Programming Assignment 1 Winter 2011

CS 255: Intro to Cryptography

Prof. Dan Boneh Milestone 1: due **Jan. 28 11:59pm**, Milestone 2: due **Feb. 6 11:59pm**

1 Introduction

Have you ever been bothered by your grandmother attempting to friend you on facebook? Have your parents 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? Then this assignment is for you. The goal of the assignment is to build functionality on top of Twitter that allows you to encrypt tweets to subgroups of your Twitter followers—just think, no more mom and dad reading the "suepr wstaed" messages you left for all of the world to see.

2 Background Material

Rather than completely rebuild Twitter, we will use several useful tools to build our encrypted tweet system, so you should familiarize yourself with these before you get started. We list these here:

- *Firefox*: We will be using several add-ons that only work for Firefox, so you'll have to use it even if you prefer another browser. Get the latest non-beta version (3.6), since some of those add-ons are not yet compatible with Firefox 4.0 beta.
  - Download Firefox: `http://www.getfirefox.com/`
- *Twitter*: This should be obvious unless you've been living under a rock for the past few years. However, you'll want at least two throwaway Twitter accounts so that you can test all of the encryption/decryption algorithms you'll be running for the assignment.
  - `http://www.twitter.com`
- *Javascript*: We will be programming exclusively in Javascript for this assignment
  - A great reference for learning javascript: `http://www.w3schools.com/js/default.asp`
- *GreaseMonkey*: GreaseMonkey is a Firefox add-on that allows users to write scripts that change HTML content on-the-fly. This assignment will consist of a single GreaseMonkey script (most of which has already been done for you), so you will need to download GreaseMonkey and obtain some minimal understanding of how it works.
  - GreaseMonkey site: `http://www.greasespot.net/`
  - GreaseMonkey download: `https://addons.mozilla.org/en-US/firefox/addon/748`
  - GreaseMonkey wiki (that is actually the easiest way to learn GreaseMonkey): `http://wiki.greasespot.net/`

- *AES*: A large chunk of the assignment will involve building secure primitives with the Advanced Encryption Standard (AES). You can read more about it at:
  - AES wiki: `http://en.wikipedia.org/wiki/Advanced_Encryption_Standard`
- *Pseudorandom Number Generator*: Spoiler alert: at some point in the assignment, you will need to use a PRG as discussed in class. A high level description is available at:
  - Pseudorandom number generator wiki: `http://en.wikipedia.org/wiki/Pseudorandom_number_generator`
- *Message authentication code*: 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*: A security model of symmetric encrytion discussed in class and in the textbook. A high level description is available at:
  - Cipertext indistinguishability wiki: `http://en.wikipedia.org/wiki/IND-CPA`

Well, we're finally done with background material. If you don't understand something, it is probably coming up soon in lecture, but feel free to ask anyways.

3 Assignment Details

Let's get to the nuts and bolts of the assignment. As we have already established, you will be building an encrypted Twitter system. But wait, good news! Most of this has already been done for you. If you haven't already, you'll want to download the GreaseMonkey script *cs255.user.js* from the course website and peruse it. If you look through it, you'll see there are five functions that you are responsible for completing: `Encrypt`, `Decrypt`, `GenerateKey`, `SaveKeys`, and `LoadKeys`. These are in the very first section of the code and it should be clear to see what to do with each one.

We will be doing more than basic encryption and decryption for one Twitter account, though. In addition to this basic functionality, you will need the notion of groups. Have you ever wished you could have separate facebook accounts for your friends and family? We will create just that sort of notion in Twitter here. You will be able to create multiple groups, and assign each person following your tweets to a group. You will have a separate secret key for each group, so that people in one particular group cannot see tweets meant for another. 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.

Hopefully it should be intuitive at this point what needs to be done. However, we will now specifically spell out the requirements. We are dividing up the assignment into two milestones. The specific requirements for milestone one are as follows:

- Maintain a secure database of other people you are following on Twitter that are using your encrypted key system. This entails:

  - Securely storing each user, their key, and their group assignment for you (using the `SaveKeys` function and any helper functions that you deign 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 Twitter.
  - Securely loading all of these things (using `LoadKeys` and whatever helper functions you choose) from a data store.
  - The database security can be a little thorny: obviously, if someone has taken over your browser, then they can get your keys and you are hosed. 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 (which is awesome if you have to share a computer).

- Maintain a secure database of your groups and their respective keys. Thus you must:
  - Securely store/load each of these with the other sensitive data mentioned above
- Provide a function to generate keys for the user
  - Note that these keys need to be indistinguishable from random. Remember, calling a function in the javascript math library is not indistinguishable from random.
- Build encryption and decryption functions that provide CPA security for tweets.
  - This should be straightforward enough, but note that you are required to build on top of the AES protocol 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.
  - Why CPA security? If someone can predict or influence your tweets, then this is a nice feature to have.

For milestone 2 we add message integrity (to be covered in the lecture). The requirements are:

- Build a MAC system based on the AES implementation given to you.
  - Note that you are **NOT** allowed to use the SHA-256 implementation lurking in the bottom of the script. The point of the assignment is to understand how to build primitives.
- Use your MAC system to authenticate tweets.
- Use your MAC system to make sure keys in the key store haven't been changed between program runs.

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` 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. Inclued 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 3 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.

  The writeup must be in PDF or TXT format, named `cs255-Lastname1-Lastname2.pdf` or `.txt`.

5 Grading

Your system will be graded on two bases:

- Does it work?
  - We will check to make sure that your system works. All tweets entered should be correctly displayed and interpreted, the key store should work, and the script should not become unresponsive or crash.
  - 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. Thus, it would be extremely helpful (and possibly beneficial to your grade) if you clearly comment your steps to ensure 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 Piazzza discussion group for details on how to submit.

7 Frequently Asked Questions

- *Twitter's website has 2 user interfaces, the old Twitter and the new Twitter. Do we need to support both?*

We tried to make the code compatible with both user interfaces. However, we will only test your projects using the new Twitter UI.

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

  Although Twitter 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.

- *I can't support encrypting a full 140 character tweet! How long of a message must our project support?*

  At minimum, you should be able to encrypt messages that are 30 characters long. With ingenuity, it is possible to encrypt messages much longer than that. The Javascript `parseInt()` and `number.toString()` functions may help. Also look into base64 encoding. You may use any encoding snippets from the net for converting binary to text and vice versa (properly attributed to the source), since implementing that is not part of the assignment.

- *Can I just make the user directly enter the 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. See `http://www.w3schools.com/JS/js_popup.asp` for more information.

- *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.