- Jan 2015: Expected Recommendation

http://www.w3.org/2012/webcrypto/Overview.html

2015?   Other Options?

Participate in Mailing List



lists.w3.org/Archives/Public/public-webcrypto/

W3C home > Mailing lists > Public

public-webcrypto@w3.org Mail Archives

This is the public mailing list for the Web Cryptography Working Group. Posting is restricted to WG members.

About this list: [Indices by calendar periods] [Latest messages] [Mailbox format] [RSS]
Mail actions: [mail a new topic] [subscribe to this list][unsubscribe from this list]
Help: [How to use the archives] [Search in the archives]

Search this list for [ ] Search

period	re-sorted			messages
September 2013	by thread	by author	by subject	44
August 2013	by thread	by author	by subject	21
July 2013	by thread	by author	by subject	152
June 2013	by thread	by author	by subject	48
May 2013	by thread	by author	by subject	138
April 2013	by thread	by author	by subject	251
March 2013	by thread	by author	by subject	160
February 2013	by thread	by author	by subject	133
January 2013	by thread	by author	by subject	100
December 2012	by thread	by author	by subject	61

Require Internet Explorer 11?

- Internet Explorer 11 added support for the Web Cryptography API
  - However, based on spec before CryptoOperation changed to support TC39/DOM Promises
  - Included in Windows 8.1 or as update to Windows 7 or Windows Server 2008 http://windows.microsoft.com/en-us/internet-explorer/ie-11-release-preview
- MSDN Documentation:
  - Internet Explorer 11 Developer Guide > Privacy and security > Web Cryptography API http://msdn.microsoft.com/en-us/library/ie/dn265046(v=vs.85).aspx
  - Web applications > Web Cryptography http://msdn.microsoft.com/en-us/library/ie/dn302338(v=vs.85).aspx

Use Case: Netflix

- HTML5 Video in IE 11 on Windows 8.1
  Wednesday, June 26, 2013

  "future of premium video on the web, since they allow playback of premium video directly in the browser without the need to install plugins"

  "Microsoft implemented the Web Cryptography API (WebCrypto) in Internet Explorer, which allows us to encrypt and decrypt communication between our JavaScript application and the Netflix servers."

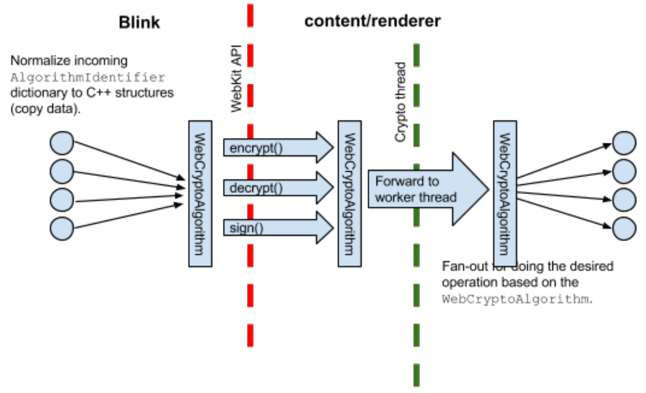
  http://techblog.netflix.com/2013/06/html5-video-in-ie-11-on-windows-81.html

Contribute to Chromium?

Open-source project behind Google Chrome.

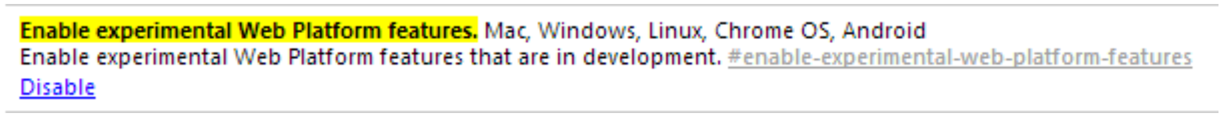
- Issue 245025: Implement WebCrypto in blink
  https://code.google.com/p/chromium/issues/detail?id=245025
- Issue 243345: [OWP Launch] DOM Futures
  https://code.google.com/p/chromium/issues/detail?id=243345
- WebCrypto implementation in Chromium
  https://docs.google.com/document/d/184AgXzLAoUjQjrtNdbimceyXVYzrn3tGpf3xQGCN10g

chromium


Blink
content/renderer
Normalize incoming AlgorithmIdentifier dictionary to C++ structures (copy data).
WebKit API
Crypto thread
WebCryptoAlgorithm
encrypt()
decrypt()
sign()
WebCryptoAlgorithm
Forward to worker thread
WebCryptoAlgorithm
Fan-out for doing the desired operation based on the WebCryptoAlgorithm.

Chromium Crypto Status

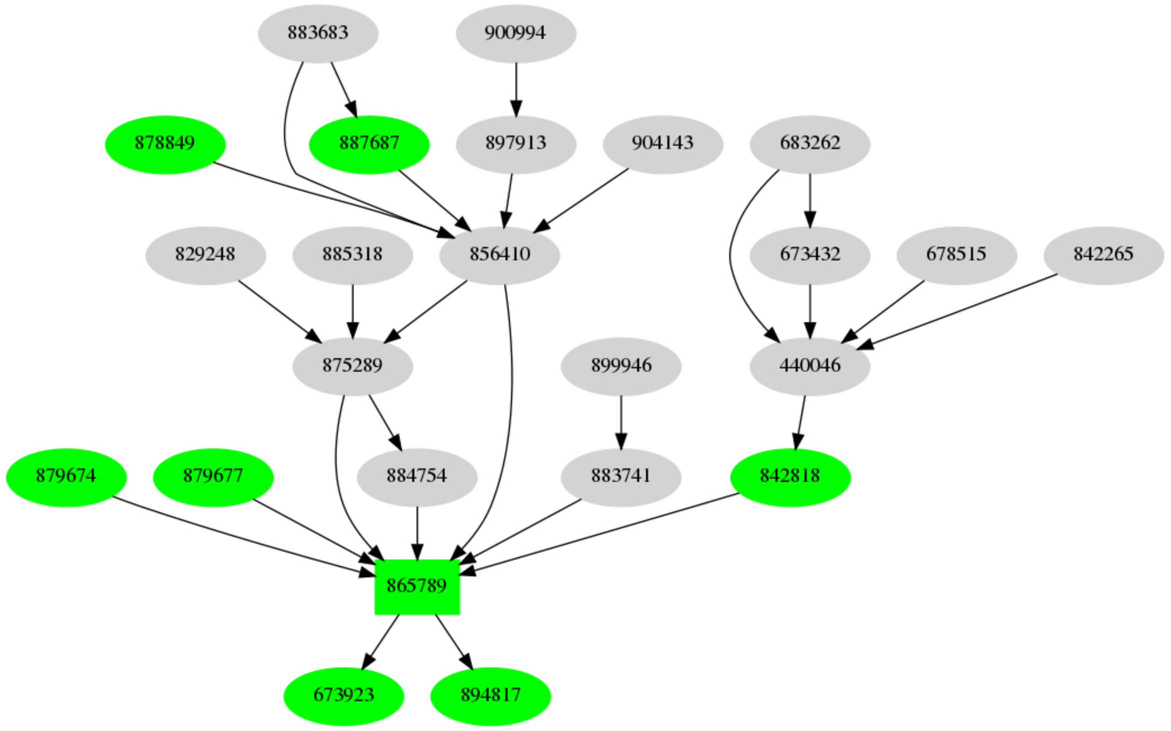
- Runtime Enabled Features
  http://www.chromium.org/blink/runtime-enabled-features
- As of *Tue Sep 24 05:59:19 2013 UTC*
  - Crypto status=test
    http://src.chromium.org/viewvc/blink/trunk/Source/core/page/RuntimeEnabledFeatures.in
- When Crypto status=experimental
  - chrome://flags/#enable-experimental-web-platform-features



Enable experimental Web Platform features. Mac, Windows, Linux, Chrome OS, Android
Enable experimental Web Platform features that are in development. #enable-experimental-web-platform-features
Disable

Contribute to Firefox?

- Bug 865789 - (web-crypto) Implement W3C Web Crypto API
  - https://bugzilla.mozilla.org/show_bug.cgi?id=865789



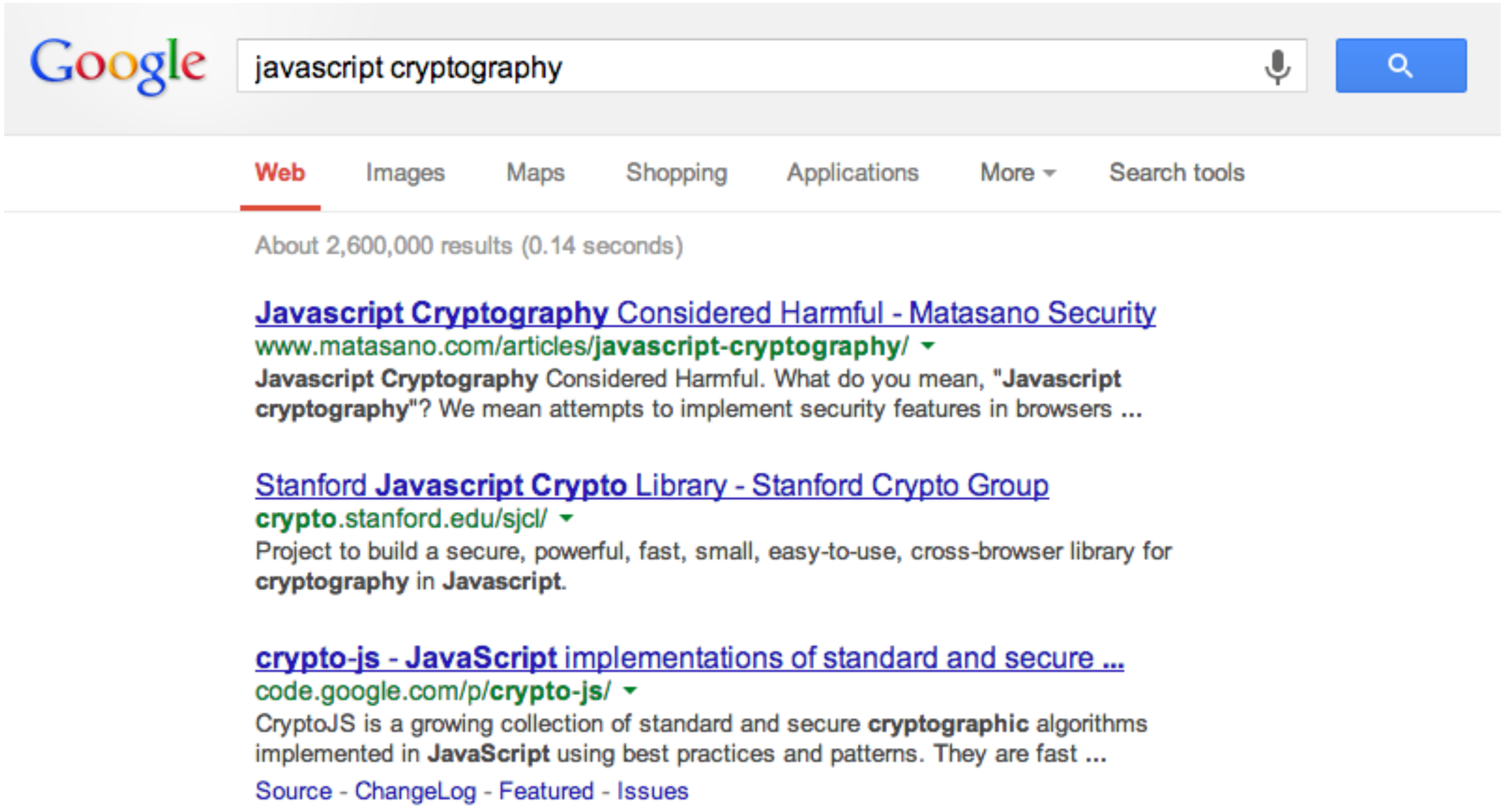
PolyCrypt: A WebCrypto Polyfill

"a pure JavaScript implementation of the WebCrypto API that people can use to get a feel for how they can use the API in practice"

Currently under active development by BBN Technologies, with funding from DHS S&T.

http://polycrypt.net/
https://github.com/polycrypt/polycrypt

Google: javascript cryptography



About 2,600,000 results (0.14 seconds)

Javascript Cryptography Considered Harmful - Matasano Security
www.matasano.com/articles/**javascript-cryptography**/
Javascript Cryptography Considered Harmful. What do you mean, "**Javascript cryptography**"? We mean attempts to implement security features in browsers ...

