Project Report: Encrypted Bookmarks Exporter Add-on for Firefox browser

Author: Anthony Epshtein

Introduction:

Nowadays web surfing is essential for every computer user.
Therefore, the bookmarks list in browser likely to contain personal info at dangerous scales.

The bookmarks exposing sites the user visiting on regular basis such as sensitive services (like banks, social networks) and habits (like saved articles, and more) revealing lifestyle of user.

I have noticed that the only options to export bookmarks in the most popular browsers is to export the bookmarks list into HTML file or JSON, which stores the bookmarks explicitly on local drive and allows even low-skilled computer user get access and read bookmarks list as text.
This is direct threat to digital privacy.

I developed this addon in attempt to secure personal data contained at exported bookmarks.

Unlike standard build-in-browsers exporting methods, it allows user to store bookmarks as backup file locally while preventing unauthorized access to stored bookmarks.
Also, the encrypted format makes is safer to transfer the stored bookmarks across devices and import them using physical storage only, such as usb drive.

What makes it more secure. Less transferring bookmarks through internet makes them less likely to be stolen by network sniffers. Thought, sniffers (man-in-the-middle) still can steal information about which site user entered at the moment, it will protect from use cases in which whole bookmarks list is synchronized across devices thorough internet.
As there is no guaranties that build-in-browser synchronization methods implement heavy encryption.

This way the bookmarks don't have to be transferred through internet while stays secure all the way to new device, i.e. bookmarks will not be exposed "as is" to curious people that got access to external physical storage.

Background:

As mentioned previously, two most popular browsers today (Chrome and Firefox) allow user to export bookmarks into readable html format, more precisely into netscape-html format.
Which is convenient to read and import back. But on the other hand, storage them like this in long term might be unsecure because general user will put this file somewhere in his filesystem system or on external drive and probably forget about this file. It rises a possibility that anyone else can just open it and read.

I did not find obvious competitors for this project idea. While there are existing solutions to secure bookmarks with password when browser in use. There is few or no solutions to secure exported bookmarks in few clicks.

Of course, there is file encryptors that could be installed on device and can encrypt any file.
Respectively, user must export bookmarks from browser, then process generated html into this encryptor and only then will have encrypted file. But what worse, when user will decide to import his bookmarks, he will be forced to make this process in reverse.

Most of users better decide to store exported bookmarks html file as is, because of complexity of process, while compromising their privacy.

This addon aimed allow user to make whole process of encryption in few clicks, hopefully make safer computer usage more accessible to regular users.

Project Design:

The final goal is to provide user friendly interface so the user can easily export his bookmarks with no effort.

In terms of limited time, I decided to focus on Firefox browser, mostly because I found enough documentation and sources that can support my research and implementation.
The support of Chrome browser requires additional research and code changes.
After achieving working prototype on Firefox, there is open possibility to extend app to support Chrome too, because Chrome and Firefox support apps in similar formats.
Yet there are differences in API. (Chrome requires slightly different manifest format alongside with API differences).

Technologies Used:
Firefox WebExtensions API - to interact with browser,
JavaScript - for implementing the core functionality,
HTML/CSS - for creating the user interface.

The project involved designing the add-on functionality and UI, more precisely implementing bookmarks export and import. And continuous debugging.
More about process in next chapter.

Implementation

I firstly focused on implementing the default export of bookmarks exactly as browsers do, because I did not found dedicated commands in API for exporting bookmarks into **netscape-html** format.
Also, support of this established format may allow to expand future features of add-on, hypothetically allowing to decrypt encrypted file and directly save into file if user require.

Alongside with that, I needed to create user interface integrated into browser environment. For that I referred to official library of add-on examples, there were examples how to interact with Firefox's API in **javascript**.

The browser environment runs mostly on **javascript** so there wasn't much choice in terms of language options. Also, most of add-ons that I have encountered used popup window to interact with user. It is commonly written with **HTML** and **CSS**.

Next, after successfully implementing the default **export** I was implementing **import** of the default **netscape-html** format when faced the critical bug.
I implemented the whole application to run inside **popup window** that is managed by browser itself and it **closes** whenever it loses focus (when user clicks outside popup window). It was unacceptable, the import process requires from user to choose the file inside their file system and for that javascript calls for OS's API which opens another window to choose file from.

