Armadill

# Overview

Armadill is a simple instant messaging application. It can send text messages and files. Users can register their usernames, that can’t be used without their password. The messages are encrypted and none, not even the server can read them. Messages are sent directly, not through the server.

# Security design

Text messages are sent in one piece, files are split into segments. Each text and segment is encrypted, header and MAC are added to form a single message. Armadill uses central server for user registration and authentication. Server creates temporary certificate for each login session. Users use desktop clients to connect to server for authentication and send messages to each other directly.

Server’s public key is stored in application source code and therefore distributed with the application. Armadill therefore doesn’t rely on any public key infrastructure.

### Client to server authentication

Client will first establish TLS connection to the server. It will then authenticate the user using his username and password. The client will then generate its RSA key pair and send its public key to the server. The server will add username to the public key a sign it creating certificate for the client, which will be valid for 24 hours. The server will send the certificate back to the client.

### Registration

Client will establish TLS connection with server, then enter his username and password. If username exists, it is rejected by the server, if not, user is registered and first session certificate is created. User password hash is stored in server database with username and online status. Password hash will be created using pbkdf2-SHA512 and a random 32 byte random salt.

### Client to client authentication

Clients establishing connection will first send its signed certificates to each other. They will then verify them. If both are valid, they will establish TLS connection and use Diffie-Hellman key exchange to create encryption and MAC keys. The exchange messages will be appended with usernames of both clients and timestamp, then signed by the client’s private key. All further messages will be encrypted with AES-256 and authenticated with HMAC-SHA512 using the newly generated shared secret.

### Client to client communication

Each message will be composed of:

* Both client usernames
* Timestamp
* Key number used
* Message type
* Message payload
* Message authentication code

Both the current and one previous key will be accepted.

Message types:

* Text
* Text with DH
* File
* File with DH
* DH

First message after establishing new shared key will contain DH. If client both sent and received DH, it will create and start using new shared key. If the client receives 10 messages, but did not send any, it will send new DH regardless in a stand-alone message. Client will not send or accept more than 10 messages signed and encrypted by the same shared key.

### Server logout

Client will send logout message using the previously open TLS connection. User will be also logged out, if the TLS connection is severed.

# API design

## Client

Client is divided into 5 classes. User interface, cryptographic class, server connection class, peer connection class and message class.

Cryptographic module is providing functions for easy creation of certificates, encryption and creation of MACs.

Server connection class is responsible for connecting to server, authenticating user and handling session certificate creation.

Peer connection class is responsible for establishing connection with peer. This involves exchanging and verifying certificates and establishing first shared key for communication.

Message class is responsible for handling all other peer to peer communication, ranging from creating and sending messages from text or file to receiving and decoding messages. Message class is dependent on peer connection class.

User interface is currently designed as command console, with later option to expand into GUI.