Cryptographic Libraries for CryptPad

Theo von Arx

March 14, 2023

Abstract

Changing the cryptographic primitives in CryptPad is complex and requires a lot of work to maintain backwards compatibility. For this reason, a possibly new cryptography library has to be chosen carefully.

Since its beginning, CryptPad has used the TweetNaCl-JS library to perform cryptographic operations. We re-evaluate it in terms of security and performance. We furthermore compare it against other popular libraries/APIs such as Libsodium.js and the Web Crypto API.

While NaCl is still considered to be secure and there are no known attacks, it lacks performance and native support on current browsers. Sodium provides more flexibility in terms of available algorithms (which are not necessarily desired for CryptPad). SubtleCrypto provides highly performant primitives, which differ from CryptPad's used algorithms. The performance gain is most-likely not big enough to justify the workload on adapting these algorithms immediately, but only when a bigger refactoring happens anyway.

Contents

1 Notation

We first give informal reminders about security notions required for CryptPad:

- **INT-CTXT** (Ciphertext Integrity): It is computationally infeasible to produce a decryptable ciphertext that has not previously been produced by the sender, whether or not the underlying plaintext is "new".
- **IND-CPA** (Indistinguishability of Chosen Plaintext Attack): No adversary, given an encryption of a message randomly chosen from a two-element message space determined by the adversary, can identify the message choice with probability significantly better than that of random guessing.
- **SUF-CMA** (Strong Unforgeabililty against Chosen Message Attack): No adversary can create a new signature for an existing message
- **Pre-Image Resistance:** For essentially all pre-specified outputs, it is computationally infeasible to find any input that hashes to that output.
- **Collision Resistance:** It is hard to find two different inputs that hash to the same output.

2 Problem statement

In short, the questions are: *If we deploy a new version of CrytPad's encryption schemes, should we use another library? And which?*

More specifically, the following cryptographic functions are required:

- Key derivation function (KDF) that is brute-force resistant (and thus memory and time intense), and that provides good entropy for low-entropy inputs.
- Authenticated symmetric encryption
- IND-CPA asymmetric encryption
- Asymmetric signatures that is resistant to selective forgery in known-message attacks and SUF-CMA
- Hashes that are pre-image resistant and collision resistant.

Further requirements are that the algorithms should be distributed in a maintained, free JavaScript library, fast loadable and compatible with Firefox, Chrome and Safari. Except for the KDF, all functions should execute quickly.

3 Libraries

Table 1: Provided algorithms by various libraries. The Web Crypto API is implemented differently for every browser.

Library	Audit	Last Update	Algorithms
TweetNaCl-JS	2017	06/2022	X25519-XSalsa20-Poly1305, XSalsa20-Poly1305, Ed25519, SHA-512
libsodium.js	2017	01/2023	XChaCha20-Poly1305 XSalsa20-Poly1305, X25519-XSalsa20-Poly1305, Ed25519, Ed25519ph Blake2b, Argon2, Scrypt
Web Crypto API	*	*	RSA-OAEP, AES-CBC, AES-GCM SHA-256, SHA-384, SHA-512 ECDH, HKDF, PBKDF2

3.1 Tweet-NaCl

TweetNaCl-JS is the library that is currently in use for CryptPad. The algorithms have been described by Bernstein *et al.* in 2014. The library [2] is written and maintained by Dmitry Chestnykh, is licensed under the GPL-compatible *Unlicense* and was last audit by Cure53 in 2017.

The provided algorithms can be seen in Table 1. They match all the requirements and are considered to be secure. While there have been some practical collision attacks against *truncated* SHA-512 [3], SHA-512 is nevertheless listed by NIST [4].

To sum up, Tweet-NaCl is considered to be secure and to meet CryptPad's requirements. Hence, the only reason to change it would be performance or quantum-resistance.

3.2 LibSodium

Libsodium "is a portable, cross-compilable, [...] fork of NaCl, with a compatible API" [5]. The library was last audited Private Internet Access in 2017, is licensed under the GPL-compatible ISC License and provides a library compiled to WebAssembly [6].