Stanford **Javascript Crypto** Library - Stanford Crypto Group
crypto.stanford.edu/sjcl/
Project to build a secure, powerful, fast, small, easy-to-use, cross-browser library for **cryptography** in **Javascript**.

crypto-js - **JavaScript** implementations of standard and secure ...
code.google.com/p/**crypto-js**/
CryptoJS is a growing collection of standard and secure **cryptographic** algorithms implemented in **JavaScript** using best practices and patterns. They are fast ...
Source - ChangeLog - Featured - Issues

Stanford Javascript Crypto Library

- SJCL is "a secure, powerful, fast, small, easy-to-use, cross-browser library for cryptography in Javascript"
- 2009 whitepaper focused on k-weight, performance and cryptographic randomness
    - "In Internet Explorer 8 our code is 11 times faster than the fastest current implementation."
    - "Our code is also 12% smaller than the smallest implementation currently available"
    - "Our PRNG itself is a modified version of the Fortuna PRNG."

http://crypto.stanford.edu/sjcl/

CryptoJS

CryptoJS is a growing collection of standard and secure cryptographic algorithms implemented in JavaScript using best practices and patterns.

- Hashers
- HMAC
- PBKDF2
- Ciphers
- Encoders

https://code.google.com/p/crypto-js/

Node.js Crypto

- Use `require('crypto')` to access this module.
- The crypto module requires OpenSSL to be available on the underlying platform.
- It also offers a set of wrappers for OpenSSL's hash, hmac, cipher, decipher, sign and verify methods.

http://nodejs.org/api/crypto.html

OpenSSL

The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library.

http://www.openssl.org/

<u>Javascript Cryptography Considered Harmful</u> (circa 2010)

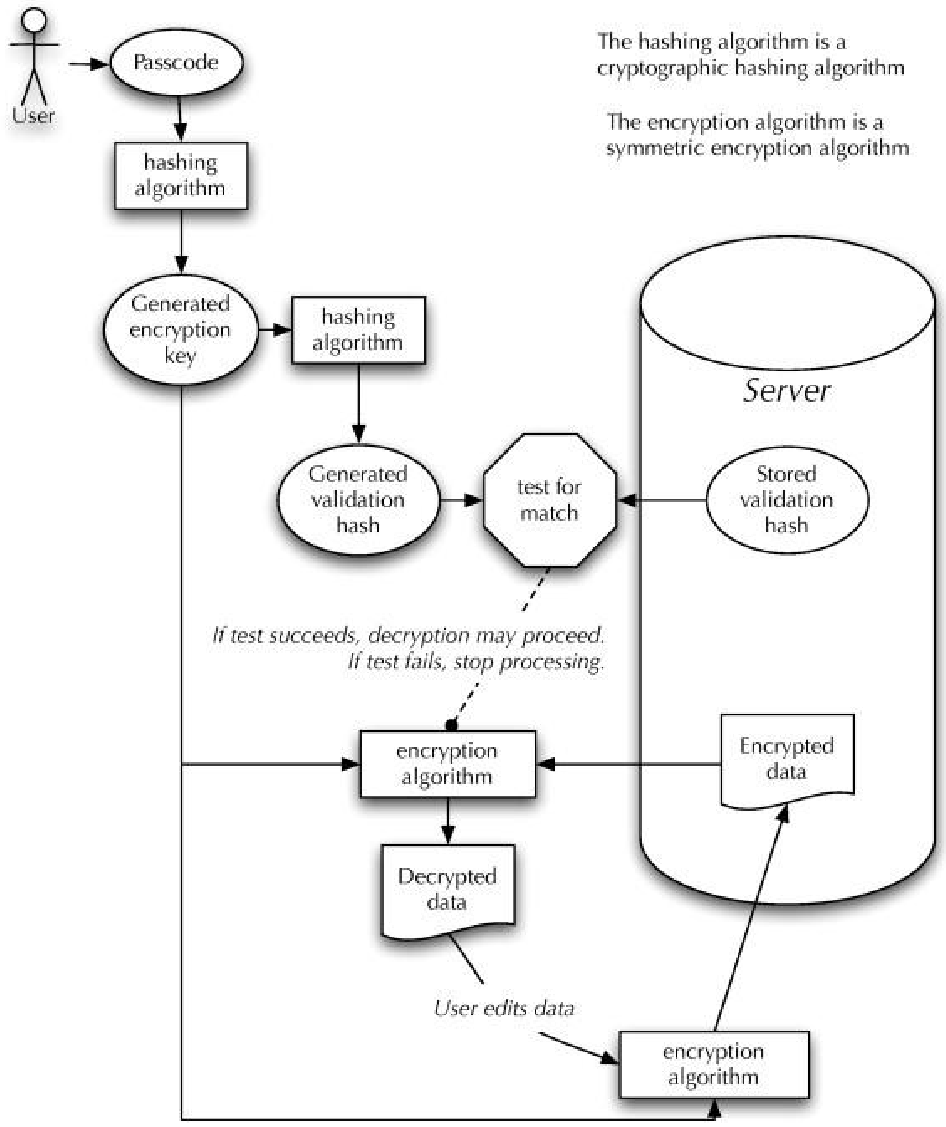
- Opinion on browser Javascript cryptography
    - "with SSL, you no longer need Javascript cryptography; you have 'real' cryptography"
    - "no reliable way for any piece of Javascript code to verify its execution environment"
    - "can't outsource random number generation in a cryptosystem"
    - "practically no value to doing crypto in Javascript once you add SSL to the mix"
    - "store the key on that server [and] documents there"

http://www.matasano.com/articles/javascript-cryptography/

Host-Proof Hosting

- In A Blink
  - Sketch: Locked inside data cloud, key at browser.
- Solution
  - Host sensitive data in encrypted form, so that clients can only access and manipulate it by providing a pass-phrase which is never transmitted to the server. The server is limited to persisting and retrieving whatever encrypted data the browser sends it, and never actually accesses the sensitive data in its plain form. It. All encryption and decryption takes place inside the browser itself.

http://ajaxpatterns.org/Host-Proof_Hosting (July 2005)


User
Passcode
hashing algorithm
Generated encryption key
hashing algorithm
Generated validation hash
test for match
Stored validation hash
Server
The hashing algorithm is a cryptographic hashing algorithm
The encryption algorithm is a symmetric encryption algorithm
If test succeeds, decryption may proceed.
If test fails, stop processing.
encryption algorithm
Encrypted data
Decrypted data
User edits data
encryption algorithm

Web-browser encryption of personal health information [http://www.biomedcentral.com/1472-6947/11/70]

Encryption data flow. A diagram laying out how the encrypted data and the user-supplied passcode are used to decrypt the data.

Morse *et al. BMC Medical Informatics and Decision Making* 2011 **11**:70 doi:10.1186/1472-6947-11-70

Host-Proof Hosting "Requirements"

- Secure transport mechanism (HTTPS).
- Trust provider that hosts web application and serves HTML and JavaScript resources.
- Defend against and accept risk of script injection (XSS) threat.
  - However, unauthorized access by hackers only attacks users who access the application while infected, and not the entire persisted data store.
- Avoid proving "Schneier's Law"
  - Anyone can invent a security system that he himself cannot break.
    http://www.schneier.com/blog/archives/2011/04/schneiers_law.html

OWASP Top 10 2013

OWASP
The Open Web Application Security Project

- A1-Injection
- A2-Broken Authentication and Session Management
- A3-Cross-Site Scripting (XSS)
- A4-Insecure Direct Object References
- A5-Security Misconfiguration
- **A6-Sensitive Data Exposure**
- A7-Missing Function Level Action Control
- A8-Cross-Site Request Forgery (CSRF)
- A9-Using Components with Known Vulnerabilities
- A10-Unvalidated Redirects and Forwards

https://www.owasp.org/index.php/Top_10_2013

OWASP Top 10 Change From 2010

2013-A6: Sensitive Data Exposure

- New category created by merging **2010-A7 – Insecure Cryptographic Storage** & **2010-A9 - Insufficient Transport Layer Protection**, plus adding browser side sensitive data risks as well.
- Covers sensitive data protection from the moment sensitive data is provided by the user, sent to and stored within the application, and then sent back to the browser again.

https://www.owasp.org/index.php/Top_10_2013-Release_Notes

A6-Sensitive Data Exposure

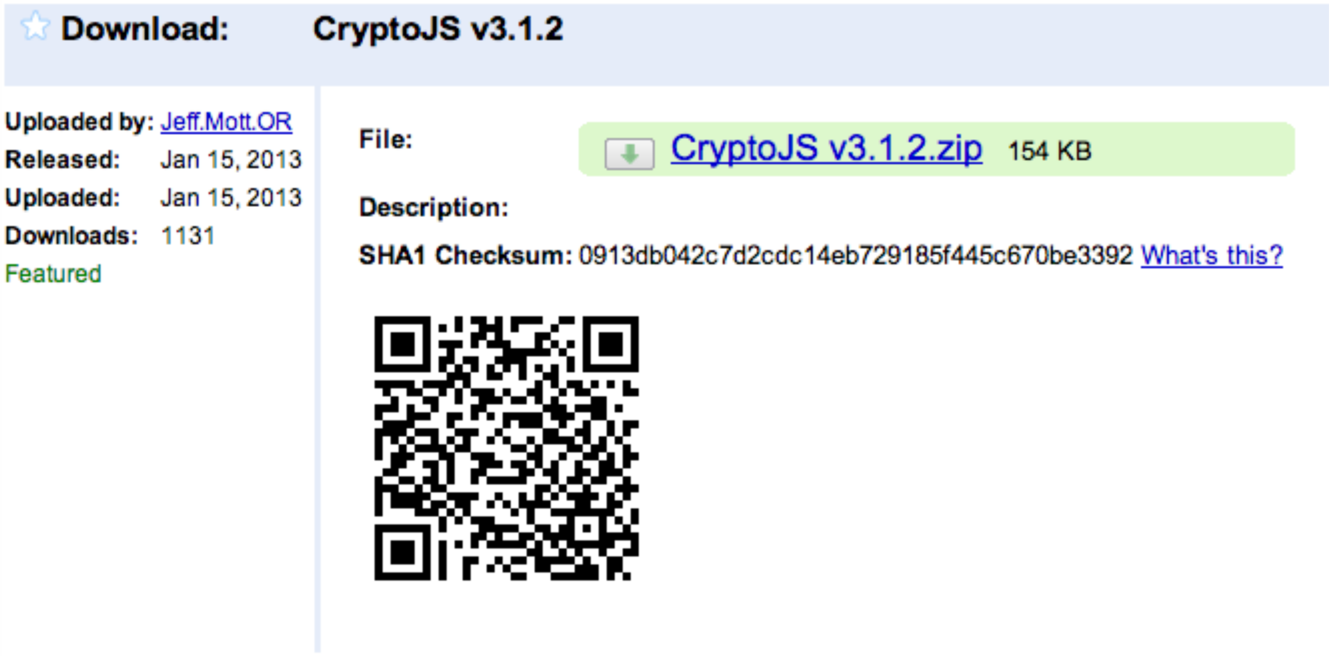
Am I Vulnerable To 'Sensitive Data Exposure'?

