CryptoBook: An Encrypted Journal

Source Code: https://github.com/ZackUhlenhuth/cryptoproject

Donald Little, Nicholas Uhlenhuth, Zachary Uhlenhuth, Rachel Wang
{dmlittle, nicku, zacku, rswang}@mit.edu

May 13, 2015

1 Introduction

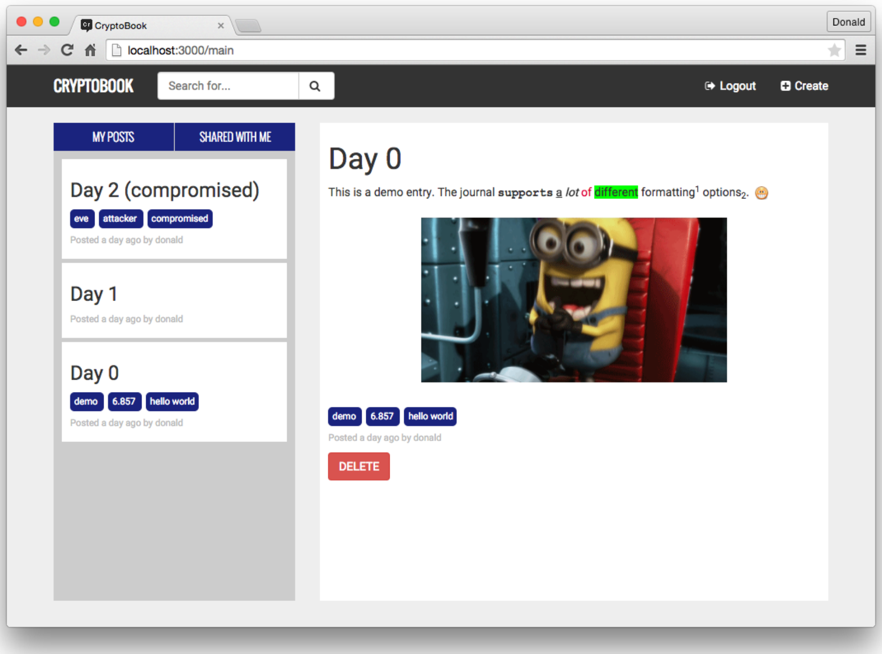
CryptoBook is an online, encrypted journal that allows users to securely store notes, journal posts, and ideas. Through encrypting content, plaintext tags and titles, a rich text editor, and a simple sharing scheme, users can conveniently write securely online and keep their content private from attackers, the government, eavesdroppers, and other uninvited readers.
Each CryptoBook user has a journal in which posts can be written. Posts are encrypted client-side with AES-256 using a custom, personal cipher key before being stored in the server. Keys are stored neither in local storage nor in the server's database, offering complete security of users' posts but requiring users to remember their passwords in order to decrypt the data contents. Notes are further authenticated through the use of HMACs to prevent man-in-the-middle attacks and database tampering. CryptoBook also allows users to share notes with selected users through RSA-OAEP public-key encryption. Shared notes are digitally signed to validate the integrity of shared posts.
Additionally, CryptoBook employs a number of additional security mechanisms to ensure complete confidentiality and authenticity of users' private posts including bcrypt to protect users from brute-force attacks, CSRF tokens to prevent unauthorized actions, and SSL ceritficates to ensure secure connections.

2 Goals and Objectives

The goal of CryptoBook is to provide a cryptographically secure journal where users can save their data knowing that it will not be compromised if the application is attacked. The journal supports all types of text-based information, images and embedded videos (hosted in third-party servers) and allows users to access it anywhere on-the-go as long as they remember their credentials and cipher keys. Users are also able to share their posts with other users through an encrypted sharing scheme that utilizes public-key encryption.

2.1 Motivation

Physical diaries are a secure means of storing private notes, but rather archaic in the modern world. Our aim is to combine the privacy of a physical diary with the availability and convenience



Day 0

This is a demo entry. The journal `supports` a *lot* of different formatting^1 options$_2$.

demo 6.857 hello world

Posted a day ago by donald

Figure 1: CryptoBook

of the Internet. There is an abundance of online blogging and journaling applications, although no current solutions provide true security and privacy. CryptoBook allows users to achieve privacy and security —encrypted entries are private from the public, their friends, the government, and application admins.
Additionally, an online journal provides searching capabilities that are otherwise cumbersome with physical journals. Our solution aims to maintain the convenience of online journals by allowing users to easily locate a previous entry despite fully encrypting the content.
The last major advantage that CryptoBook accomplishes is secure sharing. Online applications make sharing easy; most blogging entries allow users to set posts as public or private. Bloggers can often also publish their post unlisted and obtain a link to distribute to intended readers. Our final goal is to allow users to share specific entries with other users securely.

