Security Recommendations for CryptPad 5

Theo von Arx and Aaron MacSween

March 14, 2023

Contents

1 Introduction

During the writing of the cryptography white paper, we discovered multiple issues and limitations in the current version of CryptPad. While none of them are classified severe, requiring immediate action, or posing a serious threat for users, they nevertheless should be addressed.

In this document, we list the problems, show their consequences, suggest how to fix them and outline possible drawbacks. We classified them into three different categories: short-term improvements (c.f. Section 2) that can be done immediately within the current version of CryptPad, mid-term improvements (c.f. Section 3) that require the introduction of a new version of the encryption schemes, and long-term improvements (c.f. Section 4) that can only be addressed with architectural changes.

2 Short-Term Improvements

In this section, we list improvements that can be done without requiring bigger changes of the underlying encryption schemes or architecture.

2.1 Update TweetNaCl

Problem: CryptPad uses TweetNaCl [1], [2] version 0.12.2 (September 18, 2014) while the newest version is 1.0.3 (February 10, 2020).

Consequences: CryptPad misses optimizations and security fixes introduced in later versions. In the current version, `nacl.sign` or `nacl.sign.detached` could create incorrect signatures. Moreover, CryptPad it might be hard to update TweetNaCl in case of a severe security issue.

Suggestions: Update TweetNaCl to the latest version.

Drawbacks: TweetNaCl has refactored some of the utilities into a separate library and now properly validates the base64 padding before decoding. An update therefore requires significant work.

2.2 Name of Team Roles

Problem: The four roles in a team are called *viewer*, *member*, *admin*, and *owner*. The role *member* conflicts with the common notion of "being a member of a team".

Consequences: Users are confused about the difference and precise wording in documentations is difficult.

Suggestions: Replace *member* with *author*.

Drawbacks: The notion of an *author* may not smoothly match the context of creating folders or drawing in a whiteboard.

2.3 Thumbnails of Destroyed Documents

Problem: When an image is destroyed, the thumbnail can be restored using the history function.

Consequences: The image is not fully deleted which is not what users expect.

Suggestions: When destroying an image, also destroy the thumbnail. In case that the thumbnail is needed to browse the history of a folder, replace the thumbnail with a generic one.

Drawbacks: -

2.4 Asynchronous Cryptography Functions

Problem: The current cryptography functions including encryption and key derivation (such as `ViewCryptor2`) are synchronous and do block all resources.

Consequences: No operations can be done in parallel to cryptography functions. Moreover, natively supported asynchronous functions (e.g., `SHA-256`) that would bring a speedup cannot be used.

Suggestions: Opt for a more general interface in which cryptography functions are asynchronous

Drawbacks: The additional workload coming from parallel execution may be too resource-intensive on some devices, such as smartphones. Refactoring large parts of the code to make it asynchronous is tedious.

2.5 Enforcing of Self-Destructing Links

Problem: The destroying of a document that was shared with a *view-once-and-destroy* is not enforced by the server.

Consequences: A malicious user can open such a link, but decide to not destroy the document. However, it is hard to come up with a scenario in which a malicious user could not achieve the same by simply taking a screenshot of the document.

Suggestions: Enforce the destruction on the server side.

Drawbacks: Although the server sent the content, there is no guarantee that the user actually received it (e.g., due to connectivity problems or an active network attack). Hence a document could get destroyed before being accessed the first time.[1]

2.6 Contact Verification

Problem: There is no easy way to verify whether a CryptPad contact corresponds to the expected person. Profile pages contain the public key of the users, but they are not signed and hence spoofable.

Consequences: Attackers can impersonate a person and gain access to teams, documents and folders.

Suggestions: Implement a method that allows users to verify their contacts and whether a conversation is secure. One approach could be to use safety numbers [3] to let users verify whether the binding of the public key to the user is correct. It is important to provide a clean interface that is easily understandable for users and invites them to actually perform the contact verification.

Drawbacks: -

2.7 Prevent replay of old patches

Problem: Malicious users with read access can replay old patches drafted by other users. Since they have a valid signatures, they will be accepted and reset the document to an older state.

Consequences: Documents are reset and a lot of traffic is generated which might result in a Denial of Service (DoS).

Suggestions: There are multiple approaches:

1. Add a time stamp to patches and accept them only if they are within a certain range to the current time. The time stamps must be in the signed part of the message to make them unforgeable.
2. The server keeps a list of the hash of the last N accepted messages. Every incoming message is valid if its hash is not in the list and if it references the hash of one of messages in the list.

[1] Currently, a destroyed document is only achieved, and could therefore be restored by instance admins).

3. Clients that want to send patches have to first prove to the server that they are allowed to do so. For that, the server sends them a random value (a *challenge*) which they must sign and return. If they succeed, they are marked to have write capabilities and the server accepts their patches.

