# A message Encryption System For Social Networking Clients

Anshika Agarwal Modhi Alsobeihy

[aagarw6@uic.edu](mailto:aagarw6@uic.edu) [malsob2@uic.edu](mailto:malsob2@uic.edu)

# **Abstract**

In this report, we present our implementation of a messages encryption system that makes it possible to preserve the privacy of user’s messages and assures that only intended audience are capable of reading these messages. This system is build on top of social networking service as a browser extension and encrypts messages so they are in clear text to specific user groups [1]. We tested our implementation using Facebook [2] and Mozilla Firefox[[1]](#footnote-1) [3].

***Keywords:*** Encryption, Cryptography, JavaScript.

# Introduction

## Overview

In the following sections, we describe our implementation of a message encryption system that is build on top of Facebook as a Firefox extension and targets Facebook groups. Facebook groups are set of users who have the same interests. Facebook groups have three privacy options: Open, Closed, or Secret. Unlike “Secret” and “Closed” options, in a group that is set to “Open” all messages are public.

This implementation has two major goals. The first goal is to allow users to post encrypted messages such that only members of their group can read them- if the group is open. The second goal is to authenticate messages and ensure the integrity of keys between sessions.

## Background and Motivation

By subscribing to social networking services, users release their personal information to the world and have no control over who can access them. Although most of these services provide some privacy controls through their user interfaces, users’ privacy still not preserved. On one hand, the service providers may access important and sensitive messages; and on the other hand, sensitive messages may be read by other users who are not supposed to read them because of the limited functionality of privacy controls.

The main objective of this work is to provide users with a tool that makes it possible for them to get the control of the privacy of their messages by limiting the readability of messages to only intended audience. By utilizing cryptography techniques on the client side, we ensure that messages leave the client end in a scribbled format. This way only intended users on the other end who have the secret key may decipher them and view the messages in a readable format; not to mention, the messages are saved in servers in the same format they left the client.

# Implementation Tools

## JavaScript

JavaScript is a dynamic computer programming language. It is most commonly used as part of web browsers, whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed [5].

JavaScript is a prototype-based scripting language with dynamic typing and has first-class functions. Its syntax was influenced by C. JavaScript copies many names and naming conventions from Java, but the two languages are otherwise unrelated and have very different semantics. The key design principles within JavaScript are taken from the Self and Scheme programming languages. It is a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles

## GreaseMonkey

Greasemonkey is a Mozilla Firefox extension that allows users to install scripts that make on-the-fly changes to web page content after or before the page is loaded in the browser. The changes made to the web pages are executed every time the page is viewed, making them effectively permanent for the user running the script. Greasemonkey can be used for customizing page appearance, adding new functions to web pages (for example, embedding price comparisons within shopping sites), fixing rendering bugs, combining data from multiple web pages, and numerous other purposes [4].

## Mozilla Firefox

Mozilla Firefox (known simply as Firefox) is a free and open-source. web browser developed for Windows, OS X, and Linux, with a mobile version for Android, by the Mozilla Foundation and its subsidiary, the Mozilla Corporation. Firefox uses the Gecko layout engine to render web pages, which implements current and anticipated web standards [3].

## Facebook Accounts

Facebook is an online social networking service. Users must register before using the site, after which they may create a personal profile, add other users as friends, exchange messages, and receive automatic notifications when they update their profile. Additionally, users may join common-interest user groups, organized by workplace, school or college, or other characteristics, and categorize their friends into lists such as "People From Work" or "Close Friends" [2].

# Project Components

Two major components are defined for this work, each of which reflects one of the project main goals. The first component, an encryption system which ensures that one group cannot see messages meant for another group; and the second one is an authentication scheme that assures the integrity of the encryption system. In the following sections we describe how each component works a long with the design tools used to implement it.

## Building an Encryption System

To realize this phase, we have to fulfill the following requirements:

### Build a secure database of the group/key pairs.

Design choices: JSON String; CryptoJS.AES library [8];

For each group involved in the encryption system, we need to store its key securely. We also need a persistent storage on the client side so users do not have to enter the keys of their groups every time a session is run. For this, we used cookies. The group/key database is available as long as the cookies associated with the browser are not deleted. To securely store the database, we stringified the group/key pair using **JSON.stringify()** method. AES encryption [6] is then called using the method call **CryptoJS.AES.encrypt()** to encrypt the result with a database key that previously entered.

At the beginning of each Facebook session, the user is prompted to enter the database key. If this is the first time, the key is stored in a cookie and in the browser session storage as well. If the database key is already set and the key entered matches the one in the system, then the database is retrieved and decrypted using the method call **CryptoJS.AES.decrypt()**. The group/key table is found under “Settings”. The functions responsible for these tasks are

* ***SaveKeys():*** this function is called every time group/key pairs is updated. It encrypts the database and stores them in the cookie.
* ***LoadKeys():*** this function is called at the start of each Facebook session. It retrieves the group/key database from the cookie and decrypts it.

### Provide a function to generate keys.

Design Choices: Random number generator;

When a user creates a group, they can add a key to this group manually or they can let the system generate a key for this group. We implemented a random key generator which uses the **window.crypto.getRandomValues()** method call to generate and assign a key to the new added group. The function responsible for this task is:

* ***GenerateKey():*** this function is called when the user needs the system to generate a new key for the given group.