1. Is any of this data stored in clear text long term, including backups of this data?
2. Is any of this data transmitted in clear text, internally or externally? Internet traffic is especially dangerous.
3. Are any old / weak cryptographic algorithms used?
4. Are weak crypto keys generated, or is proper key management or rotation missing?
5. Are any browser security directives or headers missing when sensitive data is provided by / sent to the browser?

https://www.owasp.org/index.php/Top_10_2013-A6

Download CryptoJS

```
$ openssl dgst -sha1 "CryptoJS v3.1.2.zip"
SHA1(CryptoJS v3.1.2.zip)=
0913db042c7d2cdc14eb729185f445c670be3392
```



Download: **CryptoJS v3.1.2**

Uploaded by: Jeff.Mott.OR
Released: Jan 15, 2013
Uploaded: Jan 15, 2013
Downloads: 1131
Featured

File: CryptoJS v3.1.2.zip 154 KB

Description:

SHA1 Checksum: 0913db042c7d2cdc14eb729185f445c670be3392 What's this?

Cryptographic Hash Function

A function that maps a bit string of arbitrary length to a fixed length bit string. Approved hash functions satisfy the following properties:

1) (One-way) It is computationally infeasible to find any input which maps to any pre-specified output, and

2) (Collision resistant) It is computationally infeasible to find any two distinct inputs that map to the same output.

SOURCE: SP 800-21

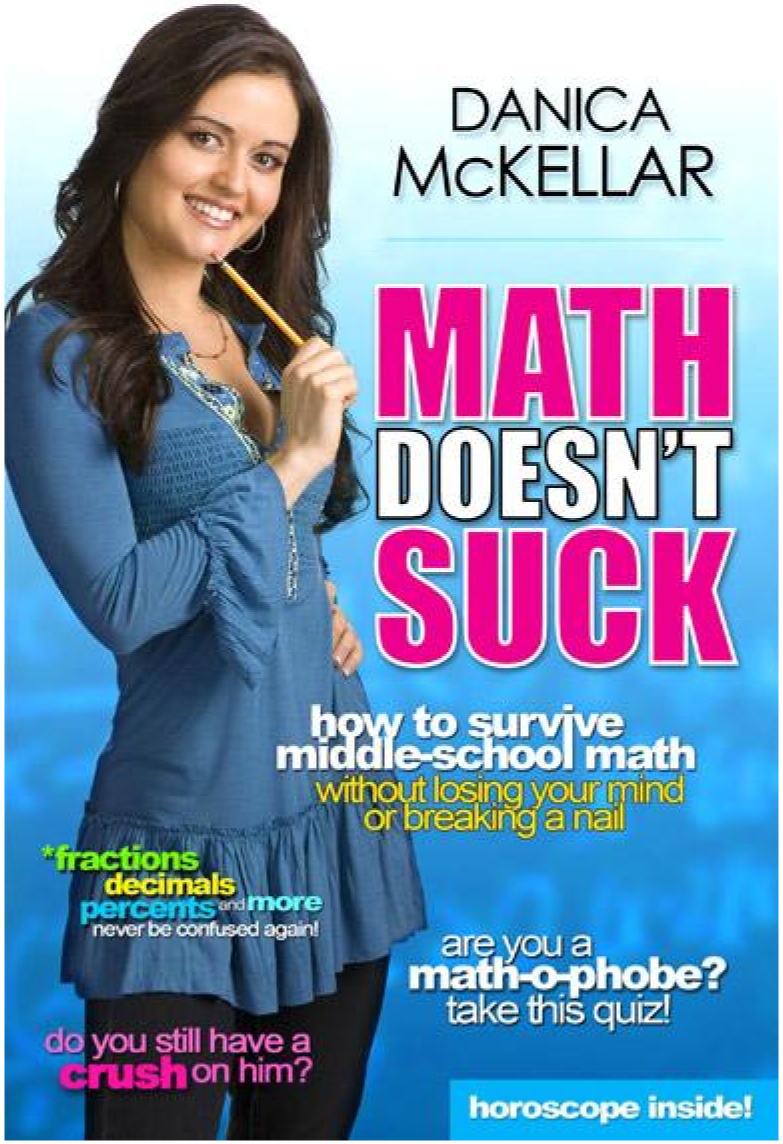
SHA-1 Definition

SHA-1 uses a sequence of logical functions, f_0, f_1,…, f_{79}. Each function f_t, where 0 ≤ t < 79, operates on three 32-bit words, x, y, and z, and produces a 32-bit word as output.

The function *f(x, y, z)* is defined as follows:

$$f_t(x, y, z) = \begin{cases} Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) & 0 \le t \le 19 \\ Parity(x, y, z) = x \oplus y \oplus z & 20 \le t \le 39 \\ Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \le t \le 59 \\ Parity(x, y, z) = x \oplus y \oplus z & 60 \le t \le 79. \end{cases}$$

SOURCE: FIPS 180-4


DANICA McKELLAR
MATH DOESN'T SUCK
how to survive middle-school math
without losing your mind or breaking a nail
*fractions decimals percents and more
never be confused again!
are you a math-o-phobe? take this quiz!
do you still have a crush on him?
horoscope inside!

OpenSSL

```
$ echo -n "The quick brown fox jumps over the lazy dog" |
openssl dgst -sha1
(stdin)= 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
```

CryptoJS

```
> CryptoJS
    .SHA1("The quick brown fox jumps over the lazy dog")
    .toString();
  "2fd4e1c67a2d28fced849ee1bb76e7391b93eb12"
```

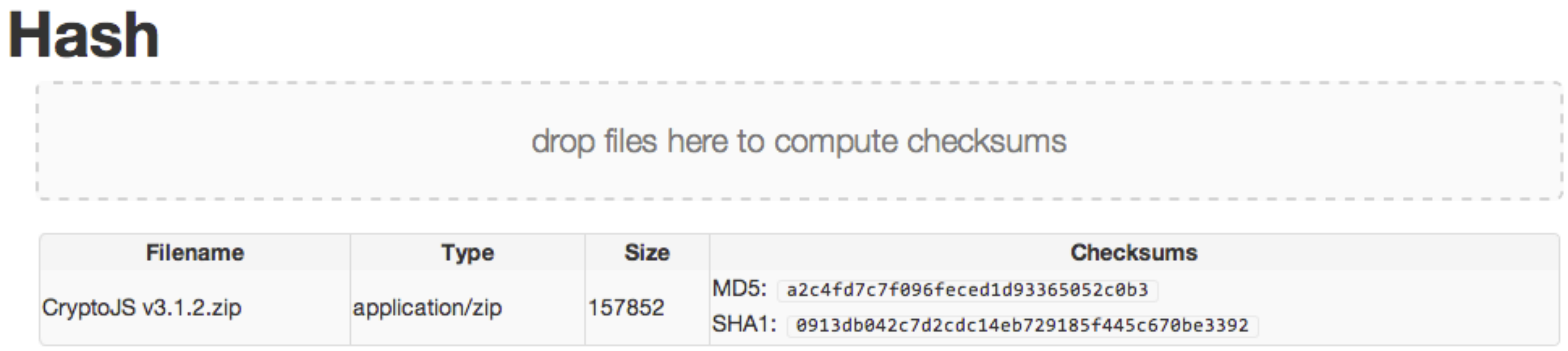
Node.js

```
> require("crypto")
    .createHash("sha1")
    .update("The quick brown fox jumps over the lazy dog")
    .digest("hex");
'2fd4e1c67a2d28fced849ee1bb76e7391b93eb12'
```

HTML5 Drag & Drop SHA1

```
var dropZone = document.getElementById('drop_zone');
dropZone.addEventListener('drop', handleFileSelect, false);

function handleFileSelect(evt) {
    evt.stopPropagation();
    evt.preventDefault();
    var files = evt.dataTransfer.files;
    for (var i = 0, f; f = files[i]; i++) {
        var reader = new FileReader();
        reader.readAsArrayBuffer(f);
        reader.onload = (function(theFile) {
            return function(e) {
                var wordArray = CryptoJS.lib.WordArray.create(e.target.result);
                var hash = CryptoJS.SHA1(wordArray);
            };
        })(f);
    }
}
```

Demo



Hash

drop files here to compute checksums

Filename	Type	Size	Checksums
CryptoJS v3.1.2.zip	application/zip	157852	MD5: `a2c4fd7c7f096feced1d93365052c0b3` SHA1: `0913db042c7d2cdc14eb729185f445c670be3392`

Hash-based Message Authentication Code (HMAC)

A message authentication code that uses a cryptographic key in conjunction with a hash function.

SOURCE: FIPS 201; CNSSI-4009

Symmetric Key

A cryptographic key that is used to perform both the cryptographic operation and its inverse, for example to encrypt and decrypt, or create a message authentication code and to verify the code.

SOURCE: SP 800-63; CNSSI-4009

Digital Signature

The result of a cryptographic transformation of data which, when properly implemented, provides the services of:

1. origin authentication,
2. data integrity, and
3. signer non-repudiation.

