The security of JavaScript-based cryptography

PassLok is written in JavaScript. There is no way around it because it is designed to run locally within a browser, so it can run on any device and not communicate with any outside machine. But JavaScript has long been considered to be inherently unsafe for cryptography applications. Why?

Two words: code injection. Late in the summer of 2013, it was revealed that the NSA temporarily disrupted the operation of the TOR anonymizing network by this method. Simply put, code injection means that instructions and components that are not an original part of a webpage can be added later, usually by action of an external server. This is good in many cases, because it allows a webpage to change according to the content, as it downloads from the server. But it is terrible when it comes to cryptography.

If a server can send a new encrypting function to the client (your machine), and the user never knows that this has happened, what's to keep a rogue server from totally messing up the encryption, so it only appears to work, but not really? Then whoever prepared the messed up encrypting function would be reading all the supposedly encrypted messages, and nobody else would be the wiser.

This is the main reason why many experts are horrified at the idea of cryptography written in JavaScript, but PassLok neatly skirts the issue, even though it is written in JavaScript, because it doesn't talk to servers that could possibly inject malicious code, same as your mother told you to never talk to strangers.

Well, with a few exceptions, which are noted here and you can check for yourself by inspecting the source code:

1. PassLok loads initially from a web server (obviously). To make it harder for someone to intercept the send request and instead supply a modified copy of the code, PassLok loads only by https. The preferred server is outside US control, run by a libertarian group in Italy. Still, it is recommended that users check the authenticity of the code by obtaining the source's SHA256 hash and comparing it with the hash read by the author on YouTube.

2. That hash and the link to the video are in a mirrors.html page, which is loaded from the main PassLok page via a button. This page comes from the same server as the main code. Even though there is no video to double-check this file, its code is not the issue, but the authenticity of the links and the SHA256 hashes displayed on the page. These could be modified by code injection if an attacker hijacks any of the links. Not a game stopper since there are duplicates at other locations, but discretion is advised.

3. PassLok contains links to tutorial videos hosted on YouTube. All the links are https, but there is still a chance that a very powerful attacker might be able to

hijack one of those, and use it to inject malicious code. My recommendation: don't use PassLok's encryption functions right after clicking on a video, especially if you detect something funny in the way it loaded. Close the browser and reload, to make sure you've got again the original code.

If you follow these precautions, it will be pretty much impossible for anybody to mess with the security of PassLok. This is something that cannot be said of a regular program not written in JavaScript, by the way.

How so? First, because JavaScript code is human-readable, while compiled code is not. You can always display the code and read it, to make sure it's not doing anything unbecoming. Compiled code might be communicating with the outside without telling you anything, making it liable to an equivalent of code injection, while such behavior can be visually detected in JavaScript code.

Then, compiled code can be compromised at the source just as easily as a webpage, although it is usually trusted by default. When the code is served by an app store like Apple or Google, you are being forced to trust them, even though their noses in this respect are no longer clean. You are actually better off having a choice of where to get your code from and an easy way to check for differences.