### Develop encryption and decryption functions.

Design Choices: CryptoJS.AES library;

For the encryption and decryption functions we are using the CryptoJS.AES library. A user type a message and then click the encrypt button. The massage is retrieved from the interface in its readable form, encrypted by calling **CryptoJS.AES.encrypt()** and then displayed in an unreadable format. The user then posts the encrypted message. When posted, the encrypted message is retrieved from the interface, decrypted using **CryptoJS.AES.decrypt()** and displayed in a readable format if the user has the decryption key, otherwise, the message is displayed as it has been posted. The functions responsible for these tasks are

* ***Encrypt():*** this function gets the message in plaintext and returns it encrypted.
* ***Decrypt():*** this function gets the message in cipher text and returns it decrypted.

## Building an Authentication Scheme

For this phase we fulfilled the following requirements.

### Building a MAC system based on AES.

Design Choices: CryptoJS.SHA3 library [8];

For building the MAC system we used CryptoJS.SHA3 library. SHA-3 uses the sponge construction in which message blocks are XORed into the initial bits of the state, which is then invertibly permuted [7]. When a new key or message is generated, a hash of the original message or key is computed (the hash is computed using the key or message in plaintext form). The hash is computed using **CryptoJS.SHA3()** where the plaintext is passed as an argument. The method returns the hash of the plaintext, which is then appended to the original plaintext. Then the resulting string is encrypted using AES encryption standard as defined in the previous section.

### Using the MAC system to authenticate messages.

We use the MAC system to authenticate the messages. The message in the plaintext is used to compute the hash. The hash code of the message is then appended to the message plaintext. The concatenated string is then fed to the encryption algorithm to get the encrypted message. This encrypted message is then posted on the wall or as comments. To decrypt the message, the encrypted message is read from the interface, which is then fed to the decryption algorithm as defined in the earlier sections. From the decrypted message, original plain text and the hash code are extracted. Then to check the authenticity of the message, the hash code is calculated from the plaintext. The new hash code and the old hash code are compared to check for any changes.

### Using the MAC system to ensure the integrity of keys between runs.

We use the MAC system to ensure the integrity of the keys in the same way we authenticate messages. The plaintext keys are used to calculate the hash code. The hash code is then appended to the plain text keys and encrypted using the AES encryption scheme. The encrypted key is then stored using the cookies. The browser fetches the encrypted keys from the cookies each time the user logs back in to facebook. The encrypted text is decrypted using AES and the plain text key and the hash code is extracted. A new hash code is computed for the plain text key and compared with the old hash code. The keys are said to be valid and allowed to decrypt the messages if the two hash codes are same.

# Operational Requirements:

1. Download Mozilla Firefox: www.mozilla.org/**Firefox**‎
2. Download GreaseMonkey: http://www.greasespot.net/
3. Download the latest version from <https://github.com/Modhi-A/facebookProject/>
4. Create an account on Facebook and create some groups

# Screenshots

|  |
| --- |
| **Creating database key** |
| **Accessing group/key table** |
| **Macintosh HD:Users:giric:Desktop:Screen Shot 2014-05-06 at 3.07.38 PM.png**  **Generating a key for a given group** |
|  |
| Macintosh HD:Users:giric:Desktop:Screen Shot 2014-05-06 at 3.18.24 PM.png **The Facebook message encryption system** |
|  |
|  |

# Design Issues

## One platform

This message encryption system is specifically implemented on the client side. The system is supposed to work as a Firefox browser extension on computers. The system is working properly as long as the environment is as described. The extension script has to be changed to fit other browser interfaces. If the user needs to change to another computer then the group/key database must be entered again or the cookie s has to be transferred to the browser on the other platform.

## Database key leak

Since all of the system files are saved as cookies, if an attacker was able to brute-force database key or read the cookies and got the database key, all the message encryption system would be compromised. Not to mention that the platform is already vulnerable to XSS and CSRF attacks.

# Work Distribution

We divided the workload around the project goals. Modhi Alsobeihy was responsible for achieving the first goal, while Anshika Agarwal was responsible for achieving the second goal. The members were communicating and discussing any issues that came up while developing the system. Thankfully, the team spirit was always present and the work was smooth.

#### Acknowledgment

As mentioned before, this work is based on a project assignment given in a cryptography course in Stanford. Because the user interface is the same, we used the parts of the starter script that handle the communication between HTML and JavaScript with some modifications to fit Mozilla Firefox context.

# References

[1] Encrypted Facebook Groups, Online [Website: http://crypto.stanford.edu/~dabo/courses/cs255\_winter12/hw\_and\_proj/cs255-p1-handout.pdf]

[2] Facebook, Online [Website: http://en.wikipedia.org/wiki/Facebook]

[3] Firefox, Online [Website: http://en.wikipedia.org/wiki/Firefox]

[4] GreeseMonkey, Online [Website: https://addons.mozilla.org/en-US/firefox/addon/greasemonkey/]

[5] Javascript, Online [Website: http://en.wikipedia.org/wiki/JavaScript]

[6] Miller, F. P., Vandome, A. F., McBrewster, J., 2009. Advanced Encryption Standard.

[7] M. Bellare, R. Canetti, H. Krawczyk. Keying hash functions for message authentication.

[8] CryptoJs library, Online [Website: https://code.google.com/p/crypto-js/]

1. This work is based on a project assignment given in a cryptography course in Stanford. The handout available at: http://crypto.stanford.edu/~dabo/courses/cs255\_winter13/hw\_and\_proj/cs255-p1-handout.pdf  
    [↑](#footnote-ref-1)