Libsodium provides a superset of the algorithms implemented by TweetNaCl-JS (c.f. Table 1). The library thus offers much more freedom allowing a specific choice of algorithms as well as generic interfaces for which the underlying algorithms can be updated. However, this freedom is not necessarily

desirable for CryptPad: since ciphertexts are stored permanently, the algorithms cannot be easily exchanged. A newer version could modify the default behaviour and therefore break existing documents.

3.3 Web Crypto API

The Web Crypto API [7] is a standardized JavaScript API supported by current versions of all major browsers (Firefox, Chrome, Safari). SubtleCrypto, which is part of the Web Crypto API, provides interfaces for symmetric and asymmetric encryption, signing, and hashing (c.f. Table 1). The API also provides a lot of flexibility in terms of choosable algorithms. This might be tricky, since, e.g., AES-CBC does not provide authentication, but could be used in combination with signatures.

The main advantage of the Web Crypto API is the native support by browsers. Not only does this eliminate any loading time of library code, but also gives access to hardware-optimized functions.

However, all functions are implemented *asynchronously* and therefore can only be called from within `async` functions. CryptPad currently calls all functions from within synchronous functions. Therefore this library is not a drop-in replacement, not even for identical functions such as Web Crypto's SHA-512 and TweetNaCl-JS' SHA-512.

4 Discarded Libraries

4.1 Stanford Javascript Crypto Library

While this library seems to be quite popular and is written by renowned experts, it does no longer seem to be maintained: the last commit was from July 2019 and the last release from November 2018 [8]. Furthermore, the library provides only low-level functions.

4.2 CryptoJS

This library is also very popular [9] and has seen its last commit in September 2021. It provides AES-256 in several modes (among which are CBC, CTR, and ECB), but not in GCM mode. Hence, there is no authenticated encryption. Furthermore, the library does not provide any digital signatures.

5 Performance Comparison

Figure 1 shows the performance of the different libraries mentioned above on Firefox, Chromium, and Webkit. The measurements where done using Playwright on a Lenovo ThinkPad X1 Carbon 5th Generation (Intel® Core™ i5-7200U CPU @ 2.50GHz × 4), the actual results might therefore vary

