Introduction:

As the internet has grown, many security measures became a necessity, to prevent attackers from accessing systems, or stealing information easily. And Data encryption has become one of the most essential parts of data protection. AES stands for Advanced Encryption Standard. This is a small variation of Rijndael which was first born, to replace the DES algorithm. Is an iterated block cipher algorithm which means that encrypts and decrypts a block of data by the iteration or round specific operations.

After DES was found to have vulnerabilities that could compromise the security of the algorithm AES became selected by the NIST, as a standard, after being tested under multiple situations including cost of implementation, and was found to have the best performance, security and efficiency. It was developed originally by Vincent Rijmen and Joan Daemen.

The principal changes that the Rjindael received to become AES were the part to change some names, and in order to improve readability of the standard. And this naming can be found in the official document of NIST: https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf

AES has improved communication security against multiple attacks, one which is the most popular, and is the Man-in-the-Middle attack. This attack consists of acting as an intermediary, between the communication, and spy which packets are being sent to a server, or received. And AES encryption is a mitigation to fight back this attack, as it makes it difficult for the attacker to get the messages.

This document will explain an implementation of a secure chat system made in C++, similar to penguin chat. And also how does Rijndael influence the implementation in regard to performance, security, and also, what was required for the implementation, including some tests with wireshark.

Theoretical Framework:

AES stands for Advanced encryption standard, and it is the standard accepted by NIST for encryption. It replaced older algorithms such as the DES, and is an iterated block cipher algorithm, Rjindael encrypts and decrypts its data by iterations or rounds of iterations doing a specific transformation. And it supports encryption key sizes of 128, 192, and 256 bits handling data in 128-bit blocks. The algorithm is designed based on DES, taking into account multiple points that were their goals, such as making it resistant to all known attacks; to ensure source code compactness and speed on multiple computing platforms; and a simple design.

The Rijndael accepts the input as 8-bit byte arrays that create data blocks segmenting those. And the plain text input is added onto a map of bytes, which correspond to the cipher key, and is also a one dimensional 8 bit byte array. In the algorithm, it is used for multiple transformations. We also have the initialization vector, which is an arbitrary number used with the secret key during the data encryption. This value is typically required to be random or pseudo random but sometimes IV only needs to be unpredictable or unique. Randomizing this is important to archive security, to prevent attackers inferring relationships between segments of encrypted messages.

This is achieved by the library osrng which provides randomness to our encryption. The IV size depends on the primitive used, and generally for block based encryption is the same size as the key. It basically ensures that the same message does not return the same value each time it is encrypted.

On the other hand we got the block ciphers which operate on blocks of text of fixed size, and in the case of AES it's 128 bit, but it has support up to 256 bit sizes, and it is a symmetric algorithm.

Subkey and key, on its own are the keys required, in the case of subkey, it’s generated by the derivation process, as seen on the source code, The cipher key needs to be expanded and the derived key must be used to ensure security and protect the algorithm.

Rijndael encryption actually happens along multiple matrix operations in multiple rounds, and the number of loops to encrypt, depend on the size of the blocks, the more block size, the more rounds are required, and this algorithm, has its fundament on byte replacement, using swap or XOR operations, generating 10 128-bit

keys, from the 128-bit key, and they are stored in 4x4 tables. Then, the plain text is divided on the same 4x4 tables. And each 128-bit plain text goes through a different number of rounds, generating the cipher text after the 10th round.

Other than the encryption key, and IV, there are different modes to encrypt the data, and these provide different security levels. AES is a mostly symmetric encryption algorithm. First it does the key expansion which is added into a set of round keys. Then it does the initial round, And after that it does the different rounds with the subbytes (the 4x4 matrix segment), shift rows, which shifts rows, and then mix columns and add the round keys. Finally it has another final round.

