COMP 448: Computer Security **Released: Nov. 28, 2023**
Final Project **Due: Dec. 15, 2023, 11:55 PM**

Group Registration

This is a **group assignment** (groups of 3, with one group of 2). Please register your groups in the "Final Project" sheet of this shared spreadsheet:

https://grovecitycollege-my.sharepoint.com/:x:/g/personal/johnsonej_gcc_edu/EReSLag5dQBIkOgQTZOeZoMBZA1hR8wKA9eqKScgVrpwdw?e=E2BU4h

(This is the same spreadsheet we've used for previous group signups, but a different sheet (tab).)

Assignment

In Project 3 (Encrypted Messaging with RSA and AES), you used Python's `cryptography` library to build a simple encrypted messaging app, similar to the well-known PGP ("Pretty Good Privacy") system, which uses *hybrid encryption* — a combination of asymmetric (RSA) and symmetric (AES) encryption algorithms — to exchange keys and messages confidentially over an insecure channel. However, for that project, you only had to worry about protecting *confidentiality*, one of the three major security properties we've studied in this class. You will now build on that to protect *integrity* as well, as part of an app of your own design that provides end-to-end secure communication across an insecure channel.

Specifically, you will need to learn about *message authentication codes* (MACs) and *authenticated encryption*, two alternative techniques for ensuring the integrity (authentication and tamper-resistance) of messages sent across an insecure channel. You will need to devise a design that securely integrates these with the tools you've already learned for exchanging and using secret keys for confidentiality purposes.

Instructions

For this project, your task is to **design and develop an end-to-end secure communication app that allows users to communicate securely (confidentiality + integrity) across an insecure network.** Your design must allow communication between different computers on the same network (or, optionally, across the Internet), and must do so in at least a semi-automated fashion (i.e., it cannot rely on users manually copying/pasting and sending encrypted text and keys via email or chat apps, the way PGP does).

It is up to you to decide how your app should work and how to implement it. It is recommended (but not required) that you use your Python-based encrypted messaging code from Project 3 as a starting point; however, you are encouraged to make your own choices as to what messaging features, modes, and user interfaces your app will support.

To make your job easier (since you only have ~2.5 weeks to complete the project), I've created a simple "pastebin"-type website that will be hosted on my instructor VM for the duration of the class. As described in detail in the "Pastebin Site API" section below, this provides both a human-oriented HTML front-end, as well as an HTTP/JSON back-end API that can be accessed programmatically from other software. **You are encouraged (but not required) to utilize this pastebin site as "cloud storage" for your app. However, you need to be aware that it is shared by the whole class and is completely insecure — meaning other class members can not only read, but potentially fake or tamper with the messages you store on the site. Your design also needs to be tolerant of the "noise" of other groups' posts on the shared system. Therefore, it will be essential that you utilize strong cryptography to defend against such threats!**

(You are *not* required to guarantee *availability*, the third of the three primary security properties, for this project. That would be much more difficult to ensure without running your own trusted cloud back-end, and in general is much harder to comprehensively guarantee.)

Due to the limited time you'll have to build this, I would suggest sticking with a simple but meaningful set of features. Some examples of suitably complex designs include:

- An app that lets users post messages and public keys to "the cloud", and browse and download the messages and keys posted by other users in order to exchange messages with them.

- A real-time (or near-real-time) "chat app" that lets users exchange secure instant messages in "live" conversations. You might or might not find the provided pastebin site useful as a cloud back-end for this.

- A peer-to-peer file sharing app that provides end-to-end encryption. *(If you use the pastebin site as a cloud backend for this — which is recommended — I would ask that you limit the files you exchange to **10 kilobytes or less**. Your app is welcome to support much bigger files, but dumping huge pastes on the server could cause issues for the class, since I haven't put a ton of work into making the server super robust.)*

Pastebin Site API

The pastebin site is currently hosted at **http://cs448lnx101.gcc.edu/**. Please note that this address may change if technical difficulties are encountered and I need to make changes to the hosting setup; I'll announce any changes to this on Teams.

You should familiarize yourself first with how the pastebin site works by trying it out manually in a web browser. If you've ever used other popular pastebin-type websites (e.g. for sharing code), such as the "original" https://pastebin.com/ — which has spawned countless imitations and alternatives over

the years **(note: do NOT use this or any other public site for your project unless you are confident you are operating within their terms of service!)** — this will be quite familiar. Our own pastebin is much simpler and lacks "fancy" features like syntax highlighting, user registration, etc., but it has all the essentials you'll need for this project.

