DGMT Protocol Specification
First draft

David Montoya, Raphaël Bonaque, Charles-Pierre Astolfi,
Émile Contal, Martin Gleize

October 30, 2010

1 Naming conventions

In this paper, packet structures for the DGMT protocol will be described in detail. We will employ the following types of terms:

- *data_type*: the following data types will be employed:
  - *byte*: an ordered collection of 8 bits.
  - *int*: an integer of unspecified precision. We will employ both the decimal `[0-9]+` and hexadecimal `0x[0-9A-F]+` notations in this paper.
    Implentors's note: implementors are free to choose how they store integer data, as long as the precision employed is enough to represent the whole set of expectable values.
  - *bool*: a boolean, whose value can be either 1 or 0.
  - *ustring*: a finite sequence of characters from the Unicode character set.
- DATA_ENCODING: a byte encoding of a given *data_type*.
- `CONSTANT_VALUE`: a constant byte value, used to distinguish different message types or answers.
- `variable`: a variable used to store sensitive protocol data.

We will refer to the value `NULL` to refer to either the 0x00 *byte* or the empty *ustring*.

1.1 Encodings

We define the following data encodings:

- BYTE: data is taken as "is", a *byte* array is formed.
- INTEGER: a big-endian integer encoding of an *int*. The actual interpretation depends on the number of bytes employed.

- BOOLEAN: an encoding for a *bool*. Any non zero value represents 1, while a zero value stands for 0. If more than 1 byte is used, a *bool* array is formed.
- STRING: a ASCII encoding for *ustring*, using an arbitrary number of bytes.
- USTRING: a UTF-8 encoding for *ustring*, using an arbitrary number of bytes.
- ARRAY: an encoding for an array of arbitrary data. Individual data has particular encoding, and this data is concatenated at the byte level to form an array.

Implementor's note: STRING can actually be interpreted as a USTRING without any loss of data. The reason we define the former is so that we can specify the kind of information we expect to have, although this should be done at a higher level of abstraction.

2 Overview

The DGMT protocol defines how a MindTris client can communicate with a MindTris server. Clients must maintain the following state information for each server it connects to:

- `server_pub_key`: The server's public RSA key. Used by the client in communications where secret transmission of data is necessary. The default value is `NULL`.

The client keeps track of the current status of the client/server connection. This could be whether the client has logged in, is in a game, etc. It should include information such as following:

- `connected`: Whether or not the client has received a positive handshake answer from the server. Defaults as 0.
- `logged_on`: Whether or not the client has successfully authenticated itself as a certain user existing in the server's database. Defaults as 0.
- `user`: The user the client has authenticated as. Defaults as `NULL`.
- `lobby`: The id of the lobby the user has joined. Defaults as `NULL`.
- `am_playing`: Whether or not the user is in a game. Defaults as 0.

Conversely, server must maintain state information from each client it's connected to. This should include, **connected**, **logged_on**, **user**, **lobby** and **am_playing**. Additionally, it will include the client's public key (**client_pub_key**), which will be used for communications between peers to authenticate their messages. This key defaults as `NULL`.

All messages sent between the server and the client must be structured as follows:

Name	Encoding	Size (B)	Default	Comment
Protocol	STRING	4	"DGMT"	Default protocol identifier.
Size	INTEGER	2		The size of the message in bytes, including the header.
Payload	BYTE	varies		The content of this field will vary depending on the type of message.

3 Message Types

3.1 Hello

This message is the first message in the handshake between the server and the client. The client will first send a `HELLO_FROM_CLIENT` message to the server, who in turn should answer with a `HELLO_FROM_SERVER` message.

Request:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x00	This identifies the message as `HELLO_FROM_CLIENT`.
Protocol Version	INTEGER	4	N/A	Protocol/Client version; the format is xx.xx.xx.xx.

Response:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x80	This identifies the message as `HELLO_FROM_SERVER`.
Server Answer	INTEGER	1	N/A	Server's response. Must be one of the following: 0x00 (`SUCCESS`), 0x01 (`WRONG_PROTOCOL_VERSION`), 0x02 (`UNKNOWN_ERROR`).
Server Public Key Size	INTEGER	2	N/A	The size of the Server Public Key field. Should be 0 if the Server Answer is not `SUCCESS`.
Server Public Key	BYTE	varies	N/A	The server's public key, used for transmission of secret data from the client to the server.
Message	USTRING	varies	N/A	A human readable message from the server to the client. This could be a welcome message, if the Server Answer was `SUCCESS`, or a few details about the error that occurred.

If the server answers `SUCCESS`, the client will store the server's public key in its `server_pub_key` variable. Additionally the state variable `connected` is set to 1. Any other messages exchanged between the server and the client assume that `connected` is 1.

3.2 Keep alive

Clients regularly send a message with an empty Payload to the server from the moment they received a `HELLO_FROM_SERVER` . They are expected to do so every 60 seconds, otherwise the server will drop the connection, and terminate any ongoing transactions with the client.

3.3 User creation

By this process, a client can create a user to be stored in the server's database. The client will first send a `CREATE_USER` message to the server, who in turn should answer with a `USER_CREATION` message.

