

CS 255: Intro to Cryptography

Prof. Dan Boneh

Milestone 1: due **Feb. 4 11:59pm**, Milestone 2: due **Feb. 11 11:59pm**

1 Introduction

Have your parents ever been less than thrilled by some inopportune comments you made on your favorite social networking sites while you were perhaps not in the most coherent state? On a different note, are you concerned about the idea that a company could hand over your personal correspondence to the government? Then this assignment is for you. The goal of this project is to build functionality on top of Facebook that allows you to encrypt messages (posts or comments) to members of your Facebook groups. By encrypting messages with your own key, you can be sure that no one with access to your messages – even Facebook itself – can find out their contents.¹

2 Assignment Details

In this project will be writing a *user script* (in Javascript) for that will allow you to encrypt messages when you post in a webpage for a Facebook group. This requires managing encryption keys for the user, and actually encrypting messages using the keys.

But wait, good news! Most of the work to interact with the Facebook site has already been done for you. We're providing you with a single file, `cs255-facebook-encryption.user.js`, that already works as a Chrome user script and handles the interaction with the Facebook site.

If you look through it, you'll see there are five functions that you are responsible for completing:

- `Encrypt(plainText, group)`
 - Return the encryption of the message for the given group, in the form of a string.
- `Decrypt(cipherText, group)`
 - Return the decryption of the message for the given group, in the form of a string.
- `GenerateKey(group)`
 - Generate a new key for the given group.
- `SaveKeys()`
 - Take the current group keys, and save them to disk.

¹Dramatization. As emphasized in throughout the quarter, be very careful about keeping messages secret using a system that you built yourself – especially in Javascript.

- `LoadKeys()`
 - Load the group keys from disk.

These are in the very first section of the code. You'll probably want to add helper functions, but do not modify the purposes of these functions. (i.e. what arguments they take and what they are intended to do)

In order to keep the assignment simple, we will be building encryption only around Facebook *groups*. Groups are comprised of users who can post, and groups can be set to "Open", "Closed", or "Secret". Your script should be able to handle the keys that allow users to post encrypted messages, and handle the actual encryption/decryption. You will have a separate secret key for each group, so that people in one particular group cannot see messages meant for another (even if the group is "Open"). Thus, you can update your family with "studying so hard omg lol omg lol" and your friends with "keg to finish, come now" and no one will be the wiser.

Here is a full list of assignment requirements for Milestone 1:

- Maintain a secure database for the keys (one for each group). This entails:
 - Securely storing each group and its key (using the `SaveKeys` function and any helper functions that you design to write). Note that there is UI framework in the code that eliminates all of the display issues—you only need to focus on the security/cryptography issues. You can find this UI framework in the code and under Settings on Facebook.
 - Securely loading all of these things (using `LoadKeys`) from a data store.
- Ensure that the keys are stored *securely*. Thus you must:
 - Allow the user to use a key database password to securely store/load the keys. The first time a user visits Facebook with the user script, you should ask them to create a key database password. If they already have a database, you should ask them for the password once per session.
 - You should use the password to generate a secure key, by implementing a key derivation function (which should use a salt).
 - The database security can be a little thorny: obviously, if someone has taken over your browser while you're logged-in, then they can get your keys and all hope is lost. Thus, our requirement for the database security is that any attacker who has access to all of your stored material must not be able to learn any significant information about any of your keys. This means that an attacker that sits down at a computer you were using after you have closed the browser cannot get your keys (unless they have your key database password). You'll probably want to create a *session-only cookie* to store the key, which the starter code has helper functions for.
- Provide a function to generate new keys (in `GenerateKey`).
 - Note that information encoded by these keys need to be indistinguishable from random. Note that calling a function in the javascript `Math` library is *not* indistinguishable from random. However, Chrome provides cryptographically secure random numbers using `window.crypto.getRandomValues`. The starter code contains a convenience function for this.
- Build encryption and decryption functions that provide CPA security for Facebook group messages (`Encrypt` and `Decrypt`).

- This is mostly up to you. Apply the concepts from class to create a secure encryption scheme. You must build this on top of the AES implementation provided in the script. (Do not attempt to implement RSA or Diffie-Hellman or some other protocol—it will probably not be a secure implementation and will take you much longer than doing it this way.)