In addition to the browser-oriented HTML interface, our site has a programmer-friendly HTTP API that allows you to communicate directly with the backend using JSON. The endpoints available to you are as follows:

- **Create Post:** `/posts/create`

  Accepts HTTP POST requests. Expects the contents of a new post to be provided in the **`contents`** form field.

  Returns a JSON object with the following fields:
  - **`error`** (boolean): true if an error occurred, false otherwise.
  - **`error-reason`** (string): If an error occurred, gives a human-readable explanation.
  - **`id`** (int): The ID of the newly-created post, if it was successfully created.

- **View Post:** `/posts/view/<int:id>`

  Accepts HTTP GET requests. The parameter **`id`** in the URL should give the numeric ID of the post you wish to download. IDs start at 1 (for the first post on the site) and increment automatically. IDs used by since-deleted posts are never reused.

  Returns a JSON object with the following fields:
  - **`error`** (boolean): true if an error occurred, false otherwise.
  - **`error-reason`** (string): If an error occurred, gives a human-readable explanation.
  - **`contents`** (string): The text of the requested post, if it was successfully found.

- **Get Range of Posts:** `/posts/get/<int:from_id>/<int:to_id>` or `/posts/get/latest`

  Accepts HTTP GET requests. The parameters **`from_id`** and **`to_id`** should give a range of post IDs that you wish to download. This will return 0 or more results (keep in mind that there may be "holes" due to deleted posts).

  The range **`to_id - from_id`** is restricted to a maximum of 1000. Exceeding this will result in an error being returned.

  Returns a JSON object with the following fields:
  - **`error`** (boolean): true if an error occurred, false otherwise.

  - **`error-reason`** (string): If an error occurred, gives a human-readable explanation.
  - **`posts`** (array): The posts (zero or more) that were found within the requested ID range. Will only be present in the response if no error occurred. Each element in the array is itself a JSON object (dict) with the following fields:
    - **`id`** (int): The ID of the post.
    - **`when_created`** (string): An ISO-standard timestamp indicating when the post was created.
    - **`contents`** (str): The text of the post.

- **Delete Post:** `/posts/delete/<int:id>`

  Accepts HTTP POST requests. Anyone is allowed to delete any post on the system without authentication, but as a friendly safeguard against accidentally deleting posts, you must provide the SHA-1 hash of the post's contents in the **`hash`** form field. The post will only be deleted if the hash matches the one computed by the server.
  - *Please note: despite SHA-1 being a notionally "cryptographically strong" hash algorithm, this is <u>not</u> intended here as a security feature — merely a "speed bump" acting as a proverbial "are you sure button" (to the extent such a concept translates into a programmatic API). As you may have read in the* `cryptography` *library's documentation, SHA-1 is outdated and no longer considered to provide strong security, as it has gotten weaker in light of increasing computing power and cryptanalytic advances over the last couple decades. It does, however, serve just fine for this non-security purpose.*

  Returns a JSON object with the following fields:
  - **`error`** (boolean): true if an error occurred, false otherwise.
  - **`error-reason`** (string): If an error occurred, gives a human-readable explanation.

<u>Rules for Using the Pastebin Site (PLEASE READ):</u>

For the project to go smoothly for everyone, I ask that you all be at least a *little* considerate in how you use the pastebin site. 🙂 While a certain amount of tomfoolery and what might be considered "abuse" by a typical public pastebin site (e.g., using it as a "poor man's cloud storage" for your own app) is expressly allowed and encouraged here in the lab environment, I have not had the time to *really* harden the site's security, nor is it running on a particularly beefy server (it's running in a Docker container on a VM identical to your own class VMs).

So, please respect the following "ground rules":

- Try not to delete other people's posts. I've included a "delete" function so that you can clean up your own messes, but if you go around nuking other people's posts, it's going to make it hard for them to complete their projects. That said, I realize it can be easy to get posts mixed up, especially if they're all encrypted gobbledygook. For the sake of your own sanity when

writing your app, you will almost certainly want to include some sort of unique "tag" in your app's messages that distinguish them from other groups'.