Request:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x01	This identifies the message as `CREATE_USER`.
Encrypted User Info	BYTE	varies	N/A	We will describe this content below.

The Encrypted User Info field should contain the following data, encrypted with the RSA-OAEP (SHA-1) scheme, using `server_pub_key`.

Name	Encoding	Size (B)	Default	Comment
Username Size	INTEGER	1	15	The size of the Username field.
Username	USTRING	varies	N/A	A name used to login.
Display Name Size	INTEGER	1	255	The size of the Display Name field.
Display Name	USTRING	varies	N/A	A name that will be displayed to other users.
Email Size	INTEGER	2	320	The size of the Email field. This value cannot be greater than 320.
Email	STRING	varies	N/A	An email address, used to recover a lost password.
Password Size	INTEGER	1	N/A	The size of the Password field.
Password	STRING	varies	N/A	The password that will be used to login.

Valid usernames must match `[A-Za-z][A-Za-z0-9_.\-]*`, and valid password must match

```
^{}.*(?=.\{6,\})(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=]).*$
```

Response:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x81	This identifies the message as `USER_CREATION`.
Answer	BYTE	1	N/A	• 0x00: the user has been created with success. • 0x01: this username already exists. • 0x02: invalid username. • 0x03: invalid password. • 0x04: invalid email.

3.4 Login

By this process, a client can login as a user present in the server's database. The client will first send a `LOGIN` message to the server, who in turn should answer with a `LOGIN_REPLY` message.

Request:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x02	This identifies the message as `LOGIN`.
Encrypted User Login Info	BYTE	varies	N/A	We will describe this content below.

The Encrypted User Login Info field should contain the following data, encrypted with the RSA-OAEP (SHA-1) scheme, using `server_pub_key`.

Name	Encoding	Size (B)	Default	Comment
Username Size	INTEGER	1	15	The size of the Username field.
Username	USTRING	varies	N/A	A name used to login.
Password Size	INTEGER	1	N/A	The size of the Password field.
Password	STRING	varies	N/A	The password that will be used to login.

Response:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x82	This identifies the message as `LOGIN_REPLY`.
Answer	BYTE	1	N/A	• 0x00: login success. • 0x01: username does not exist. • 0x02: bad username/password. • 0x03: too many tries, try again later. • 0x04: login success, but another instance was disconnected elsewhere.
Display Name Size	INTEGER	1	255	The size of the Display Name field.
Display Name	USTRING	varies	N/A	The user's name that is displayed to others.

Normally, the server will only give a success response if the username and password provided by the client match in the server's database. The variable `logged_on` is set to 1 and `user` is set to the provided username on both the server and the client.

3.5 Lobby Creation

By this process, a user can create a lobby. This is only possible if `logged_on` is set to 1.

Request:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x03	This identifies the message as `CREATE_LOBBY`.
Lobby Name Size	INTEGER	1	N/A	The size of the Lobby Name field.
Lobby Name	USTRING	varies	N/A	The name of the lobby to be created.
Has Password	BOOLEAN	1	0	Whether or not the lobby will require a password.
Encrypted Lobby Password	BYTE	varies	N/A	This field is empty if the Has Password field contains 0.

The Encrypted Lobby Password field, if provided, should contain the following data, encrypted with the RSA-OAEP (SHA-1) scheme, using `server_pub_key`.

Name	Encoding	Size (B)	Default	Comment
Password Size	INTEGER	1	N/A	The size of the Password field.
Password	STRING	varies	N/A	The password that will requested to join the lobby.

Note that a valid password must match

```
^{}.*(?=.\{6,\})(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=]).*$
```

Response:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x83	This identifies the message as `LOBBY_CREATION`.
Answer	BYTE	1	N/A	• 0x00: lobby created with success. • 0x01: invalid password. • 0x02: you do not have enough rights create a lobby.
Lobby ID	INTEGER	4	N/A	An ID generated by the server for the lobby than has been created. If Answer wasn't 0x00, then this value is 0x0 (`NULL`). Otherwise, this value should be generated so that all current available lobbies have unique IDs.

3.6 Lobby List Retrieval

Users might want to know the list of available lobbies. This is only possible if `logged_on` is set to 1.

Request:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x04	This identifies the message as `GET_LOBBY_LIST`.

Response:

Name	Encoding	Size (B)	Default	Comment
Type	INTEGER	1	0x84	This identifies the message as `LOBBY_LIST`.
Lobby List Size	INTEGER	1	N/A	The size of the Lobby List array.
Lobby List	ARRAY	varies	N/A	An array of Lobby data, described below.

The Lobby List contains an array of the following data structure:

Name	Encoding	Size (B)	Default	Comment
Lobby ID	INTEGER	4	NULL	The lobby ID.
Player Count	INTEGER	1	N/A	The number of players present in the lobby.
Player Allowed Count	INTEGER	1	N/A	The maximum number of players allowed.
Password Protected	BOOLEAN	1	0	Whether or not this lobby requires a password to join.
Creator Size	INTEGER	1	N/A	The size of the Creator field.
Creator	USTRING	varies	N/A	The display name of the creator of this lobby.