SOURCE: FIPS 140-2

require('cookie-signature')

- Node module used by `connect`, which is used by `express`
- Signs the value of a cookie

```
exports.sign = function(val, secret){
  if ('string' != typeof val) throw new TypeError('cookie required');
  if ('string' != typeof secret) throw new TypeError('secret required');
  return val + '.' + crypto
    .createHmac('sha256', secret)
    .update(val)
    .digest('base64')
    .replace(/\=+$/, '');
};
```

https://github.com/visionmedia/node-cookie-signature

This express code

```
app.use( express.cookieParser( "key" ) );
res.cookie( "text", "The quick brown fox jumps over the lazy dog",
  { signed : true } );
```

creates this signed cookie

```
Set-Cookie: text=s%3AThe%20quick%20brown%20fox%20jumps%20over%20the%20lazy%
20dog.97yD9DBThCSxMpjmqm%2BxQ%2B9NWaFJRhdZl0edvC0aPNg; Path=/
```

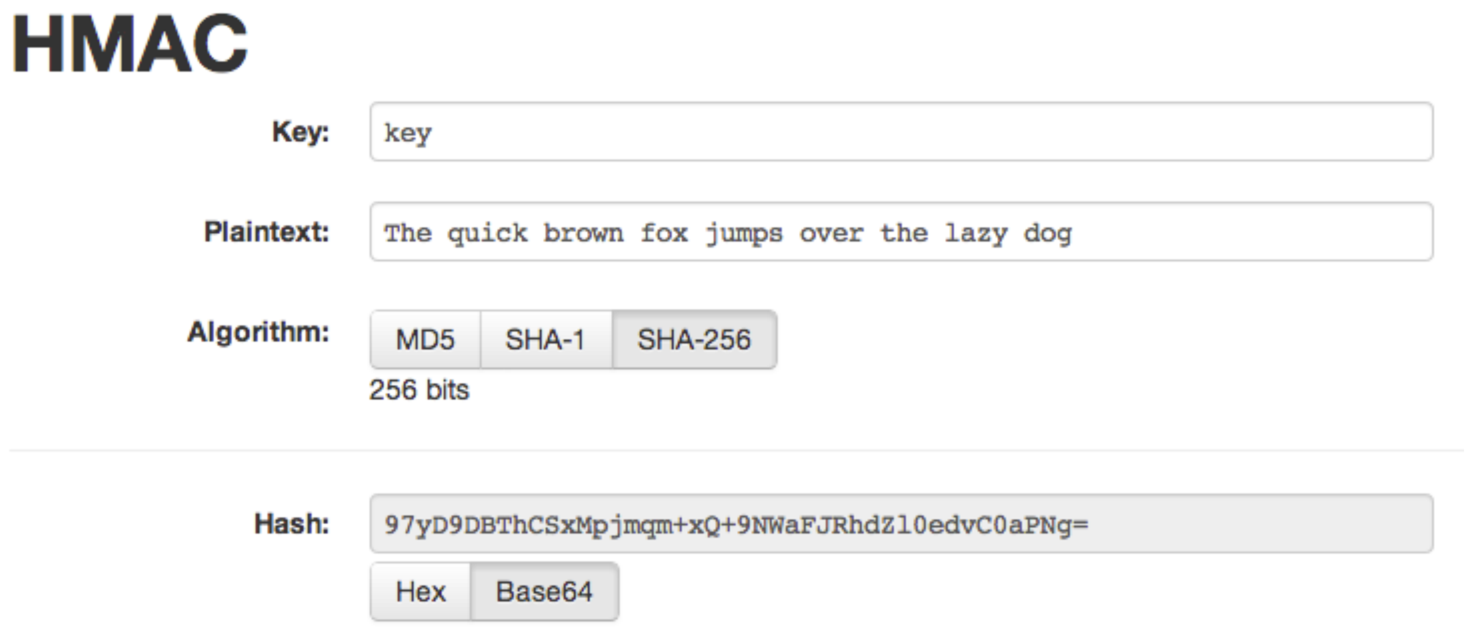
which matches our CryptoJS test

```
var hash = CryptoJS.HmacSHA256( "The quick brown fox jumps over the lazy dog",
"key" );
var base64 = CryptoJS.enc.Base64.stringify( hash );
equal( base64, "97yD9DBThCSxMpjmqm+xQ+9NWaFJRhdZl0edvC0aPNg=", "matches value"
);
```

and OpenSSL command line.

```
$ echo -n "The quick brown fox jumps over the lazy dog" | openssl dgst -sha256 -
hmac "key" -binary | openssl base64
97yD9DBThCSxMpjmqm+xQ+9NWaFJRhdZl0edvC0aPNg=
```

Demo



HMAC

Key: key

Plaintext: The quick brown fox jumps over the lazy dog

Algorithm: MD5 | SHA-1 | [✓ SHA-256]

256 bits

Hash: 97yD9DBThCSxMpjmqm+xQ+9NWaFJRhdZl0edvC0aPNg=

Hex | [✓ Base64]

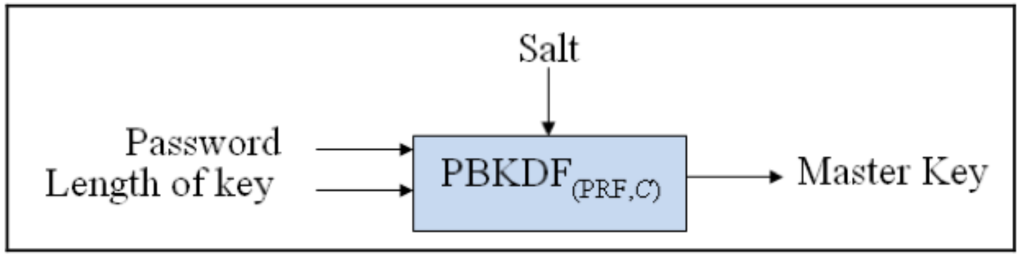
ONE DOES NOT SIMPLY
ENCRYPT WITH A PASSWORD
quickmeme.com

Password-Based Key Derivation Functions (PBKDF)

- The randomness of cryptographic keys is essential for the security of cryptographic applications.
- Most user-chosen passwords have low entropy and weak randomness properties.
  - **shall not** be used directly as cryptographic keys
- KDFs are deterministic algorithms that are used to derive cryptographic keying material from a secret value, such as a password.

SOURCE: SP 800-132

PBKDF Specification



Input:

P	Password
S	Salt
C	Iteration count
kLen	Length of MK in bits; at most $(2^{32}-1)$ x hLen

Parameter:

PRF	HMAC with an approved hash function
hlen	Digest size of the hash function

Output:

mk	Master key

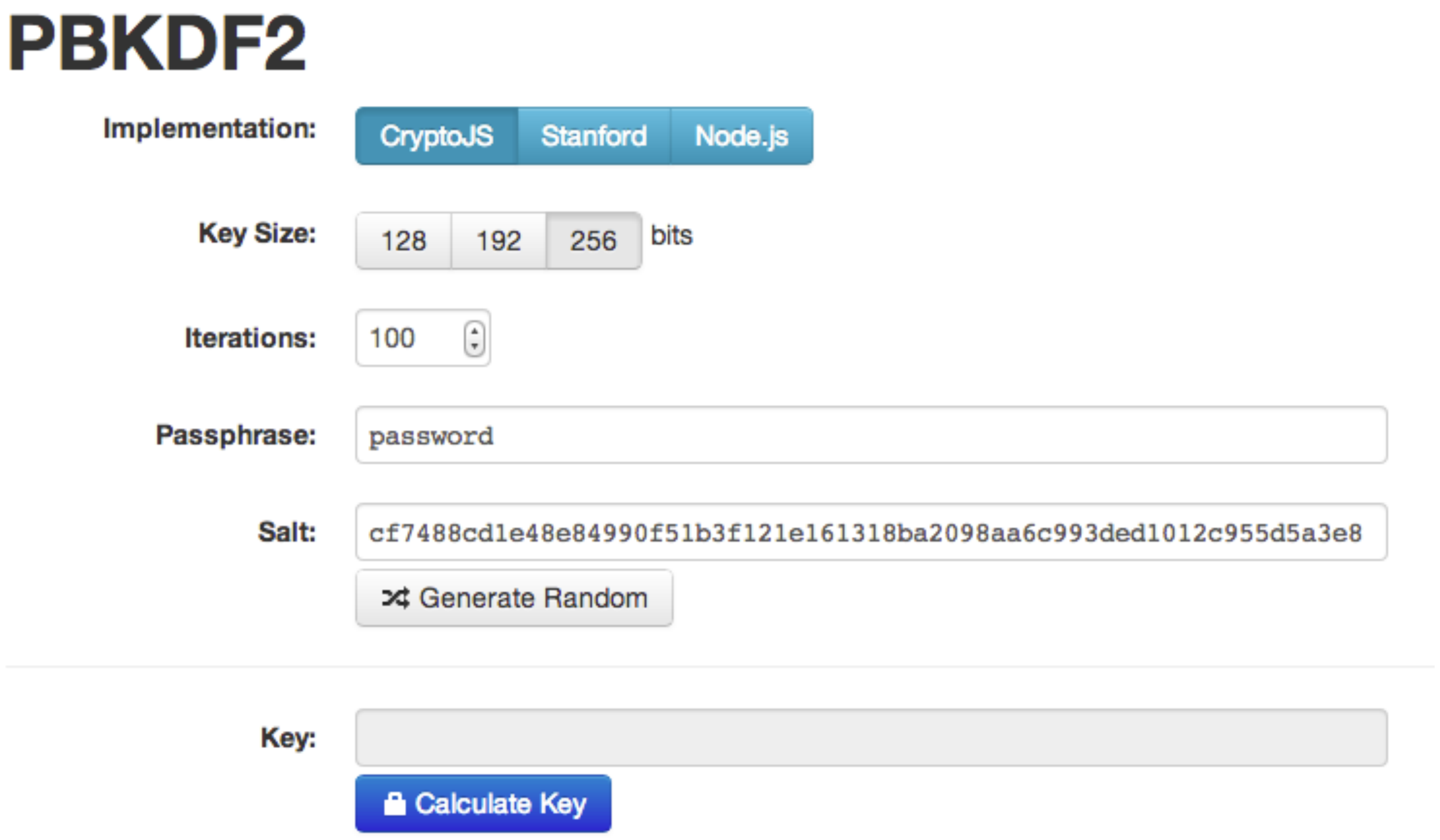
http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf

Salt

A non-secret value that is used in a cryptographic process, usually to ensure that the results of computations for one instance cannot be reused by an Attacker.

SOURCE: SP 800-63; CNSSI-4009

Demo



PBKDF2

Implementation: [✓ CryptoJS] Stanford Node.js

Key Size: 128 192 [✓ 256] bits

Iterations: 100

Passphrase: password

Salt: cf7488cd1e48e84990f51b3f121e161318ba2098aa6c993ded1012c955d5a3e8

Generate Random

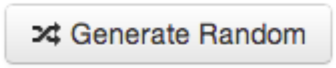
Key:

Calculate Key

Pseudorandom number generator (PRNG)

An algorithm that produces a sequence of bits that are uniquely determined from an initial value called a seed. The output of the PRNG “appears” to be random, i.e., the output is statistically indistinguishable from random values. A cryptographic PRNG has the additional property that the output is unpredictable, given that the seed is not known.

SOURCE: CNSSI-4009



Random Data Functions

- JavaScript
  - ~~Math.random()~~
- Node.js
  - crypto.randomBytes(n)
- WebCryptoAPI
  - window.crypto.getRandomValues(array)
- CryptoJS
  - CryptoJS.lib.WordArray.random(n)
- SJCL
  - sjcl.random.randomWords(n, paranoia)

Binary Data Structures

- JavaScript
  - "binary" String
- Node.js
  - Buffer
- "HTML5"
  - TypedArray
- CryptoJS
  - CryptoJS.lib.WordArray
- SJCL
  - sjcl.bitArray

Node.js crypto.randomBytes()

- Generates cryptographically strong pseudo-random data.

```
var buf = crypto.randomBytes(256);
```

- The Crypto module was added to Node … before there were **`Buffer`** objects for handling binary data.
- As such, ... many methods accept and return Binary-encoded strings by default rather than **`Buffers`**.

  http://nodejs.org/api/crypto.html

Node.js Buffer

- Pure JavaScript is Unicode friendly but not nice to binary data.
- Raw data is stored in instances of the **`Buffer`** class. A **`Buffer`** is similar to an array of integers but corresponds to a raw memory allocation outside the V8 heap.
- Converting between **`Buffer`**s and JavaScript string objects requires an explicit encoding method.
  - **`'binary'`** - A way of encoding raw binary data into strings by using only the first 8 bits of each character.
  - **`'hex'`** - Encode each byte as two hexadecimal characters.

http://nodejs.org/api/buffer.html#buffer_buffer

window.crypto.getRandomValues()

If you provide an integer-based `TypedArray`, the function is going fill the array with cryptographically random numbers.

```
var buf = new Uint8Array(32);
window.crypto.getRandomValues(buf);
```

https://developer.mozilla.org/en-US/docs/DOM/window.crypto.getRandomValues

http://msdn.microsoft.com/en-us/library/ie/dn302324(v=vs.85).aspx

TypedArray

ECMAScript [ECMA-262] has traditionally been used in contexts where there is no access to binary data. Where binary data has needed to be manipulated, it is often stored as a String and accessed using charCodeAt(), or stored as an Array with conversion to and from base64 for transmission.