- I haven't implemented a hard limit on how long a post can be (perhaps I should have, but I didn't have a lot of time to hack this together over break 🙂); but please refrain from posting messages larger than **10 kilobytes** (10,240 bytes), so as to not overload the server (or your fellow students' bandwidth when they're loading the web page in their browsers or downloading batches of posts). Keep in mind that anything you post will likely be downloaded *many* times by other groups, as they have to blindly search around for their own posts!

- Please don't spam the server *too* hard with lots of posts. Again, this is mainly about not breaking the server, and not "burying" other groups' messages under a pile too big for them to reasonably search through. It's fine to send messages as often as might be needed for, say, an "instant messaging" chat app, but please don't programmatically generate piles of posts faster than anyone could reasonably type or read them.

- Likewise, please don't hit the server multiple times a second with requests. Again, there are no hard rate limits implemented, but if your app is hitting the server more than once or twice a second, that's probably unnecessary for the purposes of this project. If you want/need to exchange "live" data at a faster rate than this allows, you should have your app communicate over network sockets directly (or create and host your own server backend with a more suitable design) rather than using the pastebin.

- Please be aware that anything you post on the pastebin is public to the whole class (except to the extent your encryption is effective, but that's your responsibility, not ours 😉) — and that I can make no guarantees of how long (or not) the data will be retained after the class is over. I also cannot guarantee that the server won't fall over tomorrow and lose all the posts. 🙃

  Fellow students are **welcome and (if they feel so inclined) encouraged to try to break the encryption on your posts**, so you should understand that anything you post on the server does not enjoy any implicit legal expectation of "expectation of privacy", including with respect to legal issues surrounding the cracking or stealing/trafficking of passwords. This is a lab environment expressly intended to give students opportunities to try advanced security techniques that might be challenging to attempt safely or legally in the "real world".

  I will likely continue to run this site next semester for Advanced Security, during which several of these rules (especially regarding deleting others' posts) may be loosened or removed. I may retain some or all of the posts from this semester in the database.

  (That said, if you do your encryption right, it "should" be well-nigh impossible for anyone to break in the foreseeable future; so I would not recommend that any of you waste much time trying to break each other's encryption, unless you spot a really sloppy mistake that would make it computationally tractable. 🙂)

Written Report and Submission

Upload the following to D2L **in a single .zip file**:

- All of your source code, along with instructions for how to run and use your app. If your app isn't something I could easily set up and run on my own, please speak to me in office hours so we can figure out how best to grade your work.

- A written report **(as a PDF file)** describing your design and the security-relevant choices you made in it. In particular, you should **discuss which encryption primitives/libraries you used, and why,** to address the particular threats you could foresee to the confidentiality and integrity of your app's communications. **State your security goals and assumptions (your "threat model") clearly and explain how you addressed the relevant threats.** If you intentionally chose *not* to defend against certain foreseeable threat scenarios, explain why you feel these were reasonable trade-offs to make against other desirable factors (user experience and usability, implementation complexity/difficulty, computational or cost limitations, etc.). You might, for instance, choose to partially mitigate certain threats in a way that reduces the likelihood or severity of their impact, despite not providing 100% protection. If you have to leave any threats unmitigated for lack of time, you should explain how you *would* have defended better against them if you had more time.

Final Project Demos and Grading Rubric

Pending administration approval, I am planning to dedicate the class's final exam period **(6:00-8:00 PM on Saturday, December 16)** to final project demos and presentations. **You should plan on giving a short (5-10 minute) presentation, as a group, explaining your design and demonstrating your app in action.**

I will grade your project as follows (out of 100 points):

- **(40 points)** Your design is sound, well-explained in your written report, and effectively prevents or mitigates all identified threats that you defined as in-scope in your threat model. Any trade-offs you have chosen to make in incompletely/partially mitigating particular threats are reasonable and well-explained, in light of the time and resources available to you, and the user-experience goals of your app.
- **(10 points)** Your threat model (as described in your written report) is well-reasoned and makes reasonable assumptions relative to how you expect your app to be used.
- **(40 points)** Your app "works as advertised", and (in particular) implements all security controls described in your design without obvious flaws or mistakes.
- **(10 points)** Your final presentation is well-articulated and effectively explains your design choices and trade-offs to the audience.