As you can imagine, whenever user starts to choose file for import - the popup window closes and disrupts the importing process. There was no workaround because addons have no power over that particular popup window functionality.

I found alternatives for popup window in the official Firefox add-on examples. There was option to open whole app in standalone tab and another less popular option to run UI in the browser's **sidebar**. I chose to use sidebar because I wanted the add-on to be compact on the screen. I wanted users to experience the app as light addon and not full screen program inside browser. After I moved HTML that I created into side bar the add-on wasn't closing anymore when out of focus. Problem was solved.

For now, I had successfully implemented process :
[extract_bookmarks > save_into_file > read_from_file > add_bookmarks_in_browser]

Next task was to convert output_file into encrypted format.
For that I made research which encryption algorithms are used today, which is stronger than others and their unique features and downsides. I decided to use "Advanced Encryption Standard" but it has several implementations, each with their own features.

I knew that I need feature of “integrity” to ensure that bookmarks weren’t changed since encrypted. “Electronic Codebook” - ECB and “Cipher Block Chaining” - CBC modes of AES encryption is lack of integrity verification feature, along with AES-CFB and AES-OFB.
Next in consideration were GCM, CCM and OCB. I chose not to go along with OCB mainly because of legal restrictions around this mode and licensing.
I decided to use "**Advanced Encryption Standard with Galois/Counter Mode**" shortly **AES-GCM Algorithm**.

Among his features it worth to highlight the:
- high security compared to other algorithms especially when using 256-bit key, as I did.
- authentication tags that provides data integrity, in other words algorithm can and will check if the data in encrypted file was changes since encryption. I made sure of it by providing input of “tagLength: 128” into the encryption function. (It will concatenate final ciphertext with 128 bits of authentication tag string).
- also it is highly efficient, it does that by using multiple cores of device in parallel processing.
- using unique nonce per encryption to encrypt each block of data

Among the downsides is its complexity.
Also there is restriction to not use same nonsense more than once because it can compromise security of data.

I referred a lot to documentation of usage of AES-GCM .
Also, it required from me an extremely precise debugging process, and it was challenging to make AES-GCM work without way to check program with 100% valid input (the encrypted file). Especially in importing implementation.

There was no way to check myself if I encrypted it right or not unless I implemented both encryption and decryption process perfectly. What required a continuous debugging process and following each variable in console on each step of processing.

At the end, all known bugs were solved and addon is ready.

Now I had successfully implemented full cycle:
[extract_bookmarks > encrypt > save_into_file > read_from_file > decrypt > add_bookmarks_in_browser]

Strengths:

1. Reliability – the encryption method is very secure.
   1.1 Salt – salt was used during encryption to ensure security

   1.2 Strong key derivation algorithm – was chosen, Password-Based Key Derivation Function 2 with SHA-256 (a.k.a. PBKDF2 with SHA-256) as shown in source [10.3.1].

   1.3 Non-Extractable Key – achieved by setting "extractable" parameter to false. This means the derived key cannot be exported outside of encryption and decryption process.

2. Portability – the export method provides a single file that can be kept anywhere and using the same addon you can import it on any other device. (Into Firefox.)

3. Robustness against attacks –
   3.1 Brute Force Attacks - AES-GCM is known for resilient to brute force attacks. Large key size like 256 bits that add-on using makes brute-force attacks infinite long with current computing power.

   3.2 Replay Attacks – The nonce (a.k.a. Initialization Vector) ensures that even identical plaintexts encrypted multiple times will result different ciphertexts.

   3.3 Chosen Plaintext & Ciphertext Attacks - Integrity protection achieved with "Galois Message Authentication Code" defends against altering or swapping encrypted ciphertext.

   3.4 - Man-in-the-Middle Attacks – Thanks to Authentication Code, any unauthorized modification to the ciphertext is detected during decryption, preventing attackers from altering or swapping encrypted ciphertext.