Drawbacks: The drawbacks for the different approaches are the followings:

1. Users with a wrong system time won't be able to produce new patches.
2. - The mechanism is fairly complex for the server and N has to be chosen carefully, and dependent of the number of users with right access (the more there are, the bigger the chance of simultaneous messages that could result in missing the window).
   - Some types of data structures built on top of channels (such as mailboxes) contain sequences of messages which are independent of each other. For these types of channels it could be inconvenient to have to know the parent patch. Replay protection could therefore be enforced selectively with an attribute set in the channel metadata.
3. The challenge/verification has to be s.t. a DoS attack (state exhaustion) against the server is not possible.

2.8 Concurrent Uploads

TODO: Aaron

Problem:

Consequences:

Suggestions:

Drawbacks:

2.9 More Entropy for Login Keys

Problem: Generally, neither the username nor the password provides good entropy. The only rule imposed on the password is that it should have a minimum length of 8 characters. Therefore, weak passwords such as `password` are accepted.

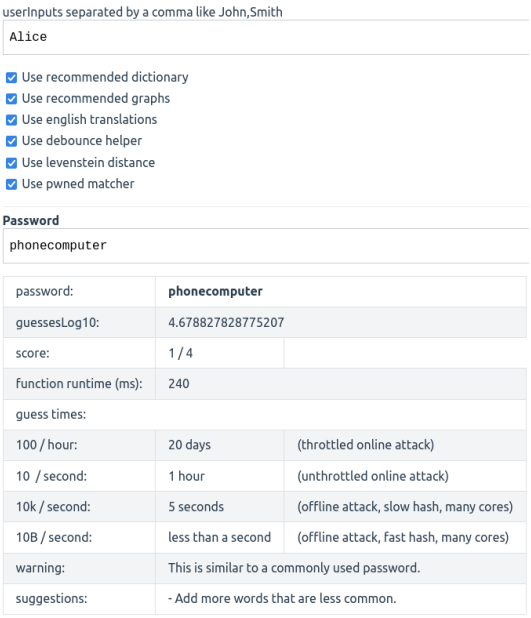

Figure 1: The library `zxvbn-ts` provides a score, guess times, warnings, and suggestions.

Consequences: We use `Scrypt(password, username + InstanceSalt)` to generate the login keys (master keys). Since these master keys are not generated from a good source of randomness, attackers can gain access to all documents of users with weak passwords.

A side consequence is that the probability to register a new account with the same `username` and `password` combination is not negligible. If this happens, then users without bad intentions get access to a third person's drive.

Suggestions: Use password strength estimation such as `zxvbn-ts` [4], [5] which rates the password strength on a score from 0 to 4 and provides, guessing times, indicators what makes the password weak and how it can be improved (see Fig. 1). We may display the password strength estimation and/or require a minimum score.

Drawbacks: Users are annoyed by password requirements.

2.10 Limit Login Block Size

Problem: There is no maximum login block size enforced by the server.

Consequences: Attackers can run a DoS attack on the server's memory by uploading large data as login blocks. The initial login block has to be signed which may impose too much computational work for the attack to be successful. However, the initial login block can be replaced in which case only the corresponding public key has to be signed.

Suggestions: Impose a limit on the login block size.

Drawbacks: The username is part of the login block. Therefore the size is not bound theoretically, but only practically (e.g., username not bigger than 100 characters).[2] Moreover, the login block size might grow in the future in which case the limit has potentially to be adapted.

2.11 Instance salt is rarely set

Problem: Instance administrators need to set a custom `loginSalt` before running CryptPad in a production environment. This value is supposedly only rarely changed from the default.

Consequences: The `loginSalt` makes it such attackers who want to brute-force common credentials must do so again on each CryptPad instance that they wish to attack. If `loginSalt` is the default one, then there is no protection against this.

Suggestions:

1. Mention the `loginSalt` more prominently in the documentation.
2. Write an installation script that initializes this salt to a random value.

Drawbacks:

3 Medium-Term Improvements

In this section, we outline improvements that should be addressed with the introduction of a new version of the encryption scheme.

3.1 Fix Bit Overlap in ViewCryptor2

Problem: In the function `createViewCryptor2`, there is a bit overlap of 128 when generating a symmetric key and a signing key pair.

[2] It is unclear, whether the username is needed to be in the login block at all. This has to be further investigated.

Consequences: An adversary having only access to one of the keys has better chances to guess the other one. Since there are still 128 independent bits, the vulnerability is not considered severe, but should be fixed in a future version.

Suggestions: There are several ways to resolve the issue:

1. Increase the size of the seed to have enough bits to make the keys independent.
2. Hash bytes 16 to 64 to get again a string of 64 bytes.
3. Use a consumer-based programming pattern (similarly as it is already done for the credentials) that can produce "infinitely" many pseudo-random bytes. This essentially hinders us from making such mistakes in the futures.