As the algorithm of AES is way too long, I decided to use a library called Crypto++,A library, is a segment of code that can be called as a header in C++, commonly are sets of functions, namespaces and classes that get imported into the program, and make programming more clean and easier. Crypto++, is a library focused on cryptography. This library tries to resist side channel attacks implementing various remediations. And this library also uses hardware instruction when possible for block ciphers, hashes and other operations. The hardware acceleration remediates some timing attacks.

This library also provides multiple encryption algorithms in order to protect, and also it is commonly used to protect information, Is a library based on classes, for cryptographic schemes.

For the development of this program, we used the CBC mode, CBC stands for Cipher Block Chaining, which is a mode of operation for block ciphers like AES, which is used for arbitrary length, where the plain text is divided into fixed size blocks. This mode is achieved by XOR-ing the first plain text block with an initialization vector, before initializing it. The greatest advantage CBC has over EBC is that, with CBC mode, identical blocks do not have the same cipher. This mode is provided with the IV and the plain text is divided into blocks and needs to add padding data, but first a XOR operation is required, plaintext XOR Ini. Vec. In this mode even if we encrypt the same plain text block, we will get a different ciphertext block. We can decrypt the data in parallel but it is not possible when encrypting data.

CBC introduces an initial random block, known as the Initialization Vector and combines this with the previous one with the result of static encryption. producing a different output each time.

It is important to mention that according to a microsoft article in 2022, (https://learn.microsoft.com/en-us/dotnet/standard/security/vulnerabilities-cbc-mode) is not secure anymore if the data is not verified under certain circumstances. It is performed by using a padding oracle attack which is a type attack against encrypted data that permits the attacker to decrypt the content without the key.Basically it consists of testing segments and get whether it is performing the correct process or not. Imagine playing poker, and you get the actions of your opponent by watching their reaction. It is the same, this type of attack is based on testing whether the result is going the correct way or not. and it permits deciphering the text without the key. it's fundamentals of this attacks are based on PADDING, as I mentioned before, the AES, has fixed size blocks, and whenever it don't fit these block sizes, padding is used to cover the remainings and keep the algorithm working, and using this padding, we can perform an ORACLE ATTACK, by testing whether the information is valid or not. The proper mitigation for this attack, is to detect the changes into the message, and reject to perform any actions if it was done, by this we use a SHA256 sum to verify the integrity of the information. HMAC signatures can work for the validation using a secret key.

For the development of the user interface on this project, I used OpenGL which stands for Open Graphics language, it is a library which provides a state machine inside. By state machine, I mean that after we created the OpenGL context, this means that by using instructions to the machine we modify the drawing stack, and the rendering modes that the machine provides, and using this we can draw multiple stuff, it can be defined as a multilingual platform or API for 2D and 3D graphics. For the development of this project GLUT was used, which stands for GL utilities, and is a library with OpenGL 2.0 support which even though it may be considered to be outdated, it is also considered good for simple graphics, and also for prototype development. As well as quick graphics that can be stable and provide compatibility with almost any system that has a GPU. Please note that for the development of this project it is required to have a knowledge of OpenGL functions, and modes, which

as it is not the focus of the project, will not be completely explained in detail.

For the development of this project the standard libraries of Linux were used for remote connection with the server. I will be working with Sockets. We can imagine a socket, as if we had a cable that can be configured and connected into a computer. It provides the capability for the communication between 2 computers or processes and are basically the medium for communication, There are multiple types of sockets, which are SOCK_STREAM (the one used) which provides the possibility to keep information integrity, and is basically TCP connection, we also have DGRAM_SOCK which stands for datagram socket, and can be considered as UDP communication as it does not require an open connection. As the resources for our server are limited, the server was written using PHP. And our socket must be SOCK_STREAM, to perform the HTTP request to the server, and get the response complete, for the application to properly parse and prevent data loss.

A protocol basically is the form/format that our message will have so the other computer can understand the message in this case we are going to be using the HTTP protocol which islands for HyperText Transfer Protocol, the HTTP is basically the request and response we will perform to the server using the socket, it can be considered as the shape the message has. A way of understanding the importance of the protocol are communications among politics. When they need to be polite, they have a set of rules for the communication, well the computers are almost the same, they need a set of rules before communication, and this set of rules includes the technical language.