2.2 Related Work

No solutions currently address all motivations detailed in the previous section. There are abundant blogging applications, such as WordPress, Tumblr, and Blogger, but these applications do not provide ample security. Once an attacker has gained access to a user's blog, they can view all the users' posts as well as create and edit posts as the user. Wordpress has an Encrypted Blog plugin in Alpha version that allows users to encrypt blog posts and pages in the client-side before sending it to the Wordpress database. However, there have been few installs of the plugin and reviews

mentioning that an unencrypted version of the blog post is sill available.
There also exist a number of encryption tools available for users to encrypt data. However, these tools are rarely packaged with a secure data store or other capabilities. For example, hacker Vincent Cheung, has created a Javascript Encryption tool with the original intention of encrypting blog posts. Cheung uses client-side 256-bit AES encryption.
Finally, there are a few instances of encrypted diaries but that use AES encryption to protect journal entries. Monkkee is an encrypted diary, in which all journal entries are encrypted with a user's account password. The data is encrypted client-side, and the encrypted data is then sent to the server. Penzu is also an online private journal. They provide a charged Pro service, "Military Encryption," that uses 256-bit AES encryption to protect journal entries. No applications currently exist to share notes with others securely. Emails, social media, and public blogs are current outlets to share notes with others online but are vulnerable to man-in-the-middle attacks and are not secure as data is usually stored in plaintext on the server.

3 Design

Each user's journal consists of a collection of notes either written by the user or shared with the user. The security of the system relies on not storing journal content plaintext nor keys in the server. Users can create two types of posts: private and shared posts. Private posts are protected with AES Encryption and HMACs while shared posts are protected by RSA public-key encryption and signatures.

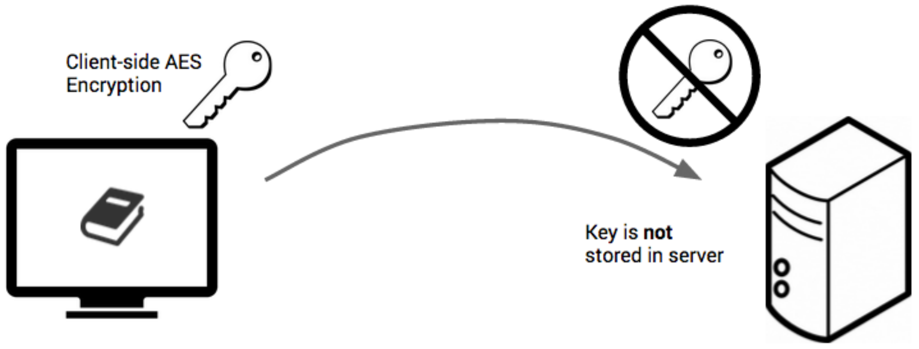


Figure 2: The main security mechanism in CryptoBook stems from the fact that posts are encrypted before being sent to the server, which stores no cipher keys.

3.1 System Architecture

The application is built on a Node.js server with a MongoDB data store. The application back-end maintains the data models and supports reading and writing from the database, but does not perform any computationally intensive work. All cryptographic computations are performed through client-side Javascript. Cryptographic Node.js libraries including Cryptico and Bcrypt are used throughout the application to provide encryption and decryption functions. The server and client communicate through a secure HTTPS connection.

3.2 User Accounts

User accounts accomplish two goals: to maintain each user's journal separately and to store each user's public key for shared posts. Users must register an account before using the application. The application uses Express.js sessions middleware to establish communication between users and the server. Users establish a session upon login or signup, which is then destroyed upon logout; they are redirected to the login screen as shown in Figure 3 whenever unauthorized.

