650.445/600.454: Practical Cryptographic Systems February 20, 2024

Assignment 2

Instructor: Matthew Green *Due: 11:59pm, March 11*

Name: _____

The assignment should be completed individually. You are permitted to use the Internet and any printed references.

Please submit the completed assignment via Gradescope.

Problem 1: Implementing a Messaging Client (50 points).

You have been asked to implement a client for the JMessage encrypted instant messaging service. JMessage is a client-server architecture that allows individual clients to communicate using end-to-end encryption. The detailed operation of JMessage is described in a specification document.

To assist you in your implementation, we have provided you with two components:

1. A complete specification for the JMessage system
2. A working JMessage server, implemented in Python using the Flask toolkit

We will also provide you with remote access to a client that you can use to test compatibility. Note that your implementation must be cross-compatible with the reference server and client. *In the (un)likely event that the specification proves inconsistent with the reference client/server, you should prioritize compatibility with the reference implementation.*

You may find the server and an *unfinished* skeleton of the Golang client at the following location, along with instructions for running the server:

`https://github.com/matthewdgreen/jmessage_2024`

Submission and grading: As a deliverable, you must provide the source code for your client. Your code can be delivered in a single file, or in multiple files contained within a proper go module. It should support the command-line interface options implemented in the partial Go implementation, as follows:

```
-attachdir string
   attachments directory (default "./JMESSAGE_DOWNLOADS")
-domain string
   domain name for the server (default "localhost")
-notls
   use HTTP instead of HTTPS
-password string
   login password (default "abc")
```

```
-port int
   port for the server (default 8080)
-headless
   run in headless mode
-reg
   register a new username and password
-stricttls
   don't accept self-signed certificates from the server (default accepts them)
-username string
   login username (default "alice")
```

Most importantly, your client *must* support the `-headless` option: this will run the client in a loop, checking and printing any received messages every 100ms without further user interaction.

Note: Particularly nice submissions will be considered for extra credit. This includes things like mobile clients, GUIs and additional features. Even if you choose to implement these extensions, you must turn in a base client that supports the command-line interface described above.

Problem 2: Attacking a Messaging Client (50 points).

There is a flaw in the JMessage cryptographic implementation that may allow you to violate the confidentiality of other users. Assume that you have intercepted an encrypted message sent by user Charlie to user Alice. Unfortunately you do not have Alice's secret decryption key and she will not tell you the message's contents. But Alice is online and continuously checking her messages, and thus attempt to decrypt any messages sent to her.

Your task is to implement an automatic attack program that can decrypt a message sent from another user to Alice. We will assume that Alice is simply another copy of your client, running with the `-headless` option. You should assume that you can intercept an encrypted message sent by Charlie to Alice, and write it into a file as a JSON `message` object.[1] Your program should run the following steps:

1. Load a target ciphertext sent to Alice from a file.
2. It should then repeatedly:
   (a) Modify the ciphertext in some way.
   (b) Send the modified ciphertext to Alice for decryption.
   (c) Obtain any responses sent by Alice, and then repeat from step 2a.
3. At the conclusion of this process, your code should output the plaintext of the target ciphertext.

[1]NB: The structure of a JSON `message` object is provided in the specification. For testing your code, you should obtain this JSON yourself by running a client as "Alice" and having a second client run as "Charlie". Now either modify Charlie to print out the ciphertext it is sending to the server (easy), or try to intercept it from the server by guessing Alice's password.

For simplicity, you should include your attack code directly into your Go client implementation from Problem 1. (This means you will turn in a single program for both Problem 1 and Problem 2!) Your automated attack should execute from the command line with the following command-line option:

```
jmessage-client -attack <ciphertext filename> -victim <username>
```