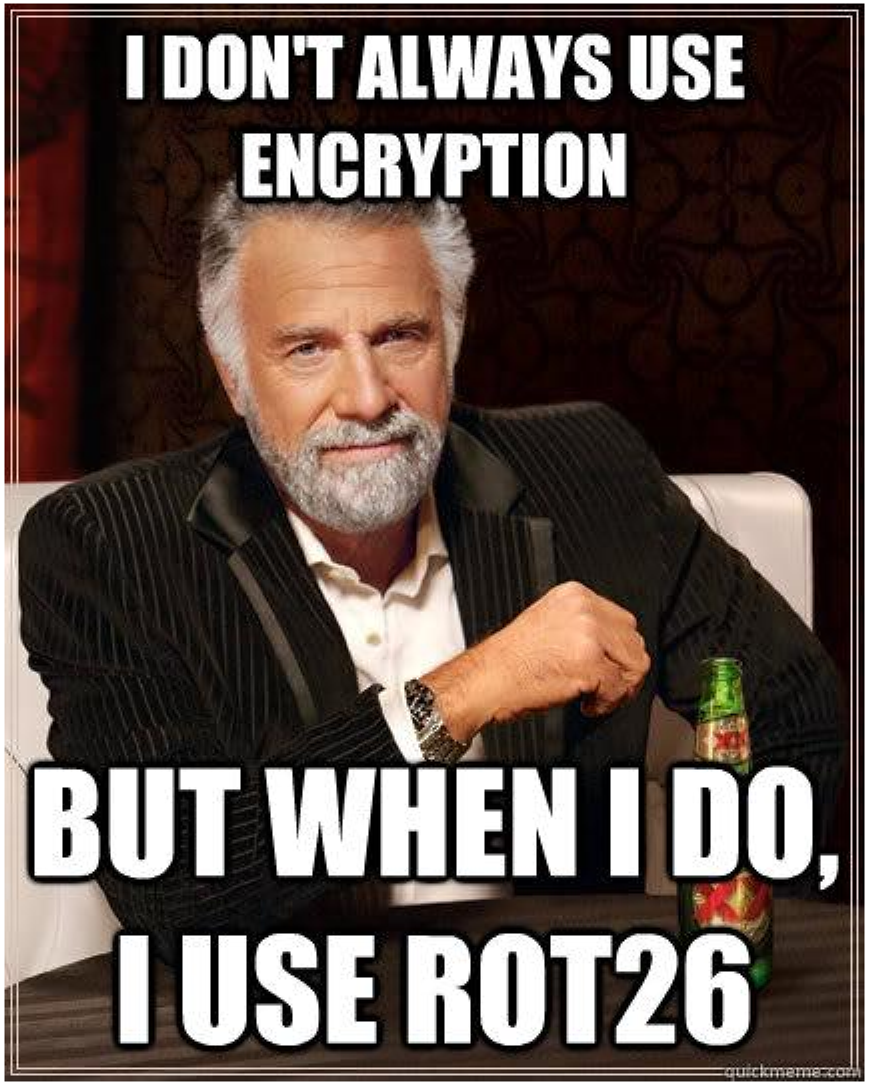
https://www.khronos.org/registry/typedarray/specs/latest/

CryptoJS.lib.WordArray

A WordArray object represents an array of 32-bit words.

```
var hash = CryptoJS.SHA256("Message");

alert(hash.toString(CryptoJS.enc.Base64));
// L3dmip37+NWEi57rSnFFypTG7ZI25Kdz9tyvpRMrL5E=

alert(hash.toString(CryptoJS.enc.Latin1));
// /wf��ûøÕ���ëJqEÊ�Æí�6ä§söÜ¯¥+/�

alert(hash.toString(CryptoJS.enc.Hex));
// 2f77668a9dfbf8d5848b9eeb4a7145ca94c6ed9236e4a773f6dcafa5132b2f91
```

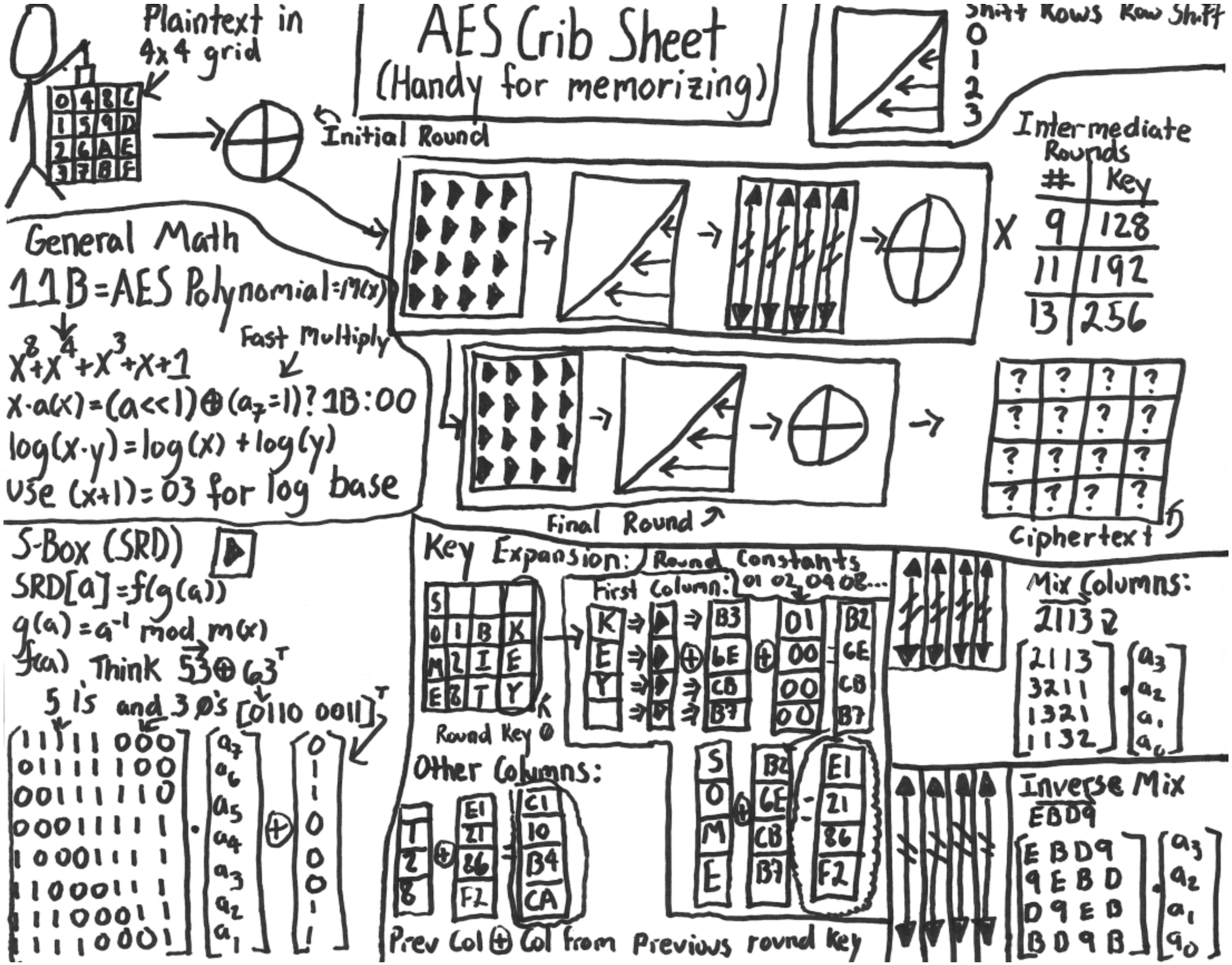
https://code.google.com/p/crypto-js/#The_Hasher_Output


I DON'T ALWAYS USE ENCRYPTION
BUT WHEN I DO, I USE ROT26
quickmeme.com

AES

The Advanced Encryption Standard specifies a U.S. government approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information. This standard specifies the Rijndael algorithm, a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits.

SOURCE: FIPS 197


AES Crib Sheet
(Handy for memorizing)
Plaintext in 4x4 grid
Initial Round
Shift Rows Row Shift 0 1 2 3
Intermediate Rounds
# Key
9 128
11 192
13 256
General Math
11B = AES Polynomial = m(x)
x^8+x^4+x^3+x+1
Fast Multiply
x·a(x) = (a<<1) ⊕ (a7=1) ? 1B:00
log(x·y) = log(x) + log(y)
Use (x+1) = 03 for log base
Final Round
Ciphertext
S-Box (SRD)
SRD[a] = f(g(a))
g(a) = a^-1 mod m(x)
f(a) Think 53 ⊕ 63^T
5 1's and 3 0's [0110 0011]^T
Key Expansion:
Round Constants
First Column:
01 02 04 08...
Round Key 0
Other Columns:
Prev Col ⊕ Col from Previous round key
Mix Columns:
2113
Inverse Mix
EBD9

Initialization Vector (IV)

A vector used in defining the starting point of an encryption process within a cryptographic algorithm.

