Crypto Google Docs in Google Chrome

CS 6377: Introduction to Cryptography | University of Texas at Dallas | Spring 2012
Mark Adams <mark@markadams.me>, Matthew Pettersson <mcp085000@utdallas.edu>,
Daniel Trimmell <dht100020@utdallas.edu>

Overview

This document describes a Google Chrome browser extension implemented using the Javascript language that allows for encrypted content to be stored and retrieved through Google Docs by using publicly available APIs and Javascript libraries. This project was undertaken in order to fulfill part of the requirements for CS 6377: Introduction to Cryptography at the University of Texas at Dallas taught by Professor Murat Kantarcioglu.

Architecture Description

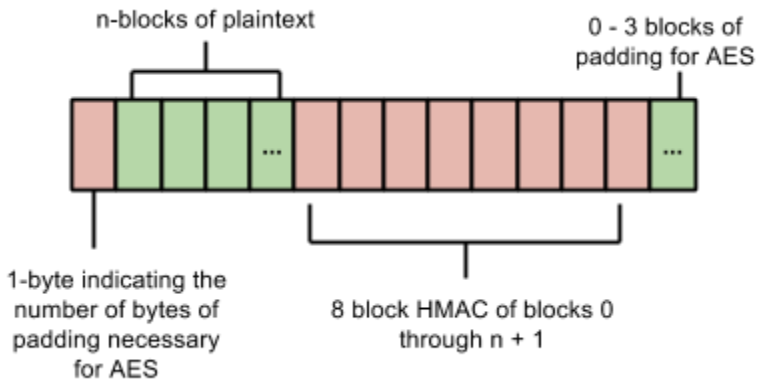
Our team implemented the project as a plugin for the Google Chrome web browser.
To achieve this, we created a custom JSON container and used the Stanford Javascript Crypto Library to implement standard encryption algorithms on the underlying data.

Encrypted Storage Container:

The extension stores the encrypted content and associated metadata in a JSON container containing the following information:

- _t: An identifier indicating that our extension generated the container
- nonce: The starting nonce value to use when decrypting the encrypted file
- alg: An integer indicating which block-cipher algorithm was used
- payload: The encrypted ciphertext

The ciphertext is encrypted using 128-bit AES CTR mode with a SHA256-based HMAC. The ciphertext payload uses the following structure:



If any of the above are modified unintentionally or maliciously a error will be generated on the next attempt to read the crypto-doc. See the usage documentation below for more details on the conditions for the different errors appearing.

If a file is modified in such a way as to invalidate the HMAC, it will make the current version unusable and it will be rejected by the extension. However, the user can restore the document using the Google Docs revision history feature to a time the document was valid, decrypt and continue from there.

Key management

AES keys for each document are kept in local storage in key-value pairs with the docId as the key and the AES encryption key as the value. This value will be referred to as the doc_key.

1. User has a single master password
2. This password has salt added to it and is expanded into an AES sized key (128 bits in our case) using PBKDF2.
    a. The salt is stored in the clear and is used to make brute force attacks on user passwords more difficult, because the attack would have to be a brute force password attack given that particular user's salt value.
    b. The key is generated by providing a hash function that you want to use and the number of rounds to hash
        i. Hashing for several rounds also makes brute force attacks AES keys expanded from passwords more difficult. The legitimate user only has to do this expansion once per password use, where an attacker has to run this expansion for every plaintext string they try.
    c. The resulting key will be referred as the pass_key below
3. In addition to the pass_key another random key will be generated
    a. This key is the key that will encrypt the document with the docId-AES key pairs
    b. This key is secret even to the end user, it will be referred to as the master_key
    c. The master_key is encrypted with the pass_key to protect it when it isn't being

used
    d. Having the master_key be a separate key from the pass_key allows the user to change their password and only have to re-encrypt the system_doc_key instead of the each docId to key listing.
4. To use a specific docId key the following sequence is used
    a. The application prompts the user for a password
    b. User enters password
    c. The application generates the pass_key
    d. The pass_key is used to decrypt the master_key
    e. the master_key decrypts the doc_key
    f. The encrypted document is downloaded using the API
    g. The key to decrypt the document in question

Loading the application

1. Open Google Chrome
2. Click the wrench icon in the upper right area of the window.
3. Select Tools...Extensions from the menu
4. Click the “Load Unpacked Extension” button
5. Browse to the source folder containing the manifest.json and hit “Open”

Usage Documentation

Once the application is loaded the following can be done by the user:

Create a new crypto-doc

1. Create a new document using the standard Google Docs interface
2. Name the document
3. Click the lock icon on the right side of the address bar
4. This launches a window within the tab the user is currently in
    a. From the edit window enter the text desired
    b. When done click the save button
    c. The saving operation automatically closes the current tab once the operation is complete

Decrypt and edit a crypto-doc that the user has previously created

1. Browse to the document you would like to edit in the browser.
2. Click the lock icon on the right side of the address bar
3. This launches a window within the tab the user is currently in
    a. From the edit window enter the text desired
    b. When done click the save button
    c. The saving operation automatically closes the current tab once the operation is complete

Share a crypto-doc with another user

Because our application allows users direct access to the Google Docs file management system the process for sharing a doc is simple

1. Share the document with the user in question using the Google Docs UI
2. In a secure manner give the other user the secret key for encrypting/decrypting the document that is being shared.
3. The other user may then install the extension and follow the instructions above.

Decrypt a crypto-doc that another user has created

Assumes the two steps in the share a crypto-doc case above have already been performed by someone with the secret key

1. Navigate to the document being shared
2. Click the lock icon in the address bar
3. The application will prompt the user for their master password.
4. If verified, the application will retrieve the proper key for decryption. Otherwise, it will prompt the user for the correct key for that document.
5. At this point the steps from the standard decrypt and edit a crypto-doc apply

Error conditions the user may notice

- Not a crypto-doc - An error message will be given to the user if they try to decrypt a non-encrypted doc.
- HMAC failure - The user will receive an error if the HMAC is not valid.

Third-Party Libraries

The creation of this extension would not have been possible (or at least would have been considerably more difficult) without the following third-party libraries:

Used under a 2-clause BSD license.

jWYSIWYG
Copyright (c) 2009 Juan M Martínez
Used under the MIT license

References

The following publications were referenced during the development of this project:

Dworkin, Morris, “Recommendation for Block Cipher Modes of Operation”, NIST Special Publication 800-38A, December 2001,
http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

M. Turan, E. Barker, W. Burr, and L.Chen, “Recommendation for Password-Based Key Derivation”, NIST Special Publication 800-132, December 2010,
http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf

“Google Chrome Extensions Developer Guide”, Google, Inc., April 2012,
http://code.google.com/chrome/extensions/devguide.html

“Google Documents List API version 3.0”, Google, Inc., April 2012,
https://developers.google.com/google-apps/documents-list/

“Specification for the Advanced Encryption Standard (AES)”, FIPS Publication 197, November 2001,
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

“Specifications for the Keyed-Hash Message Authentication Code (HMAC)”, FIPS Publication 198, National Institute of Standards and Technology, March 2002,
http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf

“Specifications for the Secure Hash Standard (SHS)”, FIPS Publication 180-4, National Institute of Standards and Technology, March 2012,
http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf