security design

# Client

Made up of three main parts:

1. Allegiance.exe (Alleg.exe)
   * Check for valid connection with CSS Server
2. A launcher exe (Launch.exe)
   * Validate Alleg.exe (and any other files needed)
   * Grab hardware information (same as ASGS now)
   * Download & run the "blackbox"
   * Act as a debugger for Alleg.exe/Handle error reporting
3. A small bit of executable code, to be downloaded upon each login attempt. ("Blackbox"/"BB")
   * Validate Launch.exe (see below)
   * Take the user credentials and hardware information and encrypt them
     + The key used is the part that will change upon each download, this is the part that ensures the correct "blackbox" was used
   * Send the encrypted data to the authentication server

Alleg.exe will remain very similar to its current form. The user will start with Launch.exe, like ASGS. Launch.exe will show a window for the user to enter their credentials. When the user tries to login, Launch.exe will initiate contact with the server. The server will reply with the Blackbox. Launch.exe will load the Blackbox assembly. Upon loading, the Blackbox will get the checksum of the program that launched it (Launch.exe). This checksum will be used to decrypt a string stored inside of the Blackbox. The value being decrypted is the key that Launch.exe will use to communicate with the server. If Launch.exe differs from the version that the server expects, the decryption will not happen properly. An exception will be thrown... causing the user to be rejected.

Launch.exe will be given a short time to respond back with the correct encryption key. This short period will hopefully prevent amateur hackers from breaking in. I know that an "unhackable" system is impossible... I'm just shooting for something that keeps YP & crew out.

With this system, Alleg.exe is authenticated by Launch.exe. Launch.exe is authenticated by the Blackbox, which is on the user's system for a very short time.

# Server

In regard to security, other concerns will come up later):  
The server will be responsible for managing communications with the clients. It will need to track which users are using which keys to encrypt. The server will also be responsible for creating these keys as needed. The original plan was to create a new key for every login attempt, this might not be needed though. The standing idea is to use a pool of keys, to limit the number of keys that needs to be created. With this pool in place, two main rules will be enforced:

1. A user will never see the same key twice
2. A user will not be able to predict what key they will be assigned next

Multiple users will be allowed to use the same key, but no user will reuse a key. If a user has used every active key, the server will create a new one to be added to the pool. Upon login, the server will assign a random (new) key from the pool. Any attempts to login with a key that is in the pool but not assigned to the user (i.e. guessing keys) will be flagged for an admin's attention. Measures for key timeout will be added.

### Advantages:

* Small amounts of modification needed to Alleg.exe
* Different encryption keys used for every login attempt (no single encryption key like ASGS)
* Blackbox can be (more or less) changed at will, since it is on the server's side
* Proper validation of Alleg.exe

### Disadvantages:

* Somewhat complex implementation

Feature lisT

# Launch

## Features required

* Handle login, logout
* Reasonably give out progress information (at least on log file)
* Give meaningful error messages (“error 0xF7546543”)

## Features required without loose Allegiance Client integration

* Streamlined user interface
* Remember username and password
* Autoupdating system (untied from file protection), possibly to replace Alleg’s ingame AU system

## Wishlist

* Account creation
* Password recovery
* Roster management
* Diagnostic system
* Link to wiki

# Blackbox

## Features required

* Validate Launch
  + Ability to robustly validate in as many different ways as possible
  + Actual validation performed is only one chosen randomly
* Fast execution
* Fast, cheap production
* Obfuscation

## Wishlist

* Blackbox should not hit the disk
* Blackbox should be sent encrypted
* Blackbox should be obfuscated

# CSS Server

## Features required

* Account storing
* Account linking
* Backend for account creation, etc.
* Blackbox creation
* Blackbox delivery (as random as possible)
* Stats tracking
* Squad system
* Web interfaces
* Backup system
* Allegiance Lobby integration

## Wishlist

* Blackboxes should be pre-compiled and added to a reasonably-sized ‘cache’ during low activity times

# Also required

* Admin tools
* Enforcer tool
* Zone tool
* Community polling