4. High speed – achieved by parallel encryption on multi-core processors

Weak points:

1. Keyloggers – If system of user was infected with spyware specifically keylogger, nothing prevents hacker from decrypt the file using decrypting mode in original program using original password.
   As keylogger records user's input, he directly getting password from victim during encryption process.
2. Weak password – even the best encryptions in the world cant protect data if person uses obvious password like "1234" , the attackers can just try their luck or even better make smarter brute force attack that tries common passwords first. Strong password required.
3. Password loss – if user forgets password he used for encryption, his bookmarks archive will be kept extremely secure, even from user himself.

Further Improvements:

1. Include Chrome support as mentioned before. This way the add-on will reach more users.
2. Add additional encryption algorithms, to provide user a choice between algorithms that used.
3. Add option for direct processing of encrypted file into decrypted file without importing into browser. (if user want to)
4. Automated scheduled backups of encrypted bookmarks into local storage.
5. Improvements of user interface.

Conclusion:

Firefox add-on for exporting encrypted bookmarks was successfully developed.

The add-on meets its initial objectives by providing a user-friendly tool for easy backup and safely storing encrypted bookmarks.

Hopefully, the project contributes to society by making regular user's data safer than it was yesterday.

The project demonstrates a practical solution to the problem, though there is room for enhancements and expanding functionality.

It was an interesting challenge.

References and Information Sources:

Large Language Model (LLM's) disclaimer:
To ensure my understanding in the topic and field of cybersecurity I asked an ChatGPT AI model how particular encryption algorithms works and similar questions for that theme.

[1] Sources of official Firefox add-ons examples:
https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Examples

[1.1] Example of using **popup window**:
https://github.com/mdn/webextensions-examples/tree/main/beastify

[1.2] Example of creating **new bookmarks** in browser:
https://github.com/mdn/webextensions-examples/tree/main/bookmark-it

[1.3] Example of using **sidebar** and **file picker** in browser:
https://github.com/mdn/webextensions-examples/tree/main/imagify

[2] Temporary installation of add-on in Firefox:
https://extensionworkshop.com/documentation/develop/temporary-installation-in-firefox/

[3] Work with the Bookmarks API:
https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Work_with_the_Bookmarks_API
especially:
bookmarks.create ,
bookmarks.get ,
bookmarks.getTree

[4] Work with files in Firefox ():
https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Working_with_files

[5] querySelector() – select HTML elements
https://developer.mozilla.org/en-US/docs/Web/API/Element/querySelector

[6] nodeName – nodes of HTML hierarchy tree
https://developer.mozilla.org/en-US/docs/Web/API/Node/nodeName

[7] Manifest V3 in Firefox, rules and changes from Manifest V2:
https://extensionworkshop.com/documentation/develop/manifest-v3-migration-guide/

[8] JavaScript syntax
https://www.geeksforgeeks.org/how-to-access-the-value-of-a-promise-in-javascript/

[9] Netscape Bookmark File Format
https://learn.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/platform-apis/aa753582(v=vs.85)

[10] Web Crypto API
https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API

[10.1] CryptoKey
https://developer.mozilla.org/en-US/docs/Web/API/CryptoKey

[10.2] importKey()
https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto/importKey

[10.3] deriveKey()
https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto/deriveKey

[10.3.1] PBKDF2: derive AES key from password example
https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto/deriveKey#pbkdf2_derive_aes_key_from_password

[10.4] encrypt()
https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto/encrypt

[10.5] decrypt()
https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto/decrypt

[10.6] Pbkdf2Params
https://developer.mozilla.org/en-US/docs/Web/API/Pbkdf2Params

[10.7] AesGcmParams
https://developer.mozilla.org/en-US/docs/Web/API/AesGcmParams

[10.8] AesKeyGenParams
https://developer.mozilla.org/en-US/docs/Web/API/AesKeyGenParams

[11] JavaScript – binary data
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Typed_arrays

[11.1] ArrayBuffer
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer

[11.1.1] slice()
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer/slice

[12] TextEncoder - encode()
https://developer.mozilla.org/en-US/docs/Web/API/TextEncoder

[13] TextDecoder - decode()
https://developer.mozilla.org/en-US/docs/Web/API/TextDecoder