Development:

** Steps to compile the program:*

The program includes a custom script for building itself, this script basically will be in charge of assembling the entire program, and setting up the environment for development, but if we manually wanted to build the program, after we have the code, we will install the packages: krb5, libgcrypt, nettle, openssl, crypto++, freeglut, git and g++.To install these packages you are going to use your packet manager (this program was made for linux) In my case it's PACMAN, as it's the default packet manager that comes with arch linux.

the command to install the packages is:

```
sudo pacman -Syu krb5 libgcrypt nettle openssl crypto++ freeglut git g++
```

The packet manager should leave the libraries itself automatically in the ***/usr/include*** folder.
And then we will run the following command:

```
g++ -fpermissive -w -o chat main.cpp -lGLU -lglut -lGL
lib/imgui/backends/imgui_impl_glut.cpp
lib/imgui/backends/imgui_impl_opengl2.cpp lib/imgui/imgui*.cpp -pthread
-lcryptopp;
```

the first part handles warnings, by using the flags -fpermissive and -w, the second part -o chat main.cpp gives the program to compile, and the output will be set as chat, and finally the linkers for the libraries are included as the backends, thread and -lcryptopp, which will make sure to link the libraries.

Otherwise, we do not have the interest to do a manual installation for the program, we can just execute the script compile.sh with the parameter --client-audio <- to assemble with audio libraries. as shown below:

```
./compile.sh --client-audio
```

And this will basically set up the development environment for us, and compile the program.

The program is divided into multiple folders, having a structure, to keep everything organized for the development. The structure can be seen on the following On the list we can see several files and folders which are going to be explained now:

- **README.md** > It is a document which just has relevant information about the program, it's used to display this same building instructions on github (as this program is being released as OpenSource)
- **Server** > Contains the different PHP files and the DB (database) script, these files will handle the server and the client connections inside the program, basically consists of all the server side operations required for the program to work. Please notice that to connect to a custom server you must change the domain www.pbonyxapi.slpmx.com on the files: Quim_chat.h and Quim_users.h, so the program connects to your own server.
- **assets** > this folder will just contain the resources required for the game, basically just the images, and 3D models in .obj format used for the game.
- **chat** > is the output program generated after the compilation, if you are not using arch linux, or build your own program just delete this.
- **compile.sh** > this is the compilation script mentioned before.

- **include** > this includes 2 main classes developed for the game, are just included for the game that are not part of the core.
- **lib** > in this folder you may find all the third party libraries used for the game, this is to keep organization within the program
- **src** > all the source code core files for the essential functionalities of the program go here. The cryptography class has the security and encryption for the game client.
- **main.cpp** > it's the main program, basically the important file where the program starts and the one file that will be assembled when we execute the compilation command.

** The Code:*

- Crypto class {Quim_crypt.h file}:

- This is the file which contains the cryptographic functions, as the key derivation, and the encryption and decryption functions as well as the base64 encoding to format and send the messages to the server properly.

```
using namespace CryptoPP;
class Phoenix_encryptor {
   public:
       int psk[40] = {
           0x4f ,0x62 ,0x21 ,0x79, 0x6b, 0x4a, 0x24, 0x35,
           0x36 ,0x30 ,0x69 ,0x75, 0x26, 0x72, 0x4f, 0x56,
           0x26 ,0x43 ,0x67 ,0x65, 0x24, 0x6e, 0x69, 0x65,
           0x69 ,0x74 ,0x32 ,0x30, 0x21, 0x36, 0x61, 0x48,
           0x67 ,0x58 ,0x26 ,0x43, 0x67, 0x31, 0x79, 0x65
       };
       std::string gpassword;
       SecByteBlock key;
       SecByteBlock iv;
       Phoenix_encryptor();
       void decompress_passwd();
       SecByteBlock Derive_key(std::string _secret);
       std::string Phoenix_encrypt(std::string _emessage);
       std::string Phoenix_decrypt(std::string _emessage);
       std::string Phoenix_b64_enc(std::string _string);
       std::string Phoenix_b64_dec(std::string _string);
};
```