SOURCE: FIPS 140-2

OpenSSL Encryption Format

Since OpenSSL 0.9.5 (early 2000), `openssl enc` produces output in the following format :

- "Salted__" magic string
- 8 bytes of salt
- encrypted data

To decrypt a file encrypted with 0.9.4 (or earlier) or other crypto software, use the `-nosalt` command line option.

http://www.mail-archive.com/openssl-users@openssl.org/msg35646.html

```
$ echo -n "Salted__" | xxd -p
53616c7465645f5f

$ echo -n "Message" | openssl enc -
aes-256-cbc -k "password" -S
4521F86027413403 | xxd -p -cols 32
53616c7465645f5f4521f8602741340323e5eb
e72d99cf302c99183c05cf050a

$ echo -n "Message" | openssl enc -
aes-256-cbc -k "password" -nosalt |
xxd -p
0293cf0bdf5323cff809ba406ffc8283
```

OpenSSL EVP_BytesToKey()

Derives a key and IV from a password and salt.
Not compatible with PBKDF2.

```
$ openssl enc -aes-256-cbc -k
"password" -S 4521F86027413403 -P

salt=4521F86027413403
key=0CD1D07EB67E19EF56EA0F3A9A8F8A7C95
7A2CB208327E0E536608FF83256C96
iv =6C4C31BDAB7BAFD35B23691EC521E28D
```

OpenSSL Encryption

OpenSSL command line encryption with key and IV instead of password and salt.

```
$ echo -n "Message" | openssl enc -aes-256-cbc -
iv 6C4C31BDAB7BAFD35B23691EC521E28D -K
0CD1D07EB67E19EF56EA0F3A9A8F8A7C957A2CB208327E0E
536608FF83256C96 | xxd -p
23e5ebe72d99cf302c99183c05cf050a
```

CryptoJS Key and IV

```
var testVector = { plaintext : "Message",
  iv : "6C4C31BDAB7BAFD35B23691EC521E28D",
  key : "0CD1D07EB67E19EF56EA0F3A9A8F8A7C957A2CB208327E0E536608FF83256C96",
  ciphertext :
"53616c7465645f5f4521f8602741340323e5ebe72d99cf302c99183c05cf050a"};

var rawEnc = CryptoJS.AES.encrypt(testVector.plaintext,
  CryptoJS.enc.Hex.parse(testVector.key),
  { iv : CryptoJS.enc.Hex.parse(testVector.iv), mode: CryptoJS.mode.CBC});

equal(CryptoJS.enc.Hex.stringify(rawEnc.ciphertext),
  testVector.ciphertext.substring(32), "decrypt matches ciphertext");
```

CryptoJS Password and Salt

```
var testVector = { plaintext : "Message",
  iv : "6C4C31BDAB7BAFD35B23691EC521E28D",
  key : "0CD1D07EB67E19EF56EA0F3A9A8F8A7C957A2CB208327E0E536608FF83256C96",
  ciphertext : "53616c7465645f5f4521f86027413403 23e5ebe72d99cf302c99183c05cf050a"};

// https://code.google.com/p/crypto-js/issues/detail?id=85
var compatEnc = CryptoJS.AES.encrypt(testVector.plaintext, "password",
  { salt: CryptoJS.enc.Hex.parse("4521F86027413403") });

var iv = CryptoJS.enc.Hex.stringify(compatEnc.iv).toUpperCase();
equal(iv, testVector.iv, "matches iv");

var key = CryptoJS.enc.Hex.stringify(compatEnc.key).toUpperCase();
equal(key, testVector.key, "matches key");

var ciphertext = CryptoJS.enc.Hex.stringify(compatEnc.ciphertext);
equal(ciphertext, testVector.ciphertext.substring(32), "matches ciphertext");
```

CryptoJS Password and NoSalt

```
var nosalt = CryptoJS.lib.WordArray.random(0);

// { salt : null } will generate random salt
var enc = CryptoJS.AES.encrypt("Message", "password",
  { salt: nosalt });

var ciphertext = CryptoJS.enc.Hex.stringify(enc.ciphertext);

equal(ciphertext, "0293cf0bdf5323cff809ba406ffc8283",
  "decrypt matches nosalt ciphertext");
```

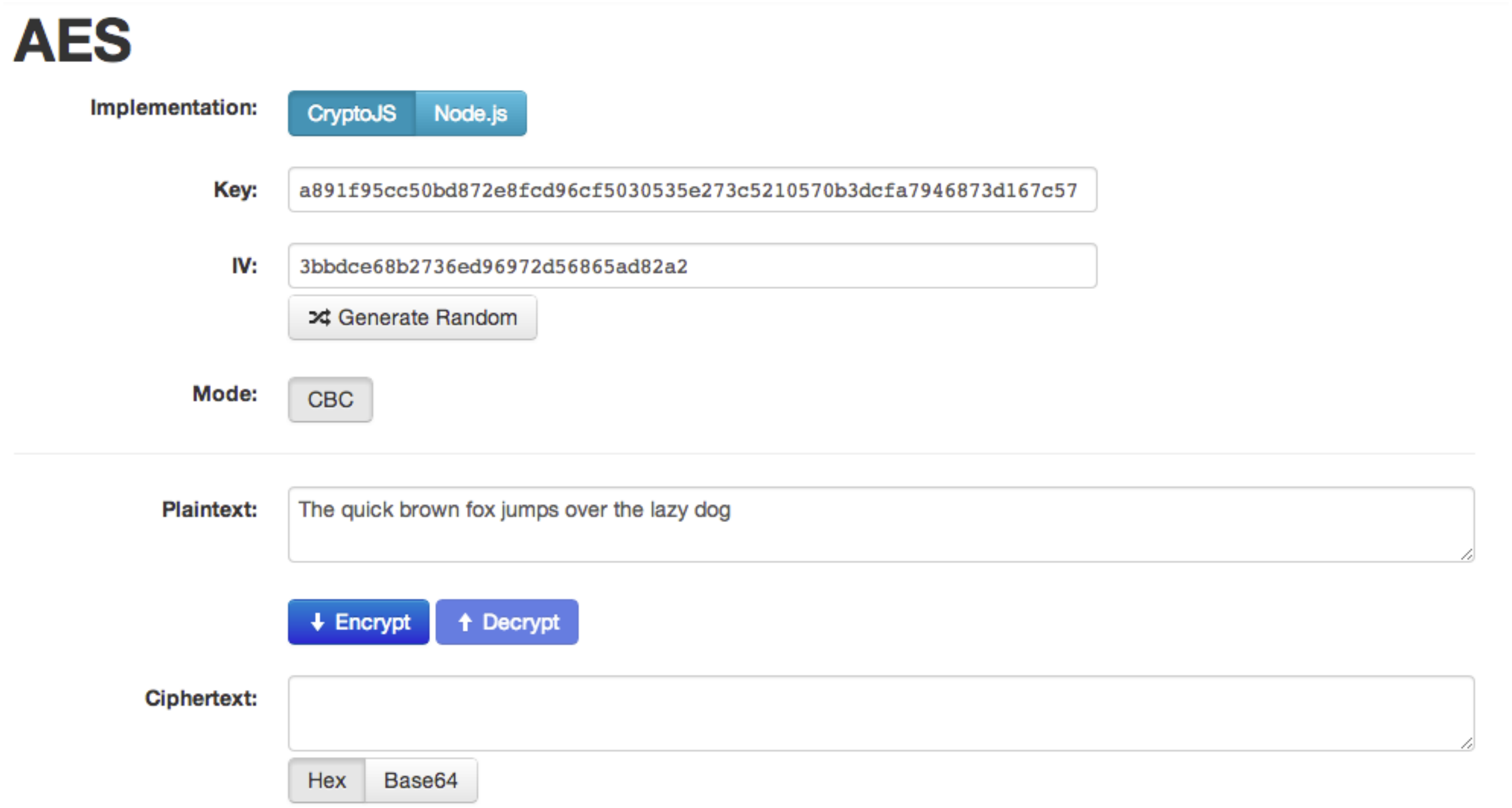
Node.js Crypto Key and IV

```
var crypto = require("crypto");

var testVector = { plaintext : "Message",
  iv : "6C4C31BDAB7BAFD35B23691EC521E28D",
  key : "0CD1D07EB67E19EF56EA0F3A9A8F8A7C957A2CB208327E0E536608FF83256C96",
  ciphertext :
"53616c7465645f5f4521f8602741340323e5ebe72d99cf302c99183c05cf050a"};


var key = new Buffer(testVector.key, "hex");
var iv = new Buffer(testVector.iv, "hex");
var rawEnc = crypto.createCipheriv("aes-256-cbc", key, iv);
var rawCrypted = rawEnc.update(testVector.plaintext, "utf8", "hex");
rawCrypted += rawEnc.final("hex");
console.log(rawCrypted);
// 23e5ebe72d99cf302c99183c05cf050a
```

Node.js Crypto Password (NoSalt)

```
var crypto = require("crypto");

/* createCipher compatible with OpenSSL -nosalt */
var compatEnc = crypto.createCipher("aes-256-cbc", "password");
compatCrypted = compatEnc.update("Message", "utf8", "hex");
compatCrypted += compatEnc.final("hex");
console.log(compatCrypted);
// 0293cf0bdf5323cff809ba406ffc8283
```

Demo



AES

Implementation: CryptoJS Node.js

Key: a891f95cc50bd872e8fcd96cf5030535e273c5210570b3dcfa7946873d167c57

IV: 3bbdce68b2736ed96972d56865ad82a2

Mode: CBC

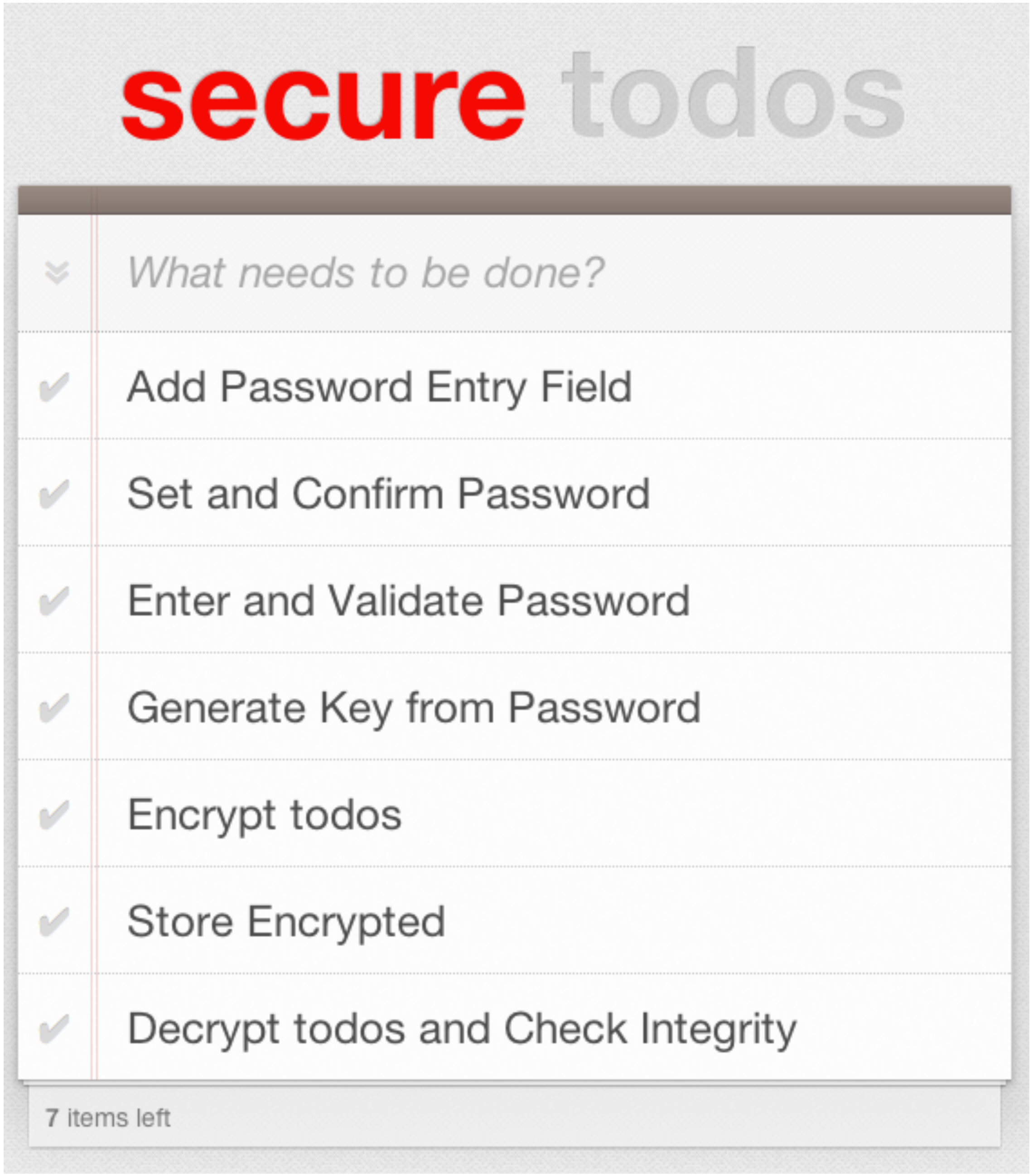
Plaintext: The quick brown fox jumps over the lazy dog

