MEGA web development test

Introduction

The purpose of this assessment is to gain a good overview of your development skills. It is not designed to test for a particularly hard programming or algorithmic task, but rather something that reflects non-trivial day-to-day work.

The size/scope of the task may seem large. It is intended to be that way to allow for a broad assessment of many aspects. If you feel that either the time or your current skill set does not permit the completion of the entire assessment, do not be alarmed. Focus on the things that you can achieve, and do them well; in examining your work we will consider not just the amount of the task that is complete but the quality of code as well.

The code you develop will be retained for the sole purpose of assessing your work. You retain the rights to the code developed.

Task Description

You are to build a simple local HTML5 web page to encrypt files and upload it to a simple backend server. The files will be between 0.1MB and 20MB in size and should be encrypted locally with a key derived from a pass-phrase, so that the files can not be decrypted without it. Your application should be the only one allowed to upload files to your backend server.

You have 4 hours to work on this project. It is not necessarily expected that you will complete the work within this time period.

Aside from cryptographic libraries you may require, our preference is for you to limit yourself on the Javascript front-end side to JQuery if possible, and not use extensive JS frameworks.

Core Requirements

The following requirements are given in order of importance:

- Limit yourself to 4 hours total, and document the time you spent on each core portion of the test.
- Style up a simple form submission page. This design is available as a PNG image file and there are separate image assets which you can use.
- The file uploader can select one file to be uploaded at a time. The file upload functionality should be implemented client side e.g. with the HTML5 FileReader API.
- The file uploader should show the progress of each file upload and a final confirmation when the file is uploaded.
- The server side should contain a simple API REST interface allowing you to upload the file to a directory on the Web server and return a success response and potentially a status message when completed.
- Failed uploads should be handled gracefully and an appropriate message displayed on the UI.
- Your work should be compatible with the current latest versions of Chrome and Firefox.

Bonus Requirements

If you do not understand cryptography, do not worry, just implement the file upload functionality as plaintext or focus on what you can do (e.g. functionality or styling UI elements). We are looking for developers with a range of skill sets and abilities.

- The files should be encrypted locally using a strong symmetric key cipher, before being uploaded to the server (end-to-end encrypted).

- The encryption key should be derived from the passphrase using a strong Password Based Key Derivation Function e.g. PBKDF2.

- The server side should only accept uploads from one authenticated user. The server may have a shared secret (e.g. key or passphrase) so you may consider using a simple challenge response protocol or simply provide a Message Authentication Code for each upload which the server can then verify. For example: HMAC-SHA-512(shared secret, nonce + data).

- File uploads and crypto operations should not block the user interface. The user interface should still be usable while these operations are taking place.

For the cryptography parts, you can also use whatever libraries (e.g. CryptoJS, SJCL, asmCrypto, etc.) and algorithms (e.g. AES, Salsa20, PBKDF2, Scrypt, HMAC, SHA-512 etc.) that you wish. However for speed and simplicity you can use the following code outlined here:

TweetNaCl.js
TweeNaCl can be used for simple encryption. The project can be found here:
https://github.com/dchest/tweetnacl-js

All parameters passed to the nacl.* crypto functions are of type Uint8Array

Example usage:
```
// Our text message to encrypt.
var textMessage = 'message to encrypt';

// Convert to format suitable for `nacl` (Uint8Array).
var message = nacl.util.decodeUTF8('message to encrypt');

// Generate nonce and key for symmetric encryption.
var nonce = nacl.randomBytes(nacl.secretbox.nonceLength);
var key = nacl.randomBytes(nacl.secretbox.keyLength);

// Encrypt.
var ciphertext = nacl.secretbox(message, nonce, key);

// Decrypt.
var cleartext = nacl.secretbox.open(box, nonce, key);

console.log('The two messages logged below should be identical:');
console.log('Original message: ' + textMessage);
console.log('En- and decrypted message: ' + nacl.encodeUTF8(cleartext));

// Some helpful utilities to convert keys/nonces/ciphertexts.
// - nacl.util.decodeBase64(string):
//   Decodes Base-64 encoded string and returns Uint8Array of bytes
// - nacl.util.encodeBase64(array):
//   Encodes Uint8Array or Array of bytes into string using Base-64 encoding.
```

asmCrypto.js

asmCrypto can be used to derive a cryptographic key from a password. The project can be found here: https://github.com/vibornoff/asmcrypto.js

Example usage:

```
// Use some value to salt the process.
var salt = userId + atob('YzNkN2E0MDVkYmM0ZmRjZiAgLQo=');

// Define some parameters.
var password = 'middleearth';
var iterations = 65536;
var dklen = 32;

// Derive a key suitable for encryption using PBKDF2.
// The key is in a format directly usable by `nacl` (Uint8Array).
var key = asmCrypto.PBKDF2_HMAC_SHA256.bytes(password, salt, iterations, dklen);
```