# Kryptose: Security Design

**Personnel**: Jonathan Shi (js2845), Antonio Marcedone (am2623), Alexander Guziel (asg252), Jeff Tian (yt336)

**Confidentiality/Integrity**

For each user, the client creates a data structure (an ArrayList of Credentials) that is used to store the individual credentials that the user wishes to store within our system. The data structure is then serialized, encrypted on the client side and then sent to the server for storage.

The encryption is done using the AES/GCM/NoPadding ciphersuite. The key is 128 bits long, and at the moment is set to be 0 and was decided based on NIST guide to key length. In future releases, the key will be derived from the master password the user chooses. The initialization vector for the encryption is randomly generated by the client using a SecureRandom implementation.
Therefore a malicious server could only re-play to the client an older version of the credential list, but not tamper it in any way. The GCM mode is an authenticated encryption mode of operation, and ensures that the server is not able to modify the stored credentials, as well as any metadata that are stored with them.

The integrity and confidentiality of the connection between the client and the server are guaranteed by the use of SSL connections. In particular, we only accept the latest TLS1.2 version of the protocol, with either the TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 or the TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 ciphersuites. Diffie Hellman Key Exchange (or its elliptic curve version) provide perfect forward secrecy, and a strong block cipher and hash function (with an authenticated encryption mode of operation) guarantee a strong level of security. No vulnerabilities on this version of TLS and ciphersuites are known at the time of writing.

The client authenticates the server by using digital certificates. We created our own certification authority and installed its certificate on the client. We provided the server with a signed certificate. The client DOES NOT perform hostname validation at the moment, meaning that any certificate signed by our CA will be accepted as valid by the client, without consideration for the common name on the certificate.
This is not a concern, as we assume that our CA will only issue certificates to our own server for now. We decided for this approach (as opposed to installing a simple self signed certificate on the client) to allow the use of short validity certificates and thus limit

the window of time for a MITM attack in case of server compromise. This will also make the switch to certificates signed by a real certification authority easier.
We plan to add hostname validation and possibly certificate revocation lists in future releases.

In the beta, authentication of a user to the server will be done using a secret derived from the same master password that is used to derive the encryption key for the credentials. We will make sure that knowledge of this authentication secret (which will be stored hashed on the server) does not help in the decryption of the credentials (even though it is derived from the same master password) by using the master password as a seed for a key derivation function (possibly chained with a pseudorandom generator), and then we will use the first 128 bits of the output for the encryption key and the second 128 bits as the authentication key. The specific algorithms have yet to be determined.

**Audit**

We logged every connection and user request on the server side, and these logs are viewable only by the server administrator (at the moment). There are two types of logs: general server logs and user-specific logs.

The server logs contain all of the major actions and exceptions raised and caught by the server, along with a timestamp for each. This includes when the server boots up, what port it listens on, which SSL connection requests it receives and accepts, and the exceptions it throws internally and handles. This log is stored in XML format and generated the java.util.logging library in Java.

On the user-specific client logs, the server logs any operation that the client requests, and whether the it is fulfilled or fails. For example, a successful put request where the client stores a new version of the password file, and a successful get request when a client retrieves their password file, are both logged. Each log entry contains the date and time the request was received, the username of the user who initiated the action, the type of request, and whether or not the request was successful. If the request was not successful, the log also contains the reason for the request failure, for example a get request could fail because of invalid client credentials, and a put request could fail because of a stale write issue. These logs are displayed in a user-friendly format, and the intention is that by the Beta release the user should be able to request to see a copy of their own log to audit their account.