CS255 Programming Project: The Authenticator

The project focuses on Anonymous Authentication. Our objective is to enable a person to authenticate himself to a service without revealing his identity. For example, the Wall Street Journal (www.wsj.com) requires its customers to purchase a subscription to gain full access to the site. Subscriptions are most frequently managed by assigning a username and password to all subscribers. Unfortunately, by doing so the Wall Street Journal can keep track of its customers browsing habits. Privacy advocates may be happy to learn that a much better solution exists.

Our goal is to enable a subscriber to prove possession of a valid subscription without revealing the subscriber's identity. We call the subscription service (e.g. WSJ) the service. The simplest solution is to assign the same username and password to all subscribers. Clearly when a subscriber logs in, the service has no clue as to which of its subscribers it is serving. Unfortunately, this solution is unworkable for two reasons: (1) when a subscriber cancels his subscription all other subscribers have to be assigned new usernames and passwords, (2) if a subscriber misbehaves (e.g. in a chat room) there is no escrow agent that can undo the subscriber's anonymity.

In this project we (partially) solve the problem of anonymous authentication with identity escrow. In other words, subscribers can anonymously login to the service, but if they misbehave then an escrow agent can expose their identity. You will be building three components:

Client: enables subscribers to anonymously login to a subscription service.

Service: verifies that a subscriber has a valid subscription. The service records its interaction with the subscriber. The transcript is sent to the escrow agent in case the subscriber misbehaves and its identity must be exposed. *Without the escrow's help the service has no information as to who it is serving.*

Escrow: contacted by the service to expose the identity of a misbehaving subscriber. To simplify things the escrow agent is also the entity assigning authentication keys to subscribers. This need not be the case, but it does simplify things for this project. To further simplify things we assume that the escrow and the service are running on the same machine.

For this assignment, our service will be a simple chat room, but it is important to keep in mind that the same escrow can be used by many different services.

The system's life cycle is as follows:

1. **Initialize**: The escrow agent generates a public key to be used by a service.
2. **Subscribe**: A subscriber obtains an authentication key from the escrow agent.
3. **Authenticate**: A subscriber authenticates itself to the service.
4. **Escrow**: The service asks the escrow agent to expose the identity of a subscriber.

Next, we describe in detail the interaction between the three components in each of these steps.

Notation:

- **s** is the maximum number of subscribers. Say **s = 1,000,003** (a prime).
- **N = p · q** is the modulus we work with. We require that **s** divides both **p-1** and **q-1**, but s^2 does not divide either **p-1** or **q-1**.
- **R** is an s^{th} root of unity modulo **N**, i.e. $R^s \equiv 1 \pmod{N}$. Furthermore, **R** is not equal to 1 modulo **p** or **q**.

Initialize:

- The escrow agent will be running as a daemon. When it is contacted to generate a public key for a particular service it does the following steps:
- First, the escrow agent generates a 1024 bit modulus **N** satisfying the properties described above.
- Next, it generates an s^{th} root of unity **R** as described above.
- Finally, it picks a random **t** in Z_N and keeps t secret. It computes $T = t^s \pmod{N}$.
- The service stores **N, T** and **s** in a public file called **<service>.pub** where <service> is the name of the service that will be using the public file. It is up to you to decide on the format of the file. You may also wish to include other fields in the file, such as an expiration date, etc. The escrow signs the contents of **<service>.pub** to ensure its authenticity.
- **R, p** and **t** comprise a secret key that only the escrow agent should know. **(R,p,t)** should be stored encrypted on disk in a file called **<service>.priv**. The encryption key should be derived from a password entered by the person activating the escrow agent.

Subscribe:

- When a user connects to the escrow agent to request an authentication key he supplies his name and the name of the service he is subscribing to. The escrow responds as follows:
- Let **i** be the number of authentication keys issued by the escrow agent for this service so far. The escrow agent sends back $\mathbf{K_i = t \cdot R^i \pmod N}$. This last message must be sent over a secure connection, i.e. encrypted with a session key between the subscriber and the escrow.
- The escrow agent records on disk the association between **i** and the subscriber's name. All associations should be stored in a single file called **<service>.subscribers**.
- The key $\mathbf{K_i}$ will be used by the user in future authentications. To keep $\mathbf{K_i}$ secret the user stores it on disk encrypted with the user's password. A **MAC** should be used to ensure the file is not tampered with.

Authenticate:

- Now comes the magic part. Using $\mathbf{K_i}$ a user can authenticate to the service that it is a legitimate subscriber without revealing its identity.
- First the service loads the information stored in the file **<service>.pub** and verifies the escrow's signature on the file. The protocol then proceeds as follows:
- The user picks a random **u, v** in $\mathbf{Z_N}$ and computes
  $\mathbf{x = (u)^{*2} \pmod N}$
  $\mathbf{y = (v)^{*2} \pmod N}$
  $\mathbf{z = K_i \cdot u^* \pmod N}$
  and sends **(x,y,z)** to the service.
- The service checks that $\mathbf{z^* = T \cdot x \pmod N}$. If not, the service rejects the user.
- Next, the service picks a random **c** in $\mathbf{Z_N}$ and sends it to the user.
- The user computes $\mathbf{w = u^c \cdot v \pmod N}$ and sends **w** to the service.
- The service checks that $\mathbf{(w)^{*2} = x^c \cdot y \pmod N}$. If not, the service rejects the user. If so, the user accepts the user as an authentic subscriber.
- The service records the value **z** for future reference.

Escrow:

- Suppose one of the subscribers misbehaves and it becomes necessary for the service to determine the identity of the subscriber. The service gives the value **z** stored from the interaction with the subscriber to the escrow agent. The escrow agent should output the name of the subscriber with whom the interaction took place.