Figure 1.0: the definition of the cryptography class

This first part of the source code, will define the object that we will be used to encrypt the data, PSK is the real password for encryption, as we do not want that someone with the client, does reverse engineering to our program and find the decryption key, they are obfuscated as hexadecimals, on an array and there will be a function which assembles the password as a string. SecByteBlock key, iv; are the initialization vectors and our expanded key that will be derived later on to generate the subkey for our encryption
and at the end we have the functions that will be defined for all the encryption.

```
Phoenix_encryptor::Phoenix_encryptor(){
   decompress_passwd();
   key = Derive_key(gpassword);
   iv = Derive_key(Phoenix_b64_enc(gpassword));
}
```

Figure 1.1: this is the class constructor executed when the class is created.

The constructor when the program starts, will generate our SubKey for the algorithm, and will generate our Initialization Vector, please notice that our derived key, for the IV, comes from the base64 of the original key.

```
void Phoenix_encryptor::decompress_passwd(){
   for(int i = 0; i < 40; i++){
       if(psk[i]!=NULL ||psk[i]!=0x00) {
           gpassword += char(psk[i]);
       }
   }
   std::cout<<"decompression process completed! ..."<<std::endl;
}
```

Figure 1.2: the deobfuscator function for our hexadecimals on the password.

This function on the figure 1.2 will manage to get each element of the hidden password, and will push it to the string gpassword, it is important to ensure that the psk has no NULL characters to prevent a segmentation fault (code dump) error, and the char() function converts from a value, to ascii char.

```
std::string Phoenix_encryptor::Phoenix_b64_enc(std::string _string){
   try{
     std::string encoded;
     Base64Encoder encoder;
     StringSource source(_string, true,
     new Base64Encoder(new StringSink(encoded)));
     return encoded;
   } catch (const Exception &e) {
       std::cerr<<e.what()<<std::endl;
```

```
        return "";
    }
}
```

Figure 1.3: Base64 encoding, for our strings

The figure 1.3 contains the function for the base64 encoding, this will limit the result to specific characters, and blocks, to keep the result possible for manipulation on the terminal, and also to make it possible to include it on the HTTP request, base 64 encoding is a way of formatting the string. the code basically does a try operation to verify it it can encode the message and it if can't catch the exception and return the reason of the error, and inside we got a string called encoded which is where our encoded string will be stored, then we have the, encoder which is the object that will be in charge of the encoding, and we tell the object to perform the operation, with _string and store it to encoded with the lines of String Source. and finally a return operator to return from the function sending our encrypted string back as a response. Most functions when using Crypto++ will have this structure.

For example the decoding base64 function:

```
std::string Phoenix_encryptor::Phoenix_b64_dec(std::string _string){
    try{
      std::string decoded;
      Base64Decoder decoder;
      StringSource source(_string, true,
      new Base64Decoder(new StringSink(decoded)));
      return decoded;
    } catch (const Exception &e) {
        std::cerr<<e.what()<<std::endl;
        return "";
    }
}
```

As we can see, we still use the same structure for the encoding, but instead of an encoder object we used a decoder, to make the operation, and returns the original data as plain text and not as base64 encoding.

```
SecByteBlock Phoenix_encryptor::Derive_key(std::string _secret){
   SecByteBlock key(AES::MAX_KEYLENGTH);
   std::string salt = "Xj9.$dfgMyLove092D$@3";

   try{
       HKDF<SHA256> hkdf;
       hkdf.DeriveKey(key, key.size(),
                  (const byte *)_secret.data(), _secret.size(),
                  (const byte *)salt.data(), salt.size(),
                  NULL, 0
       );
   } catch(const Exception& ex) {
       std::cerr << ex.what() << std::endl;
       return SecByteBlock(0);
   }
   return key;
}
```