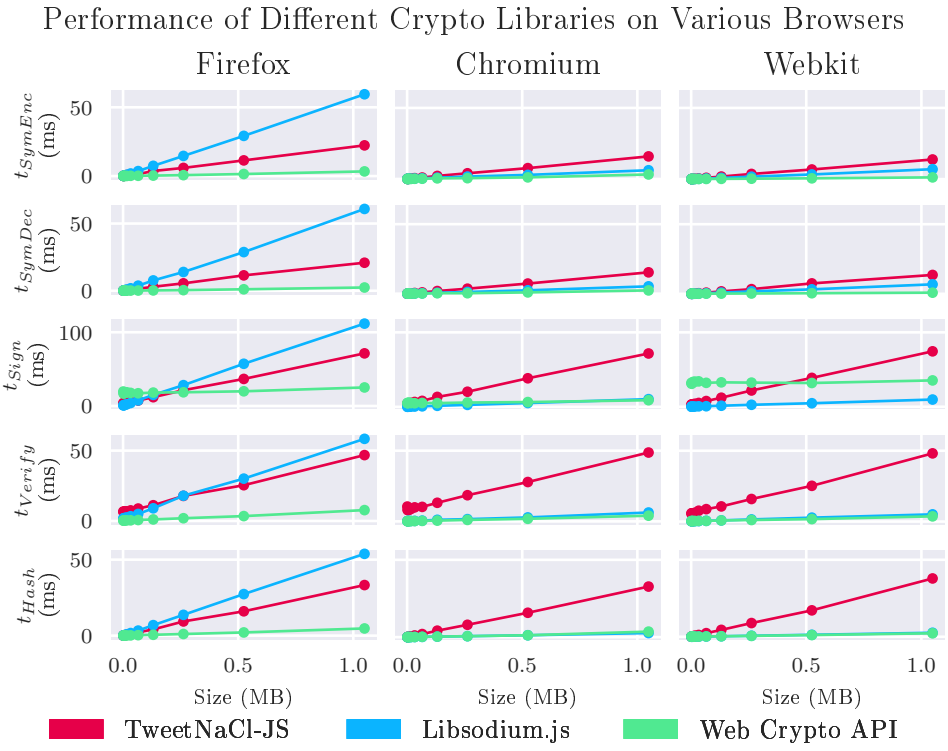

Figure 1: Web Crypto API outperforms the other libraries in the vast majority of scenarios (lower is better).

for different platforms. Nevertheless, the figure gives a raw indication to compare the performance.

For better visibility, we only show the default Libsodium functions, which are XSalsa20-Poly1305, Ed25519, and Blake2b. Similarly, we show only AES-GCM, RSA-PSS, and SHA-512 for the Web Crypto API.

When observing Fig. 1, we see that almost all functions run slower on Firefox. Especially Libsodium performs really poorly on Firefox compared to its performance on other browsers.

For symmetric encryption, TweetNaCl-JS and Libsodium.js are the slowest libraries with a runtime of up to 60 ms. The AES-GCM of the Web Crypto API performs much better and nearly independent of the input size. It is up to 19 times faster than NaCl (when decrypting 1MB on webkit).

For asymmetric encryption, TweetNaCl-JS performs worst while Libsodium.js and Web Crypto API perform about similar. The figure suggests that Webkit does have a bad support for RSA-PSS verification.

For hashing, the Web Crypto API function also outperforms the other

two. Its running time is (especially compared to the one of TweetNaCl-JS) nearly independent of the input size. While Libsodium.js shows a good performance on Chromium and Webkit, its performance on Firefox is even worse than the one of TweetNaCl-JS.

6 Conclusion

Since TweetNaCl-JS still fulfills the security requirements, and there are no post-quantum safe libraries yet, the only reason to replace it would be performance. In this point, the Web Crypto API is the clear winner. However, drawbacks are the increased flexibility as well as the requirement of asynchronous caller functions. While the first can be mitigated by writing proper wrappers, the second can only be addressed during a major rewrite of CryptPad's cryptography code and all its caller function. Rewriting them to be asynchronous gives room for potential more optimizations and might therefore be seen as an opportunity.

References

[1] D. J. Bernstein, B. van Gastel, W. Janssen, T. Lange, P. Schwabe, and S. Smetsers, "TweetNaCl: A Crypto Library in 100 Tweets," in *Progress in Cryptology - LATINCRYPT*, 2014.

[2] D. Chestnykhm, D. Mandiri, and AndSDev, *Tweetnacl.js*, 2016-. [Online]. Available: `https://github.com/dchest/tweetnacl-js`.

[3] C. Dobraunig, M. Eichlseder, and F. Mendel, "Analysis of SHA-512/224 and SHA-512/256," in *ASIACRYPT 2015*, 2015.

[4] M. Dworkin, *Hash Functions*, National Institute of Standards And Technology, 2022. [Online]. Available: `https://csrc.nist.gov/projects/hash-functions`.

[5] F. Denis, *Libsodium*, 2013-. [Online]. Available: `https://github.com/jedisct1/libsodium`.

[6] A. Ben Mrad, F. Denis, and R. Lester, *Libsodium.js*, 2015 -. [Online]. Available: `https://github.com/jedisct1/libsodium.js`.

[7] D. Huigen, M. Watson, and R. Sleevi, "Web Cryptography API," World Wide Web Consortium (W3C), Tech. Rep., 2022. [Online]. Available: `https://w3c.github.io/webcrypto/`.

[8] E. Stark, M. Hamburg, and D. Boneh, *Stanford Javascript Crypto Library*, 2015-. [Online]. Available: `https://github.com/bitwiseshiftleft/sjcl/issues/253`.

[9] J. Mott and E. Vosberg, *CryptoJS*, 2009-.