Assignment #3 Private albums

Opened: Monday, 17 March 2025, 12:00 AM
Due: Monday, 31 March 2025, 11:59 PM

In this assignment, you will design and implement a private album feature. You are provided a system that meets the correctness properties below, but one that is very insecure and provides no confidentiality.

Acknowledgment: The assignment was adapted from the project designed by MIT 6.1600

Code Overview

In this assignment, there are new encryption methods in the `crypto` module for both public-key and symmetric encryption and each client has an associated public key and authenticated encryption object that can be used to encrypt and decrypt.

As in assignment 2, you can use an out-of-band method to share *signing* public keys---the same `add_friend(friend_username, friend_signing_pk)` method--to implement the following two scenarios:

1. To create an album, all users that the album will be shared with must be added as friends.
2. To fetch an album, it will be required that the owner of the album be added as a friend.

Public profiles

Upon registration, the client can upload its own encryption public key into the public profile. The public profile now also includes a metadata field where you can add extra information.

Also note that the setup for this assignment is *mostly* separate from the log-based setup for the previous assignments. Instead of adding photos to the log, we will now be updating an "album" object on the server via new RPC types defined in `common/types.py: UploadAlbumRequest` and `GetAlbumRequest`.

Debugging

The Autotrader for this assignment uses another alternative that allows it to find all data that a client has access to. For this reason, your encryption printouts may look a bit different than you are expecting. However, they include data that may be helpful in debugging. For more information, see the `ag/ag3/crypto_overrides.py` file.

Specification

Scenario

Suppose that Alice has taken some photos which she wishes to share *privately* with some of her friends. Right now, all the photos she uploads through our application are public.

In order to allow Alice to control who has access to these pictures, you need to implement the notion of *private albums*. A private album has an *owner* (`"Alice"`), a list of *photos* (`[secret_party_invite.jpg, buried_treasure_coordinates.jpg]`), and a list of *friends* who can add photos to and view photos in the album (`["Alice", "Bob", "Carlos", "Danielle"]`). In this scenario, once Alice has created the album, she wishes for the following to be true.

1. All of the friends should be able to view all of the photos, and
2. No one else should be able to view any of the photos.

For this assignment, unlike in Assignment 1 and Assignment 2, each user will have a single device.

Assumptions

For this assignment, you can make the following assumptions:

?

- The server is functioning correctly and will not tamper with the integrity of messages. However, the server might not authenticate users before performing operations on their behalf.

- Operations between the clients and the server happen sequentially and atomically. A client will always have the latest version of the server state when the client tries to modify the state.

Correctness

Given these assumptions, the album feature should support the following **correctness** properties.

1. Any client can create a private album and upload it to the server using `create_shared_album(album_name, photos, friends)`, where all friends in `friends` have been added using `add_friend`. The client who does this is the album's *owner*.
2. Any client who is part of an album's `friends` can view photos within the album by calling `get_album(album_name)`, provided that the album owner's signing key has been added locally using `add_friend`.
3. Any client who is part of an album's `friends` can add photos to the album by calling `add_photo_to_album(album_name, photo)`.
4. An album's owner can `add_friend_to_album(album_name, friend_username)`, provided that `friend_username` is the username of a user who has been added using `add_friend`. Once a friend is added to an album, it should be able to access all photos that were already added.

Security

In addition, your implementation should also support the following **confidentiality** property: clients (identified by their public signing key) who are *not* part of an album should not be able to see any photo within the album.

1. If a client is never a member of an album, they should not be able to see any photos in the album.
2. An album's owner can remove a member from an album with `remove_friend_from_album(album_name, friend_name)`. After a member is removed, it should not be able to see any new photos that are added to the album.

In your implementation, you can use the PKI, the public profile feature, and the two new cryptographic primitives in the crypto library: **authenticated symmetric-key encryption** and **authenticated public-key encryption** (which simultaneously signs and encrypts)**.**

Liveness

Finally, your implementation should support some property of liveness. That is, your system should still work even in the face of (certain types of) attackers. With this in mind, your protocol should support the following property:

- The presence of illegitimate members in an album should not prevent legitimate members from using the album as usual. Any 'members' that were not added by someone besides should be ignored, but must not be allowed to view the album contents.

Code submission details (see report submission details below)

You need to modify `client.py` and `log_entry.py` to implement *secure* private albums.

Important note:

- Do not change the signatures of the public methods of Client (i.e., the *__init__* method or methods not beginning with an underscore _).
- Do NOT modify any other files given to you

To check your implementation, use `**make grade-lab3**`

Upload the following **client.py** and **log_entry.py** with the following header:

```
'''
Full Name:
Course ID:
Description: a very brief description of what is implemented (no more than 4-5 lines)
'''
```

- You must include enough high-level comments in your program so that a reader (e.g., the grader/TA) who is an expert and knowledgeable can understand the basic operation of your program.

- Your code must be well commented and in neat order. Specifically, please refer to Google Python Style Guide for sections on indentation, comments, naming, etc.

- **Note we will deduct points for excessive commented lines of debug code that would make reading your code difficult to read.**

This is an open-ended assignment!

As with the previous assignment, there are many different ways to solve this problem. You need to submit your own solution. It may take a few iterations to converge to a good design. Start design work early!

Assignment Report (submission link)

You are to write a comprehensive document outlining the implementation and design of your protocol. Your report should include the following:

Executive summary section

- Executive summary summarizing the main points of the report and the performed work. This should also include both Part 1 and Part 2.

Part 1: Implementation report

- *Protocol description* section. A section with a detailed description of the implemented protocol: what was implemented, how, and why, with clear references to your implementation and design choices.
- Describe and explain how your protocol satisfies the two confidentiality goals and the liveness property described above.
  - Ensure you describe and address each security goals in separate titled sections with clear references to your implementation and protocol design

Part 2: System security questions

Question 1: Consider a system where you suspect a man-in-the-middle attack during communication with a server. Suppose you possess a key, K, and the server has a public key, PK, and a private key, SK. Describe how you can confirm sharing K with the server or detect a man-in-the-middle without being discovered by a packet sniffer.

Question 2: What kind of attack could a malicious user, Eve, execute on the protocol you described in part 1 of the assignment, if we replace the Public Key Authenticated Encryption primitive with a non-authenticated Public Key Encryption primitive, potentially allowing her to access photos not intended for her?

Submit the report separately here

- **assignment3_report.md** (in Markdown syntax) (will also be checked for plagiarism)
  - The report must be in an *.md* file in a markdown format.

(**Note: TA are instructed to deduct points for submissions using non-Markdown (.md) formats - ie. don't submit docx, pdf, or other formats)**

```
---
title:  EECS 4481M Assignment 3 Report
author: Author Name
...
```

All submissions will go through a plagiarism detection check. Students who are found to have engaged in academic dishonesty will be referred to the Dean's office for disciplinary action which can result, at minimum, in receiving a FAIL grade in the course, as well as suspension or expulsion from the University.

Assignment3.zip 14 March 2025, 9:55 AM

Add submission

Submission status

Submission status	No submissions have been made yet
Grading status	Not marked
Time remaining	6 hours 14 mins remaining
Last modified	-
Submission comments	▸ Comments (0)