**Team: Hung Tran, Megan Kanne**

1. **Maintain a secure database for the keys (one for each group):**

I use function sjcl.misc.pbkdf2 to convert the group name into a key, using a salt. The salt is cryptographically-secure and uniquely-generated-amongst-different-users 128 bit salt by using function GetRandomValue(). The group name is unique so different groups will have unique keys. The keys are stored securely by function SaveKey() in which I use CBC with random IV for encrypt the key database. The key to encrypt the database is the key generated by user's password to key database. This is CPA secure because, the IV is truly random generated by GetRandomValue (Assume that the GetRandomValue() is cryptographically-secure then an attaker cannot predict the IV). Secondly, I use function sjcl.cipher.aes() which is a secure PRP. Therefore, according to CBC theorem, if sjcl.cipher.aes() is a secure PRP then the CBC with random IV is semantically secure under CPA.

Loadkeys is secure because to get the keys database out decrypted, the user must have the correct generated key from the password. If the user enter a wrong password or the key stored in sessionStorage is incorrect, the key database will not be decrypted or decrypted incorrectly. This is ensured by the decrypted CBC algorithm.

1. **Ensure that the keys are stored securely**

I allow user to use a key database password to securely store/load the keys. I save the encrypted key database in localStorage. I don't store any decrypted keys in local or session storage. I should ask first time user to create a key database password. If they already have a database, I ask them for the password once per session.

I use the generated key from user password to encrypt and decrypt database key. The generated key is created using random 128 bit salt from cryptographically secure GetRandomValue.

I use sessionStorage to store the generated key. Therefore, after I close my browser properly, the generatedKey will be deleted from sessionStorage, An attacker sits down on my computer after that will not be able to get the generated key and thus, can't decrypt the key database and can't decrypt the message.

1. **Provide a function to generate new keys (in GenerateKey).**

As discussed in part 1, I use function sjcl.misc.pbkdf2 to convert the group name into a key, using a salt. The salt is cryptographically-secure and uniquely-generated-amongst-different-users 128 bit salt by using function GetRandomValue(). This ensures that the group key is indistinguishable from random (I didn't use Math.random)

1. **Build encryption and decryption functions that provide CPA security for Facebook group messages**

All my encryption and decryption in this assignment (key database encrypt/decrypt, message encrypt/decrypt) use CBC with random IV. The IV is cryptographically random generated by using GetRandomValue(). Attacker cannot predict this IV (assume the function is implemented correctly). I use padding to ensure the aes block will be multiple of 16 bytes. So according to CBC theorem, as long as sjcl.cipher.aes() is a secure PRP then my CBC algorith m with random IV is semantically secure under CPA. No matter what the message the attacker sends to the encryptor, he will get back a random encrypted message since IV is generated randomly. Even the same plaintext submitted twice will generate different random ciphertext. Therefore, CPA attack does not work (given that the (pL)^2/|X| is negligible)

1. **What are some of the biggest issues with doing cryptography in the browser? Why might we want to do it anyhow?**

The biggest issue here is that if I don't close the browser properly after each session or I leave my session on, an attacker can sit in my computer and steal my generated key from sessionStorage. With that, the attacker can encrypt and decrypt every messages. Another danger is when people in the same group share group key with each other without a secure channel. There is no guarantee that an attacker cannot get his hand on the key during the key transfer process. We want to do it because it is still more secure than storing plaintext, limiting the number of people who can see the text. And as long the users follow the proper protocol, the chance of breaking the cryptography is low.

1. **How could somebody go about circumventing the security of your implementation, if they really wanted to? (e.g. side-channel attacks)**

As discussed in the previous answer, the attacker can:

+ Use man-in-middle attack during the key transferring process to get the group key

+ Try to get my generated key from sessionStorage if my browsing session does not close properly

+ Side-channel attack (timing attack, power monitoring attack, acoustic cryptanalysis, etc..)

+ Social engineering to get the users' key/password