Figure 1.6: key derivation process

This function will perform the key derivation process onto our encryption, the first line defines the key length to the maximum possible, on the examples is commonly set as a default size, and this will prevent the decryptor to tell that the blocksize is invalid, the second line is a SALT, the salt which will provide randomness to our encryption, it is recommended that salts are salted, this means to NOT repeat any salts, which for time and simpleness it's not done in here.And now back to the other structure of trial for the errors, in this case we are going to use HKDF is the HMAC based key derivation. and it is the function to make a more secure derivation of the key, that follows the extract then expand idea, and in this case we are telling HKDF function to use the SHA256 algorithm to our key operations and the generation of subkeys. And after the operation of derivation has concluded we return the derived key (which we use on the constructor to make the IV and Key for our encryption)

```
std::string Phoenix_encryptor::Phoenix_encrypt(std::string _emessage){
   std::string _result;
   try{
   CBC_Mode<AES>::Encryption dcipher;
   dcipher.SetKeyWithIV(key, key.size(), iv);

   StringSource source(_emessage, true,
       new StreamTransformationFilter(dcipher,
       new StringSink(_result)
   ));
   }catch(const Exception& e){
           std::cerr << e.what() << std::endl;
           return "";
   }
   return Phoenix_b64_enc(_result);
}
```

Figure 1.9: This is the actual encryption function

The encryption function, starts defining the string where we are going to store the result, then back to the other structure, we start by defining the mode to CBC and create the encryption object. Then we initialize the object by setting the key, the size of the key and the initialization vector. By this moment we have already initialized our encryption object, and all left is to do the encryption which is done with the stringsource object, this will return the ciphertext as a lot of weird characters as the old FORTRAN version did with the DES encryption, so, to prevent this we return the result as a BASE64 encoded string which will return something similar to this:

```
CoVeQ3USEia1ezo6AIshyPN3gU6DwXD17Q+YZk7nnx9z5pl4=
```

And this will store our encrypted message. Now we need a function to recover the original message, which is precisely what the decrypt function does.

```
std::string Phoenix_encryptor::Phoenix_decrypt(std::string _emessage){
   std::string _result;
   try{
   CBC_Mode<AES>::Decryption dcipher;
   dcipher.SetKeyWithIV(key, key.size(), iv);
   StringSource source(Phoenix_b64_dec(_emessage), true,
       new StreamTransformationFilter(dcipher,
       new StringSink(_result)
   ));
   }catch(const Exception& e){
           std::cerr << e.what() << std::endl;
           return "";
```

```
    }
    return _result;
}
```

Figure 1.10 this contains the decryption function

So for the decryption function we have exactly the same structure than we have on the encryption function by setting up the encryption mode, the keys that are used for the decryption etc. and this is done, as for decryption we must revert the steps previously done for the encryption, but for the case on this function when passing up our encrypted message to the function we first perform the base64 decoding, to recover the original string with the weird characters before encoding, and them we perform the decryption. also it is important to notice that we use de decryption object instead of the encryption object.

And finally we initialize the care_hound object which is the class we are going to call to perform operations.

```
Phoenix_encryptor *care_hound = new Phoenix_encryptor();
```

So, to this point we already explained the things required for the original purpose of this project, so for the overall explanation of the code, I'm going to explain the logic, behind, and the structures, rather than the code line by line, this to save time, and to prevent the document becoming tedious, confusing or useless.

- Player and user class {Quim_player.h and Quim_user.h files}:

- Quim_user and Quim_players are related to each other, but they do different things, as they are focused on different jobs. the Quim_user, is designed to perform the Network operations to call the server,and update player position and connections, while Quim_player is the file focused on management of the player objects.