For milestone 2 we add message integrity. The requirements are:

- Build a MAC system based on the AES implementation given to you.
- Use your MAC system to authenticate group messages.
- Use your MAC system to make sure keys in the key store haven't been changed between program runs.

3 Background Material

We will use several useful tools to build our encrypted message system, so you should familiarize yourself with these before you get started. We list these here:

3.1 Web

- *Chrome*: The user script provided should be installed on Google's Chrome browser. We highly recommend that you use the latest non-beta version of Chrome to develop and test the assignment.
 - Download Chrome: <https://www.google.com/chrome>
 - In order to install the file as a user script, open the Extensions page <chrome://extensions/> and drag-and-drop the file, and check "Enable" for it. Any time you want to test and updated version of the file, repeat this process.
- *Facebook accounts*: The script works when you are logged into a Facebook account. It is possible to write and test the user script using your own Facebook account, but we will be providing you with test accounts.
 - Note: All CS255 test accounts can interact with each other, but they can't interact with regular users. You can use a test user's "email" to find and "friend" another test user.
 - In order to use the user script, you will need to create a group to post in. Go to <https://www.facebook.com/groups> to create a group. Note that you need to add at least one friend to create a new group.
 - The extension will provide "Encrypt" buttons for text boxes on group pages and a tool at <https://www.facebook.com/settings> for managing your keys.
- *Javascript*: We will be programming exclusively in Javascript for this assignment
 - A great reference for learning Javascript: <http://www.codecademy.com/tracks/javascript>

3.2 Crypto

- *AES*: A large chunk of the assignment will involve building secure primitives with the Advanced Encryption Standard (AES). While no one knows exactly why AES is difficult to break, you should be somewhat familiar with how it works.
 - AES wiki: http://en.wikipedia.org/wiki/Advanced_Encryption_Standard
- *Pseudorandom Number Generator*: You will need to use a PRG. Understanding how these work will make this portion of the assignment much easier for you to understand.
 - Random number generation wiki: http://en.wikipedia.org/wiki/Random_number_generation
 - Pseudorandom number generator wiki:
http://en.wikipedia.org/wiki/Pseudorandom_number_generator
- *Message Authentication Code (MAC)*: Used to provide message integrity, these are just short bit strings used to verify that a message actually came from the purported sender. These will be covered in class in the near future, but for now it may be helpful to understand what they do.
 - MAC wiki: http://en.wikipedia.org/wiki/Message_authentication_code
- *Chosen plaintext attack security*: This is one model of security that cryptographers use to show properties of various cryptographic schemes, and which we expect your implementations to have. It will be covered in class.
 - Ciphertext indistinguishability wiki: <http://en.wikipedia.org/wiki/IND-CPA>

3.3 Potentially Useful References and Tools

These are *not* needed to complete the assignment, but you may find them useful.

- *WebKit Inspector and Chrome Debugger*: Although you can't interact with variables in your user script from the Javascript console, Chrome provides a lot of useful tools for debugging your script.
- *Chrome Cookies and Site Data*: You can view and clear any cookies and local storage data you've created under "Content Settings" in Chrome's advanced settings.
 - Direct URL: `chrome://settings/cookies#cookies`
- *[GreaseMonkey]*: GreaseMonkey allows users to write scripts that change HTML content on-the-fly. Chrome's support for user scripts is based on GreaseMonkey (which came first), so you may also find it useful to look up information about GreaseMonkey if you can't tell what's going on.
 - GreaseMonkey site: <http://www.greasespot.net/>
 - GreaseMonkey wiki: <http://wiki.greasespot.net/>
- *[PhantomJS]*: It's actually possible to run the user script without Chrome, using a tool called PhantomJS (node.js won't work, because of all the browser code). You should focus on making sure the code works in the browser, but you're welcome to use phantom.js if you find it useful for testing.
 - PhantomJS site: <http://phantomjs.org/>
 - How to run: `phantomjs --cookies-file cookies.txt cs255-facebook-encryption.user.js`

4 Deliverables

- Code, in a file named `cs255-Lastname1-Lastname2.user.js`, with your group members' last names properly substituted in the filename. You MUST also update the headers of that file, as specified in the comments.
- A file named `README.txt` containing the full names of the people in your project group, and a description of anything the course staff needs to know to be able to run your project. Include any special steps such as how to create keys, enter database passwords, etc.
- For each milestone you will need to submit a write-up describing your design choices. *At minimum, your writeup should discuss all the design decisions you made for the bullet points from Section ?? and argue why the implementation of each bullet is secure.* While we do not expect formal, rigorous proofs, we do expect a proof-like explanation of why your scheme is secure under the guidelines given by the assignment. A good write-up will include a detailed conceptual description of all encryption, decryption, storage, and generation operations and an argument explaining how an attacker that can break some part of your scheme can also break some underlying primitive that is believed to be secure.