Drawbacks: This requires to introduce a new version of encryption while keeping the old one for backwards compatibility.

3.2 Deduplicate Code

Problem: The functions `createViewCryptor2` and `createFileCryptor2` essentially provide the same functionality: provide a read-only access to a document.

Consequences: It is harder to understand and maintain what the two functions do.

Suggestions: Merge the two functions and make them callable with and without a keyStr.

Drawbacks: The chanID of `createViewCryptor2` is 16 bytes long, while the one of `createFileCryptor2` has 24 bytes. Furthermore, `createFileCryptor2` does not use the secondary signing key pair that is produced by `createViewCryptor2`.

3.3 Transition to New Encryption Schemes

Problem: When a new CryptPad versions introduces new encryption scheme, old documents are not updated to this.

Consequences: If a vulnerability in the encryption scheme is discovered, old documents can not be protected other than being destroyed. To keep the content, users would need to manually export and import it.

Suggestions: Implement an upgrade mechanism to the latest encryption scheme that can be activated by users. Ideally, the old links are still valid. To achieve this, the new keys can be generated from the same seed as the old ones and the old URL can redirect to the new one.

Drawbacks: Users might get confused, and old links may be broken after the upgrade.

4 Long-Term Improvements

In this section, we discuss improvements that require an architectural change to address the underlying problems.

4.1 Reduce Trust on Server

Problem: Users have to trust the server to deliver the correct client code.

Consequences: The threat model is reduced to a honest-but-curious server. Without this limitation, CryptPad could defend against an attacker with more active capabilities.

Suggestions: There are multiple approaches:

1. Providing an API so that a client can be distributed over an independent channel (Appstore, GitHub, Browser extensions, ...).
2. Trust on First Use (TOFU): users are warned when their client code changes and are asked whether they want to accept the update or not.
3. Sign the client code and allow users to verify the signature.

Drawbacks:

1. It gets more complicated to use CryptPad, when users first need to download a client. Hence, the webclient should still be accessible.
2. Currently, all users have the same client version. If this is no more the case, there might be incompatible features, or even "client fights", i.e., clients are resolving patches differently and thus always trying to push their version of the document.
3. Some measures such as signed code may only be effective if the users verify them. As experience in the context of HTTPS verification with extended validation have shown [6], this might not be the case.
4. The code should not only be verified for the flagship instance, but also for custom instances. However, they might want to customize the client code.

4.2 Team Key Revocation

Problem: The keys of team drives are static and will never change.

Consequences: Once a user is kicked from a team, the user does not lose the keys. Since are not prevented from subscribing to any Netflux channel, they can still receive all messages and also decrypt them with their kept keys. Hence, they do technically not lose access to the team drive.

Suggestions: When someone is kicked from a team, the admin has to automatically create a new key pair and distribute it *privately* to all members.

Drawbacks: -

4.3 Traitor Tracing

Problem: Everyone having read/write access to a document or folder can forward the keys without being noticed by anyone else.

Consequences: The access to folders and documents can be delegated without being noticed. As such team members have partial admin rights since they can share (but not deny) access to documents.

There is currently no way to find who leaked a document.

Suggestions: Deploy a scheme inspired by broadcast encryption [7] which generates new decryption and signing keys for every person allowed to access the document. CryptPad's setting is different from a traditional broadcasting one (i.e., communication over satellites where anybody can listen). This allows to simplify things, e.g., the server can enforce an access list. Similarly, there is no central node broadcasting, but potentially many authors interacting with each other. Among these authors a central user/group is responsible for managing key distribution.

Once a document is leaked, this allows to trace the traitor, i.e., find who leaked the keys, and to block the leaking keys from getting access. In the same way, per-user signing keys could allow for "traitor-tracing" and revocation of particular edits in the event of a vandal. The pure knowledge that this is possible will already discourage leaking keys.

Drawbacks:

1. Traitor tracing contradicts plausible deniability.
2. One has to be carefully, who should be able to do the traitor tracing. Possible are: owners, everyone with access to a document, or everyone. The last should be avoided in favour of privacy.

3. Tracing can easily be circumvented by importing/exporting a file or taking screenshots. However, this only leaks *read* access.

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] M. Marlinspike, "Safety number updates," *Signal Blog*, 2016.

[4] D. L. Wheeler, "Zxcvbn: Low-Budget Password Strength Estimation," in *USENIX SEC '16*, 2016.

[5] MrWook and D. L. Wheeler, *Zxcvbn-ts*, 2022. [Online]. Available: `https://github.com/zxcvbn-ts/zxcvbn`.

[6] G. Keizer, "Chrome, Firefox to expunge Extended Validation cert signals," *Computerworld*, 2019.

[7] A. Fiat and M. Naor, "Broadcast Encryption," in *CRYPTO '93*, 1993.