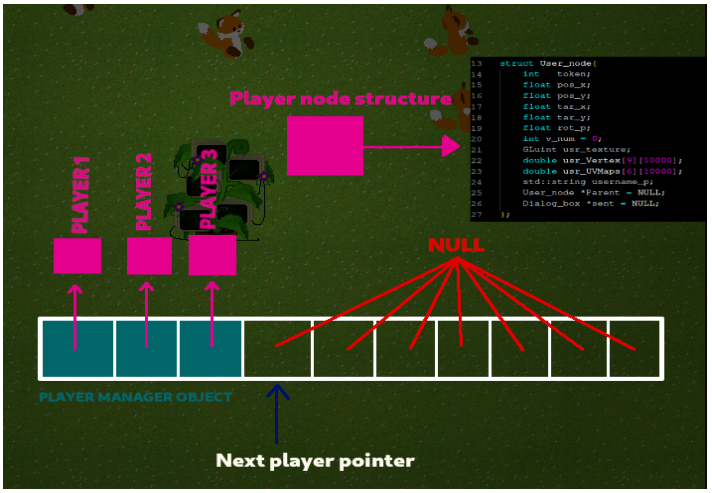


Figure 2.1: The visual representation of how does the player structure work

In this visual representation of figure 2.1 we can see how the player structure works. we have an array of pointers which at first are looking at NULL, and as the players join to the game, the pointers start to look at a structure which are the nodes, and these nodes contain the information of the players, each time a new player is added to the player manager, the next player pointer moves one space to the left, and that spot where the pointer was, starts pointing to a player structure, which represents that player on the list, we could interpret this as a list of objects that contain the properties of each player in game.and as we require, we can index players by its numbers using the position on player manager, basically this permits to have an array of players, giving us the possibility to index multiple properties, as if it was a normal array, but to access these players, we require an Auxiliary pointer, to store the data temporarily as we read or write onto the structure. so.. basically what the file quim provides, are functions to manage these structures. Please notice that it also includes 2 particular functions that in theory where made to allow different player models and textures for avatars, but are not properly implemented for time situations. And these functions are:

```
GLuint  Quim_load_texture(char *filename, int w, int h);
   int     Quim_load_model(char *filename, double mdl[9][10000], double
_uv[6][10000], int _n);
```

the texture loading function, uses the library stb_image.h in order to load a .bmp texture into the game. It is used to avoid compiling the textures into the game.

Quim_load_model is in charge of building the models, these are in .obj format, the obj format is a basic format in which everything is triangulated, and for compression, instead of including all vertex building the model on specific order and triangles, what this format does is to include the list of the vertices, and a list of the UV mapping textures, and finally, the map for building the model, which includes instructions for building each triangle of the model, and in case vertex are repeated, just index that same vertex. this is a good evolution of the RWX model format also created by wavefont, the first number indicates whether is a vertex, uv mapping or constructor, and what our loading function does, is to parse these values and reassemble everything on arrays that later on

will be used to draw ou model. For security inside the parsing we use the SSCANF function to prevent a buffer overload.
to draw the models we are just going to loop all the way around the player manager list, and accessing with the auxiliar pointer our players, we fetch the vertex, uv mapping and GLuint texture properties, as well as the coordinates which are based on the prefix pos,rot, please notice this is important, as we have 2 coordinates on each player, one represents the target position, and the other represents the actual position, this is done, to prevent weird jumpings or low frames, and just see how our players move smoothly, basically uses a pathfinder to reach the positions instead of moving frame by frame (which would decrease the framerate way too much as we are using 2 threads for the program but this will be explained further on the document).
And the last relevant function inside the Quim_player file is the updating/parsing function:

```
void Player_manager::Players_Update(){
   for(int i = 1; i < user_ptr; i++){
       auxiliar = online_users[i];
       for(int j = 0; j < online_u; j++){
       if(strcmp(auxiliar->username_p.c_str()
         , g_users_online_[j].c_str()) == 0){

               auxiliar->tar_x = g_upositions_x_[j];
               auxiliar->tar_y = g_upositions_y_[j];
               auxiliar->rot_p = g_upositions_r_[j];
           }
       }
   }

   for(int i = 0; i < online_u; i++){
       bool _wasfound = false;
       for(int j = 1; j < user_ptr; j++){
           auxiliar = online_users[j];

           if(strcmp(auxiliar->username_p.c_str(),
            g_users_online_[i].c_str()) == 0){
               _wasfound = true;
               j = online_u-1;
           }
       }
```

```
        if(_wasfound == false){
            Push_node_user(g_users_online_[i], 0,0,0, DEF_MDL, DEF_TEX);
        }
    }
```