Your write-up should also discuss the following questions:

1. What are some of the biggest issues with doing cryptography in the browser? Why might we want to do it anyhow?
2. How could somebody go about circumventing the security of your implementation, if they really wanted to? (e.g. side-channel attacks)

The writeup must be in PDF, plain text, or Markdown format, named `cs255-Lastname1-Lastname2.pdf` (or `.txt` or `.md`).

5 Grading

Your system will be graded on two bases:

- Does it work?
 - We will check to make sure that your system works. All messages entered should be correctly displayed and interpreted, the key store should work, and the script should not become unresponsive or crash. We may also run automated tests (which is why you shouldn't change the given function signatures).
 - Your security will not matter for this part of the grade. However, if your encryption/decryption methods fail and cause bad things to happen that are visible to the user, then you will lose points.
- Is it secure?
 - Your write-up should detail a fully secure protocol and clearly explain why your scheme is secure.
 - We will go through your code and check for security issues. Your code should contain clear comments explaining what it is doing to provide proper security.
 - You should be able to mitigate all possible attacks on your scheme with what has been covered in class (or will be covered soon).

6 Submitting the Assignment

At press time, we have not yet set up the submission script. Please check the discussion forums for details on how to submit.

7 Frequently Asked Questions

- *What functions are we allowed to write/edit?*

You should edit the given functions at the top of the file. Your solution should be well-decomposed, so you should also add appropriate new functions at the top of the file that make it easy for us to tell how your code works. You also *may* edit code below, but please document anything that might result in behavior we don't expect.

- *What characters do we need to be able to encrypt?*

Although Facebook supports the full Unicode character set, we only require you to be able to handle the standard ASCII character set. Unicode support is not required, although you are free to implement it.

- *May I use the same key for different purposes?*

No, re-using a key is insecure. If you need to use the same key for two separate purposes, you should use it to derive two independent keys. (Remember that separate parts of the output of a stream cipher are independent, as discussed in class.)

- *I end up getting unprintable characters when I encrypt messages. What do I do?*

You may be getting unprintable characters because the AES function returns bytes that can take any of the 256 ASCII values, even those that cannot be correctly displayed. To avoid this, and ensure that copying encrypted messages will not remove information, look into hex/base64 encoding of your displayed ciphertexts. You must ensure that any encoding mechanism you use is properly attributed and decodes correctly.

Look lower in the file for `sjcl.codec.utf8String` and `sjcl.codec.base64`.

- *Can I just make the user directly enter a 128-bit long key as their key database password?*

Although usability is not the main aim of this project, it is unreasonable to expect the user to remember 128-bit keys, and no assumptions can be made about the quality of the keys.

- *How do I ask the user for the key database password?*

You are free to create your own UI for prompting for the key database password. However, a simple UI you can use is the JavaScript `prompt()` function.

- *Do we need to support the user changing the password to their key database?*

No, you may assume that the user will never change the password protecting his or her database once the password is set.

- *I have to enter my key database password every time I visit a Facebook page. Is there a way to get around this?*

Yes, and we expect you to implement a solution that only asks the user for the database password once per session. You should store a cookie with the relevant information when the user logs into the site and enters their key database password for the first time. You can find more information about this at http://www.w3schools.com/js/js_cookies.asp. Just make sure you are meeting the security requirements when you use the cookie. (Hint: You will want to store some information in *session-only cookies*.)

- *I generated a key for my group. How are we expected to share it with members in the group?*

Secure key exchange is not the focus of this assignment. It's reasonable to assume there is some secure channel you can share the key. For the purpose of this assignment, it is okay for you to tell your friend the generated key and for him/her to just enter the key into the database.