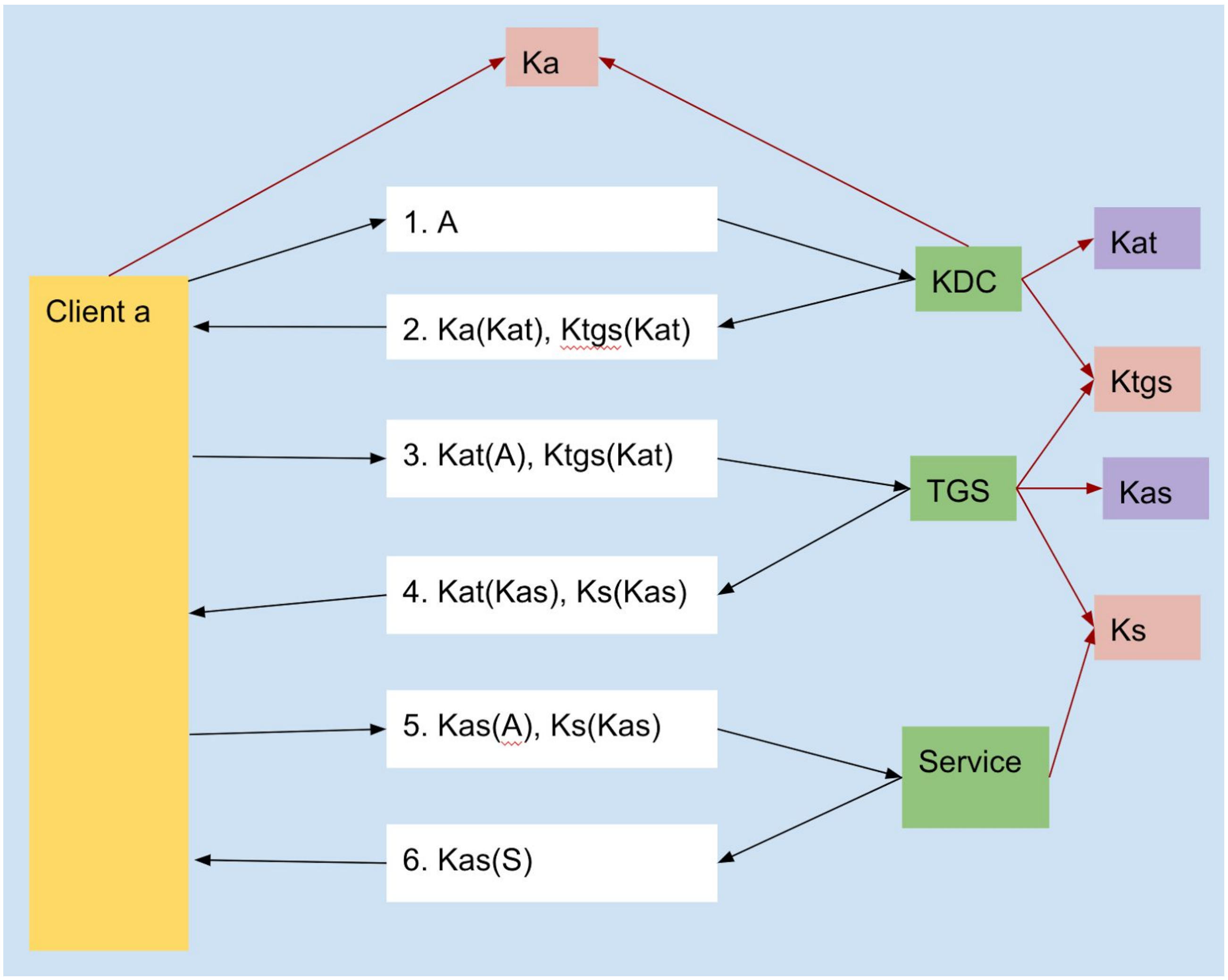
Zach Dixon and Adam DeHovitz
CS195y Final Project

Overview

For our final project we sought to model the Kerberos authentication system using alloy. Our original model came from a combination of this article as well as slide 15 of cs1380's security lecture. We then simplified the model, removing timestamps, until we came to the following model that we'd attempt in alloy. White boxes represent messages and responses. Pink boxes represent information that entities have at the start. Purple boxes are keys that would be generated during Kerberos authentication but entities start with them here since we are only modelling one authentication.



Here is a summary of what occurs at each message in our simplified model:

1. Client identifies itself to the KDC (Key Distribution Center)
2. KDC sends back two versions of a session key (Kat) for you to use to communicate with TGS (Ticket Granting service). The first is encrypted with Ka which the client can decrypt (In the real world this is decrypted using a hash of the password). The second is encrypted with the TGS secret key so that when the client makes contact to the TGS both it and the TGS will be able to use Kat.
3. The Client sends its identification encrypted with the session key (Kat) to the TGS, and it sends the session key encrypted with Ktgs so that TGS will have Kat.

4. TGS replies by sending Kas, the session key to be used between the client and the service, encrypted in a way that the client can read it and in another way that the service can read.
5. The client sends its identity to the service, encrypted with the session key (Kas), and sends the session key (Kas) encrypted with the service secret key to the service.
6. The service then sends back its identity to the client encrypted with the shared session key originally distributed by TGS. At this point the client and the service can freely and safely communicate.

Properties

Our goal was not to directly model this system, but instead to show that this system results from a simple set of properties of the system. These properties are:

1. The servers in the system are stateless. That is, they can only operate on data that they are sent during a request and on data that they inherently have access to. They cannot learn information for later use. The only exception for this is that the service can learn Kas, which reflects the real-world scheme where Kas is used for subsequent encrypted communication between the user and the service.
2. Requests to the server are discrete. No request interrupts the process of another.
3. The client never learns secret keys (Ktgs or Ks).
4. In order to reply to the client, a server must know that it is communicating with the client (It must know A).
5. No data should be sent unencrypted, except the initial client message of A, which must be sent without encryption because, in reality, the KDC wouldn't know which Ka to use until it receives the identity of the client (A).
6. The user should only have to use their password (which Ka is based on) once.

Challenges

Our major challenge is that our model is complicated, which makes solving it in Alloy slow. A large number of steps are required (encryption, decryption, sending), and in the end the solver was working with around 200,000 variables. Finding a solution takes at least 15 minutes to an hour. We worked to simplify the model, with a major simplification being that we removed decryption, and assumed it would happen if it was possible. As a consequence of this, there exists some refactoring we wanted to do but opted not to given how long it takes to verify our model.

A secondary challenge was in reasonably restricting the model without overtly specifying the real-world model. For example, a requirement had to be included which didn't allow communication between servers, and another which required that the servers would be stateless (except the service). Without these, Alloy would come up with clever solutions which couldn't be translated to reality.

Result

A valid, resulting configuration is included in the included workingModel.xml file, which can be opened in the visualizer. This is included because the solver takes a long time to produce a solution (around 15-30 minutes). The configuration is almost identical to the diagram above, except the final response (redundantly) includes Ks(Kas) and some operations are shuffled around. The KDC starts off knowing what it needs to in order to encrypt Ka(Kat) and Ktgs(Kat), so it could do that before receiving the first Message. While this isn't directly correct, it effectively makes no difference, so we left it as is.