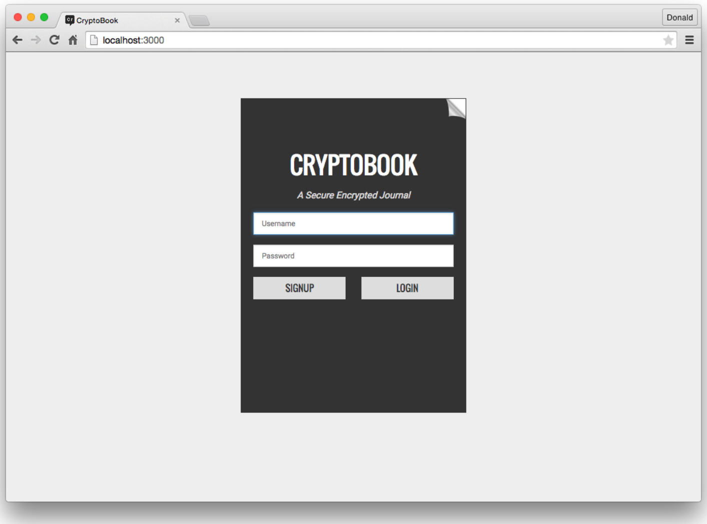


Figure 3: CryptoBook Login Screen

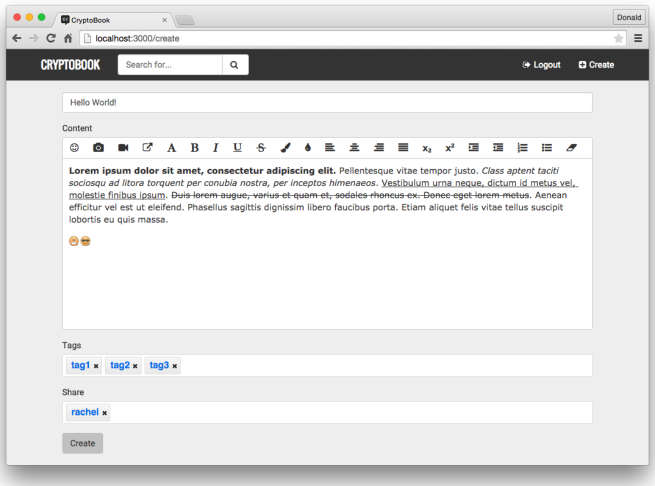
A user is identified by their username, password, and public key as shown in Figure 4. This data schema allows users to remain anonymous as they are not required to use personal identifying information while using the application. To gain access to the application journal and API, users must be authorized through their username and password. Upon registration a set of private and public keys are generated for the user based on the account credentials. Only the public key is stored in the server while the private key is never stored and needs to be regenerated by the user whenever needed.
Users' passwords are encrypted using bcrypt before the hashed password is stored in the server. As an adaptive key-derivation function, bcrypt protects users' accounts from brute-force password attacks by becoming more computationally expensive as users make multiple attempts to login. Bcrypt uses a modified version of the Blowfish encryption algorithm's keying schedule and makes hash functions five orders of magnitude more expensive than MD5 hashing. [1]
While bcrypt helps mitigate computationally powerful attackers from gaining access to a user's account, gaining access is not entirely useful for attackers. Access to a user's account only reveals the user's private and shared post titles and tags. Once authorized, the user's password can also be used to decrypt shared posts. However, a user's private posts are still secure as they are encrypted with separate cipher keys.

```
{
  _id : ObjectId("5550469293b886ef68b32c78"),
  username : "alice",
  password : "$2a08$5Yyq0Y...",
  publicKey : "fN89pcyERScvLl..."
}
```

Figure 4: The above figure illustrates the schema representing a User in CryptoBook. Users are identified by a user-friendly username and no other personal information. The password is hashed with bcrypt before being stored in the database, and the public key is used to share posts with that user.

3.3 Posts

Users store their thoughts, ideas, and notes as posts. The creation of an entry is kept locally on the client and the plaintext of the content is never transmitted over the internet. Only encrypted content is ever saved to the database. If the entry is not encrypted or saved, it will be deleted and destroyed.
Entries are composed in an HTML text editor as shown in Figure 5. The text editor supports all HTML tags, allowing users to style and format their plaintext. The editor also supports image uploads and embedded videos. Images are immediately converted to bitmap using a client-side Javascript function, allowing users to embed images directly on the application without hosting them on a third-party server. Users can also embed images and videos hosted online, although the content will not be encrypted at the hosted URLs. Entries have a maximum content size of 50 MB due to storage constraints.



Hello World!

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque vitae tempor justo. *Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos*. Vestibulum urna neque, dictum id metus vel, molestie finibus ipsum. ~~Duis lorem augue, varius et quam et, sodales rhoncus ex. Donec eget lorem metus~~. Aenean efficitur vel est ut eleifend. Phasellus sagittis dignissim libero faucibus porta. Etiam aliquet felis vitae tellus suscipit lobortis eu quis massa.

Figure 5: CryptoBook entry creation page with a rich HTML Text Editor. The page also supports the addition of tags and specifying which users the entry should be shared with.