This function is relevant as it loops along the fetched users and the current known user list and it performs multiple operations, the first block updates the targets for the known players, and the second segment, loops to find if there are unknown users on the database, and then it assembles them to be known users on the player manager structure.

Now in regard to the Quim_users, it basically contains 2 functions:

```
        void Player_fetch_coords();
        void Player_post_position();
```

The first one, will ask the server for a list of all the players registered into the temporal database of players in game, and will parse the result as a list with the format: k <player> <x> <y> <r>, and will parse that same results into many arrays that I use in the update_player function, to perform the player management operations:

```
std::string g_users_online_ [1000]; //
int         g_upositions_x_ [1000]; // THESE ARE THE USERS PARSED
int         g_upositions_y_ [1000]; // FROM THE SERVER
int         g_upositions_r_ [1000]; //
```

So, in this way it is easier to manage players in game, and perform required operations when needed.

The second function needed is post position, which basically asks the API to store our current position after a mouse click, this to tell the other users where we are, and synchronize all the clients positions.

(Socket operations will be explained on the Quim_chat explanation as I'm using the same code for API calls).

So, overall players and users work together in order to ask for players connected to the server, and then return the result as a list, and parse it to perform the list operations contained on the class quim_players, making it simple.

- Chat message system class (Quim_chat.h class):

Now, the chat message works slightly similar to the player, and user classes, but in the case of chat messages, it is limited to just 1 single class, as there's no need of storing the last position, all the time, and know what to draw over all history, instead, this one only requires the last 7 messages sent, so all it does, is perform

the API calls, and keep last 7 messages sent to the database onto an array to draw it. after that, messages don't matter anymore.

- API call operations along chat, users:

So, as mentioned before, sockets are being used to perform these operations, the API call basically refers to calling a link which has multiple parameters, and these parameters will be used to perform an operation, inside the server side of the game, in this case the API is written in C++.

```
void Chat_manager::send_chat(std::string _message){
//-----------------------------------------------
int socket_desc;
struct sockaddr_in serv_addr;
struct hostent *server;
char buffer[4096];
// -----------------------------------------------
std::string url = "www.pbonyxapi.slpmx.com";

socket_desc = socket(AF_INET, SOCK_STREAM, 0);
if(socket_desc < 0){
 std::cout<<"failed to create socket"<<std::endl;
}

server = gethostbyname(url.c_str());
if(server==NULL){}
bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(80);
bcopy((char *)server->h_addr,
        (char *)&serv_addr.sin_addr.s_addr,
        server->h_length);

if(connect(socket_desc, (struct sockaddr *)&serv_addr,
sizeof(serv_addr)) < 0){
  std::cout<<"connection failed :("<<std::endl;
}

_message = care_hound->Phoenix_encrypt(_message);
std::replace( _message.begin(), _message.end(), '+', '*');
std::string _encoded_message = "";
std::stringstream ss;
for(int i = 0; i < strlen(_message.c_str())-1; i++){
   ss<<std::hex<<(int)_message[i];
   _encoded_message = _encoded_message + "%" + ss.str();
```

```
   ss.str("");
}
std::cout<<"message sent is: "<<_encoded_message<<std::endl;
std::string request = "GET /chat_index.php?unamed=" + _self_name +
"&dmode=2&infput=" +_encoded_message+ " HTTP/1.1\r\nHost: " + url +
"\r\nConnection: close\r\nContent-Length:
"+std::to_string(strlen(_message.c_str()))+"\r\n\r\n";

std::cout<<request<<std::endl;

if(send(socket_desc, request.c_str(), strlen(request.c_str())+1, 0) <
0){
 std::cout<<"failed to send request..."<<std::endl;
}
//
int n;
std::string raw_site;
while((n = recv(socket_desc, buffer, sizeof(buffer)+1, 0)) > 0){
 int i = 0;
         while (buffer[i] >= 32 || buffer[i] == '\n' || buffer[i] ==
'\r'){

           raw_site+=buffer[i];
           i += 1;
       }
}
std::cout<<raw_site<<std::endl;
close(socket_desc);
}
```