Ciphertext:

AngularJS (Novice)Tip

Match 128, 192 or 256 bit hex string

```
<div class="control-group"
     ng-class="{ error: aesForm.key.$invalid }">
    <label class="control-label" for="key">Key: </label>
    <div class="controls">
        <input type="text" id="key" name="key" size="32"
         ng-model="key" ng-pattern="hexPatternKey" required>
    </div>
</div>


$scope.hexPatternKey = /^[0-9A-Fa-f]{32}([0-9A-Fa-f]{16})?([0-
9A-Fa-f]{16})?$/;
$scope.hexPatternIV = /^[0-9A-Fa-f]{32}$/;
```


secure todos

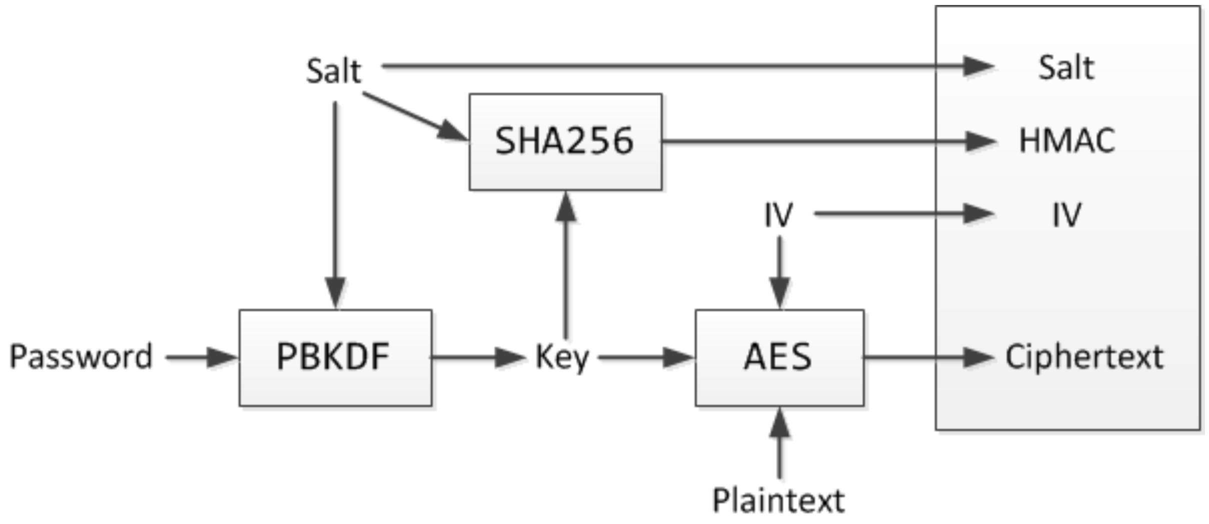
What needs to be done?

- ✓ Add Password Entry Field
- ✓ Set and Confirm Password
- ✓ Enter and Validate Password
- ✓ Generate Key from Password
- ✓ Encrypt todos
- ✓ Store Encrypted
- ✓ Decrypt todos and Check Integrity

7 items left

Data Flow and Storage

- Salt = initial random
- IV = initial random
- HMAC = SHA256(salt, key)
- Ciphertext = AES(iv | todos, key)

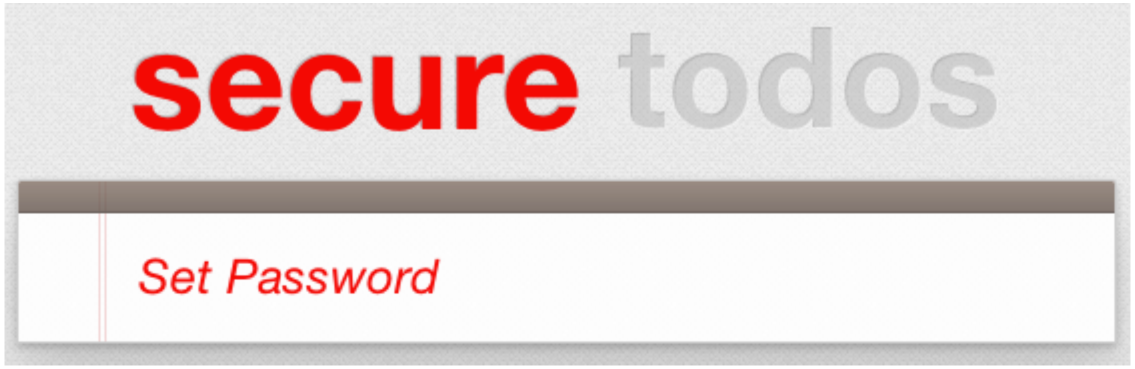


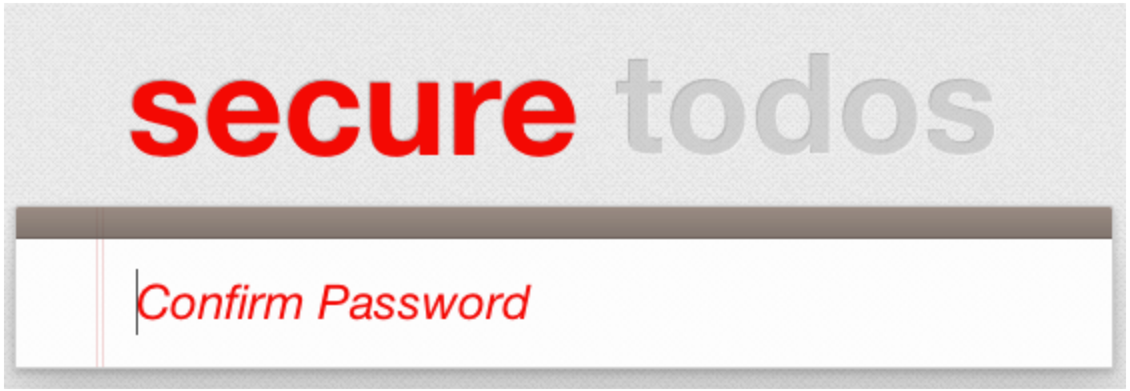
Add Password Entry Field

```
<header id="header">
  <h1><span>secure</span> todos</h1>
  <input id="new-todo"
         placeholder="What needs to be done?"
         style="display: none">
  <input id="todo-password"
         type="password"
         placeholder="Enter Password"
         autofocus
         class="edit">
</header>
```

Set and Confirm Password

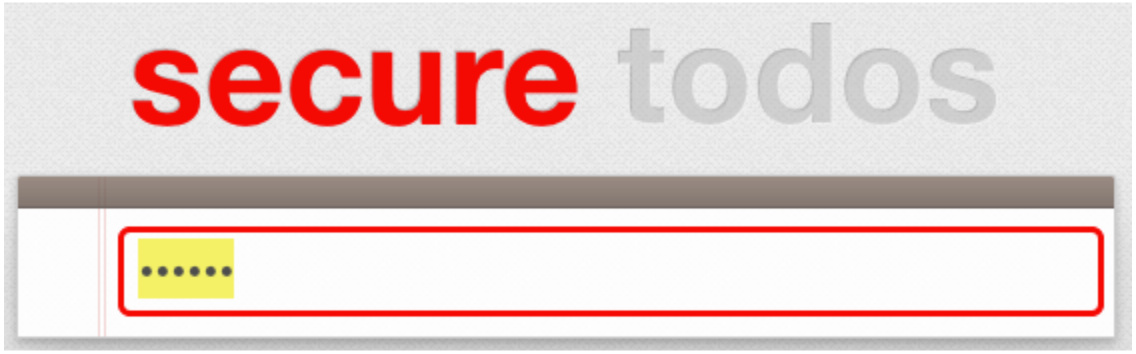
On first use, password must be established.





Enter and Validate Password

On subsequent uses, password must be entered to unlock todos.



An invalid password will shake the password input field and outline in red.

Password Validation

```
if (App.validatePassword(val, confirmVal)) {
    $input.val('');
} else {
    if (confirmVal) {
        $input.val('');
    } else {
        $input.select();
    }
    // shake animation
    $input.addClass("invalid");
}
App.render();
```

Generate Key from Password

```
var key = CryptoJS.PBKDF2(password, this.encryptedData.salt,
                          { keySize: 256 / 8, iterations: 100 });
if (confirmPassword) {
    todos = [];
    this.encryptedData.hmac =
              CryptoJS.HmacSHA256(this.encryptedData.salt, key);
} else {
    todos = App.decryptTodos(key);
}

if (todos) {
    App.key = key;
    App.todos = todos;
    App.initialized = true;
}
```

CSS3 Shake Animation

```
#todo-password.invalid {
    border: 3px solid;
    border-radius: 6px;
    border-color: red;
    outline-color: red;
    -webkit-animation: shake .5s linear ;
}

@-webkit-keyframes shake {
     8%, 41%  { -webkit-transform: translateX(-10px); }
    25%, 58%  { -webkit-transform: translateX(10px);  }
    75%       { -webkit-transform: translateX(-5px);  }
    92%       { -webkit-transform: translateX(5px);   }
     0%, 100% { -webkit-transform: translateX(0);     }
}
```

Demo

Windows Azure

Windows Azure ProTip

- Azure maps `app.js` to run on server using iisnode module in `wwwroot/web.config`
- TodoMVC uses `app.js` as JavaScript filename requested by browser
- Problem: these do not work together
- Solution: create `/public/web.config`

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <system.webServer>
        <handlers>
            <remove name="iisnode" />
        </handlers>
    </system.webServer>
</configuration>
```

```
render: function () {
  if (this.key) {
    this.$password.hide();
    this.$newTodo.show().focus();
    this.$todoList.html(this.todoTemplate(this.todos));
    this.$main.toggle(!!this.todos.length);
    this.$toggleAll.prop('checked', !this.activeTodoCount());
    this.renderFooter();
    Utils.store('todos-jquery-encrypted',
                  this.encryptTodos(this.key));
  } else {
    this.$newTodo.hide();
    if (!this.initialized) {
      var placeholder = (this.$password.data("confirm") ?
                         "Confirm Password" : "Set Password");
      this.$password.attr("placeholder", placeholder);
    }
    this.$password.show().focus();
  }
}
```

Encrypt todos