Posts can be easily searched by their title, date, and tags, which serve as plaintext identifiers to help the user locate desired posts as shown in Figure 6. The hint also allows the user to optionally store a password hint to help manage each post's password. Posts are tied to authors through their MongoDB Object ID. Finally, posts store a message authentication code upon creation to ensure data integrity when the user decrypts the data.

```
{
    author: ObjectId("..."),
    title: "Hello World",
    content: "cAFFHRm7T1W6/m...iTAFbxr77bNx24eN",
    hint: "password hint",
    date: "2015-05-11T03:12:29Z",
    mac: "33348317534c...36f830c",
    tags: ["sports"],
}
```

Figure 6: The above schema shows the Post data model stored in MongoDB.

3.4 Private Posts

Posts written by and intended for the same user (i.e. private posts) use symmetric key encryption.

3.4.1 Encryption

When an entry is saved, a user is prompted to provide cipher key. The plaintext (entry's content) will be encrypted using 256-bit AES in CTR mode with the key provided by the user.

$$E_K(\text{html\_content}) = \text{ciphertext}$$

This encryption happens via a client-side Javascript library. The ciphertext (encrypted content) is then sent to the server where it is saved in a MongoDB database. As discussed in Section 3.3, users can include a title, tags, and a password hint to help remember the contents of the post as well as the cipher key.

3.4.2 Decryption

When decryption a post, the user is prompted to provide the cipher key to view a post's contents. Again, the application uses 256-bit AES in CTR mode to decrypt the message. This decryption is also done client-side in order to avoid storing the plaintext. The result of this decryption is only displayed in the browser if all resulting characters are ASCII.

$$D_K(\text{ciphertext}) = \text{html\_content}$$

3.4.3 Data Integrity

CryptoBook uses an Encrypt-then-MAC scheme in order to ensure data integrity and authentication. After a user encrypts a post, the HMAC is generated from the ciphertext using the user-provided cipher key. This HMAC is then stored along with the ciphertext in the database.

$$HMAC = HMAC(K_{post}, ciphertext)$$

Before the user decrypts an entry, the HMAC is recalculated from the ciphertext that is retrieved from the server with the user-provided key.

$$HMAC' = HMAC(K', ciphertext')$$

$$HMAC \stackrel{?}{=} HMAC'$$

There are a few scenarios to consider:

- If the user enters the wrong key, the decrypted post will not pass the plaintext check and will not be displayed. The MAC would also not match.
- If an attacker were to replace the post content on the server with content that could be decrypted with the same key, the post will pass the plaintext check but its new MAC will not match the initially generated MAC. In this scenario, the user is notified that the post may have been compromised and that somebody may have tampered with the ciphertext that was stored in the database.

3.5 Shared Posts

Shared Posts are posts that a user is authorized to view but did not write. The Shared Post model is the same as posts, but uses asymmetric key encryption to securely store the posts on the server.

3.5.1 RSA Encryption / Digital Signatures

CryptoBook's sharing feature allows users to share their journal entries with others via public key encryption. Each user has a private and public key pair that are generated as a function of the user's username and password.

$$SK_{Alice} = RSA\_keygen(username||password, 1024)$$

$$PK_{Alice} = RSA\_public\_key(SK_{Alice})$$

The public keys of all users are stored in the database. When a user wants to share a post with another user, he specifies which users he would like to share the post with. The post is then encrypted using each of those users' public keys and signed with the private key of the author (which must be regenerated client-side) and then stored on the server.

In order for a user to view these posts, he/she must type in their password such that their private key can be regenerated and used to decrypt the shared entry. Once the entry is decrypted, the user can validate the post through the digital signature by ensuring that the user and the public signature of the user who signed the entry match. If the digital signature is invalid, the application notifies the user that the entry might be compromised and has not been validated.

4 System Analysis

The following section outlines strengths and weaknesses of the application's security components and system design.

4.1 Security

CryptoBook guarantees confidentiality and authentication for both private and shared posts: symmetric key encryption and HMACs for the former and RSA public key encryption and digital signatures for the latter.

4.1.1 Confidentiality

Assuming AES in CTR mode is secure, private posts written and stored in CryptoBook is confidential. All data is encrypted with a client-side implementation of AES, and thus no plaintext is communicated beyond the client's browser. Even if an adversary were able to guess a user's password, or decrypt it from the database, he could not read the notes. This is because no passwords or post's plaintext are stored on the server. Furthermore, since each post can be encrypted with a different password, the password to one post will not necessarily compromise the user's entire journal.
Shared posts are also kept confidential as long as keys are securely stored by users since the post cannot be decrypted without a user's private key. As discussed in Section 3.5.1, private keys are re-generated each time the keys are needed. The private key is a generated from a passphrase that is a concatenation of the user's username and password. As user's passwords are hashed before being stored in the database and further protected by bcrypt, attackers will have a computationally difficult time guessing users' passwords.