```
encryptTodos: function (key) {
    var hexSalt =
        CryptoJS.enc.Hex.stringify(this.encryptedData.salt);
    var plaintext = hexSalt + JSON.stringify(this.todos);
    var enc = CryptoJS.AES.encrypt(plaintext, key,
        { iv: this.encryptedData.iv, mode: CryptoJS.mode.CBC });

    var encryptedTodos = {
        salt : this.encryptedData.salt,
        hmac : this.encryptedData.hmac,
        iv : this.encryptedData.iv,
        ciphertext : enc.ciphertext
    };

    return encryptedTodos;
}
```

Decrypt todos and Check Integrity

```
var hmac = CryptoJS.HmacSHA256(this.encryptedData.salt, key);
if (hmac.toString() != this.encryptedData.hmac.toString() ) {
    return null;
}

var cipherParams = CryptoJS.lib.CipherParams.create({
    ciphertext: this.encryptedData.ciphertext});
var dec = CryptoJS.AES.decrypt(cipherParams, key,
    { iv: this.encryptedData.iv, mode: CryptoJS.mode.CBC });

var plaintext = CryptoJS.enc.Latin1.stringify(dec);
var hexSalt = CryptoJS.enc.Hex.stringify(this.encryptedData.
salt);
var pos = plaintext.indexOf(hexSalt, 0);
```

Store Encrypted (Write)

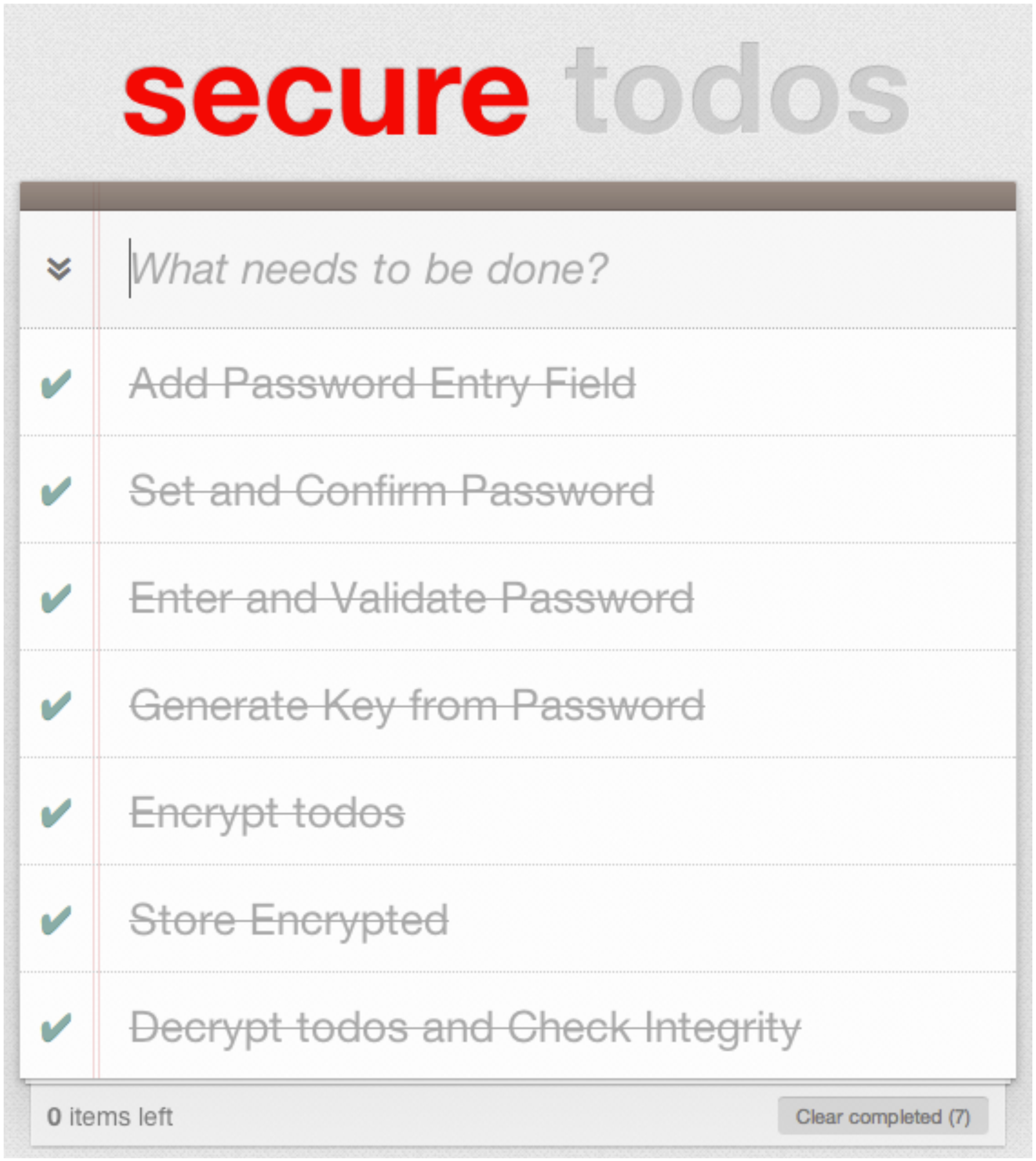
```
if (arguments.length > 1) {
    // write
    var writeStore = {
        salt : CryptoJS.enc.Hex.stringify(data.salt),
        hmac : CryptoJS.enc.Hex.stringify(data.hmac),
        iv :   CryptoJS.enc.Hex.stringify(data.iv),
        ciphertext :
               CryptoJS.enc.Base64.stringify(data.ciphertext)
    };
    return localStorage.setItem(namespace,
                                 JSON.stringify(writeStore));
} else {
    // read on next slide
}
```

Store Encrypted (Read)

```
var readStore = localStorage.getItem(namespace);
data = {};
if (readStore && JSON.parse(readStore)) {          // existing
    var tmpData = JSON.parse(readStore);
    data.salt = CryptoJS.enc.Hex.parse(tmpData.salt);
    data.hmac = CryptoJS.enc.Hex.parse(tmpData.hmac);
    data.iv =   CryptoJS.enc.Hex.parse(tmpData.iv);
    data.ciphertext = CryptoJS.enc.Base64.parse(tmpData.ciphertext);
} else {                                                 // new
    data.salt = CryptoJS.lib.WordArray.random(256 / 8);
    data.hmac = null;
    data.iv = CryptoJS.lib.WordArray.random(128 / 8);
    data.ciphertext = null;
}
return data;
```

Encrypted in `localStorage`

```
sqlite> select * from ItemTable;
todos-jquery-encrypted|{"salt":"
98e9dd2d7de990bce5b46d9194076e03cddfd53e0e3f3063
e8e9949646f27506","hmac":"
91c38bb27c4bd53029901d4bee96220b3d250cda7c8aaa7c
42876733b0f03042","iv":"
9592ab13889178b47bf75a20898b7b36","ciphertext":"
64aFQWU51ZkRLdwNTvTfSvg2jp07dBQVb0eP+KpCinGXnQYO
Bq/nkhNRh72jaS9jCQQ/SCLmHPjPg/8VNBLORazSBTFf/8Gd
URjn9+JYLyNHN+8LJ/RBipZqEgHSv31Go09evDmws6X0EYRy
rg+MxFKYqtvmtLLzHgmch6cb5sFYHy3WYrNfGR5YWyfJpMLf
RitdTzHjS5/EAOCu31qH2KuGOym3bpdLr8hGIu6JY3f6xAnN
Yd1o2WUn0dDd814a"}
```



✓ ~~Add Password Entry Field~~

✓ ~~Set and Confirm Password~~

✓ ~~Enter and Validate Password~~

✓ ~~Generate Key from Password~~

✓ ~~Encrypt todos~~

✓ ~~Store Encrypted~~

✓ ~~Decrypt todos and Check Integrity~~

Important Info (redux)

#MDC13

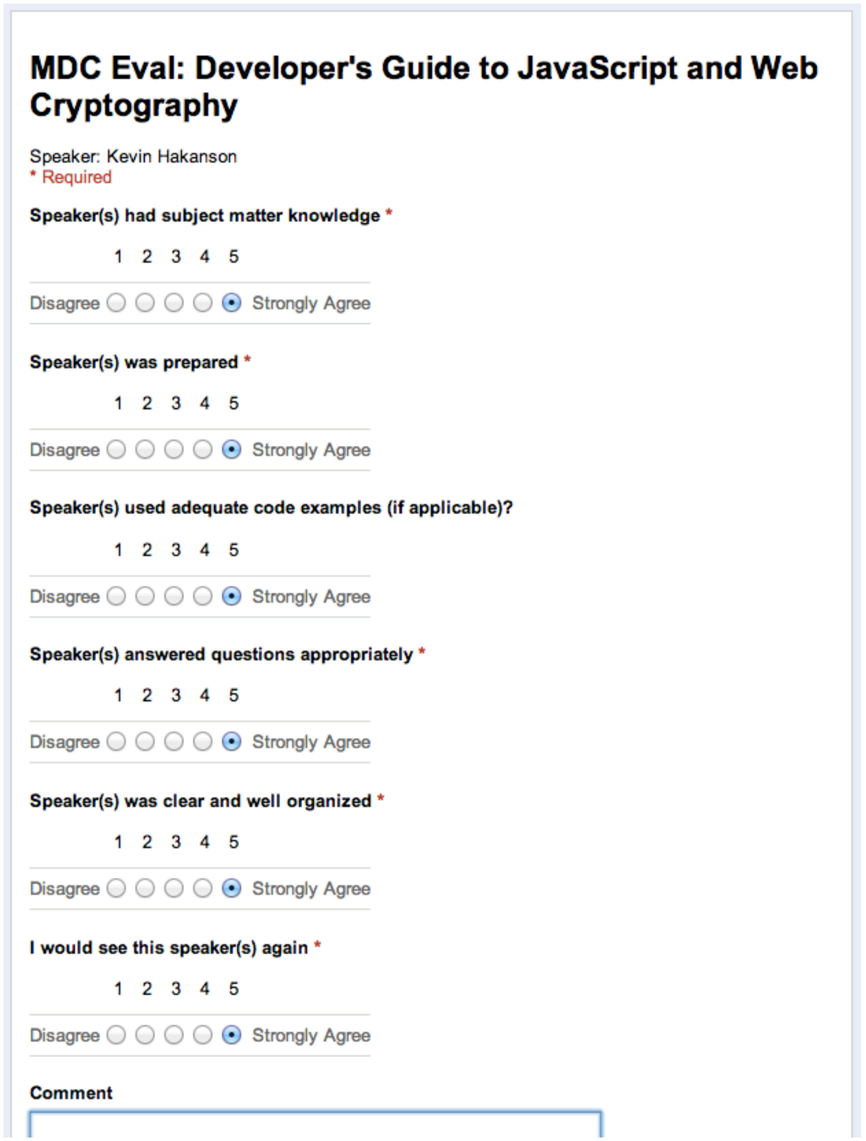
Speaker Evaluations

http://mdc.ilmservice.com/eval

Presentation Downloads

http://mdc.ilmservice.com/download

Happy Hour



MDC Eval: Developer's Guide to JavaScript and Web Cryptography

Speaker: Kevin Hakanson
* Required

Speaker(s) had subject matter knowledge *

Disagree 1 2 3 4 [✓ 5] Strongly Agree

Speaker(s) was prepared *

Disagree 1 2 3 4 [✓ 5] Strongly Agree

Speaker(s) used adequate code examples (if applicable)?

Disagree 1 2 3 4 [✓ 5] Strongly Agree

Speaker(s) answered questions appropriately *

Disagree 1 2 3 4 [✓ 5] Strongly Agree

Speaker(s) was clear and well organized *

Disagree 1 2 3 4 [✓ 5] Strongly Agree

I would see this speaker(s) again *

Disagree 1 2 3 4 [✓ 5] Strongly Agree

Comment

Questions?