4.1.2 Authenticity

Before decrypting a user's private note, the user must also validate it using a MAC. CryptoBook implements a secure SHA-256 HMAC scheme in order to provide this authentication. Because of the collision resistance property of SHA-256, this provides secure authentication of user posts.
The data integrity of shared notes is guaranteed through digital signatures. The application checks for a valid digital signature, i.e. that the post was digitally signed by the intended author.

4.1.3 Cross-Site Request Forgery

CSRF tokens protect the application users from unauthorized user requests. The CryptoBook protects users from CSRF attacks by requiring a valid CSRF token for each POST, PUT, and DELETE API request. Any API request submitted without a valid CSRF token returns an error. This prevents malicious users from creating posts or sharing posts as another user as they have no method of obtaining a valid CSRF token.[3]

4.1.4 Cross-Site Scripting

CryptoBook also protects users from XSS by sanitizing inputs before sending it to the server. Additionally, MongoDB prevents the possibility of injection attacks by not parsing its input. Since

MongoDB does not parse structured text and stores all inputs as text or numbers, there is no possibility of misinterpreting user input as instructions, and hence no security issues.[4]

4.1.5 SSL Security

While CryptoBook is not currently hosted with SSL certificates, the software enables SSL security. SSL protect network communication from man-in-the-middle attacks where adversaries eavesdrop between the client and server and within MongoDB clusters and replica sets. [4]

4.2 Security Vulnerabilities

One of the main weaknesses of the current version of CryptoBook is that it is still vulnerable to some non-cryptographic attacks such as keylogging, screen capturing, and man-in-the-browser attacks. If an adversary were able to take a screen capture as a user creates a post, the entire post would be compromised, as the adversary would now have the entire plaintext contents. If a keylogger managed to log the encryption key as a user typed it, the adversary would easily be able to decrypt the encrypted post in the future. Both of these types of attacks are outside the scope of the CryptoBook project and should be protected against through anti-virus and anti-logging software on the user's computer.

4.3 Design Weaknesses

CryptoBook is designed such that a user must provide a password for every post. Because it would be insecure to store all these passwords, a user must either remember the password or store it on a device in a secure manner.
Furthermore, like many systems, CryptoBook is susceptible to human error. If users a user-friendly password, they are likely to choose a weak password. CryptoBook does not currently require any level of password strength beyond requiring a non-null password. On the other hand, storing a password on the device causes the security of the users posts to depend on the security of the device. Users can use the same password to encrypt all posts. However, if a user only had to remember one password to decrypt all his posts, an adversary would have much easier time gaining access to all of the user's posts as he will only have to guess one password. When designing CryptoBook, there was a constant tradeoff between convenience and security.

5 Future Work

This section discusses some of the potential opportunities for extensions of CryptoBook both to improve the existing application and to extend to other applications.

5.1 Password Manager

CryptoBook provides a cryptographically secure encryption scheme to store all journal entries with different passwords. Because entries are allowed to have different passwords, it can become hard to keep track of which password corresponds to which journal entry. This inconvenience can be improved through the implementation of an entries password manager. A password manager would allow users to use safer keys without worrying about remembering them in the future.

5.2 Extensions to Current Applications

The proof-of-work presented through CryptoBook can be applied to current blogging and journaling applications by creating extensions or widgets to current websites. As several blogging sites are well-adopted, users may hesitate to switch to a new platform. Furthermore, users may not need to encrypt every post as they may choose for some posts to remain public. A plugin to current blogging sites would allow users to continue using the same framework while providing a high level of security. However, challenges would arise in porting over the sharing component as user accounts would likely need to be restructured to incorporate public key pairs for each user.

6 Conclusion

Overall, CryptoBook achieves its goal of being a web application that allows users to securely and conveniently write and store notes, journal entries, or blog posts online. The interface is intuitive and simple, yet still offers complete functionality. Furthermore, this project is easily expandable and could support many new features.

References

[1] Provos, Niels; Talan Jason Sutton 2012; Mazières, David (1999). "A Future-Adaptable Password Scheme". Proceedings of 1999 USENIX Annual Technical Conference: 81–92. https://www.usenix.org/legacy/events/usenix99/provos/provos_html/node1.html

[2] AES Code aes.js Copyright 2005-2014, MIT License. C. Veness http://www.movable-type.co.uk/scripts/aes.html

[3] Burns, Jesse (2005). "Cross Site Request Forgery: An Introduction To A Common Web Weakness" (PDF). Information Security Partners, LLC. Retrieved 2011-12-12. https://www.isecpartners.com/media/11961/CSRF_Paper.pdf

[4] MongoDB Security Manual – MongoDB Manual 3.0.3. http://docs.mongodb.org/manual/security/