Figure 3.1 simplified API call

To explain this part, the figure 3.1 I'll be using the simplified API call code, to keep things organized so:

```
//-----------------------------------------------
int socket_desc;
struct sockaddr_in serv_addr;
struct hostent *server;
char buffer[4096];
// -----------------------------------------------
```

This part of the code at the top starts by creating the structures and variables required for our socket management, it basically creates stuff for sockets.

```
std::string url = "www.pbonyxapi.slpmx.com";
```

```
socket_desc = socket(AF_INET, SOCK_STREAM, 0);
if(socket_desc < 0){
 std::cout<<"failed to create socket"<<std::endl;
}
```

This other part of the code at the top, first, sets our server domain, please notice that you need the library netdb.h for resolving this host, and then we start the socket, on the SOCK_STREAM, mode for TCP, and test if the socket was created, if our system has any trouble with creating the socket it will give an error message.

```
server = gethostbyname(url.c_str());
if(server==NULL){}
bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(80);
bcopy((char *)server->h_addr,
        (char *)&serv_addr.sin_addr.s_addr,
        server->h_length);
```

Now this code above is the actual resolution and connection for the domain, what the part of the code does, is first to resolve the domain and get it's IP address, and then set up the resulting IP address onto the socket, in the part that says HTONS(80) we are setting up the port 80 for the game, as it is the web server/HTTP port, so, it is basically a default.

```
if(connect(socket_desc, (struct sockaddr *)&serv_addr,
sizeof(serv_addr)) < 0){
  std::cout<<"connection failed :("<<std::endl;
}
```

This above code is the actual connection to our server, and if something fails, it will kill the program.

```
_message = care_hound->Phoenix_encrypt(_message);
std::replace( _message.begin(), _message.end(), '+', '*');
std::string _encoded_message = "";
std::stringstream ss;
for(int i = 0; i < strlen(_message.c_str())-1; i++){
   ss<<std::hex<<(int)_message[i];
   _encoded_message = _encoded_message + "%" + ss.str();
   ss.str("");
}
```

Now the above code IS NOT NECESSARY for the API call, but it is used, for the part of encrypting our chat message, this code is only

used on the sending message function, and it basically encodes the message as hexadecimals, after it was encrypted.

```
std::cout<<"message sent is: "<<_encoded_message<<std::endl;
std::string request = "GET /chat_index.php?unamed=" + _self_name + "&dmode=2&infput=" +_encoded_message+ " HTTP/1.1\r\nHost: " + url + "\r\nConnection: close\r\nContent-Length: "+std::to_string(strlen(_message.c_str()))+"\r\n\r\n";

std::cout<<request<<std::endl;
```

The above code, basically is the HTTP protocol, it is the message we are going to send to the server, and it contains multiple parameters for the API call.

```
if(send(socket_desc, request.c_str(), strlen(request.c_str())+1, 0) < 0){
 std::cout<<"failed to send request..."<<std::endl;
}
```

And with the above code, we finally send our request to the PHP server, which will interpret our parameters, this is done to prevent Direct connection to a MySQL server, and give a layer of protection by adding multiple checks to the string parameters preventing an Injection to happen.

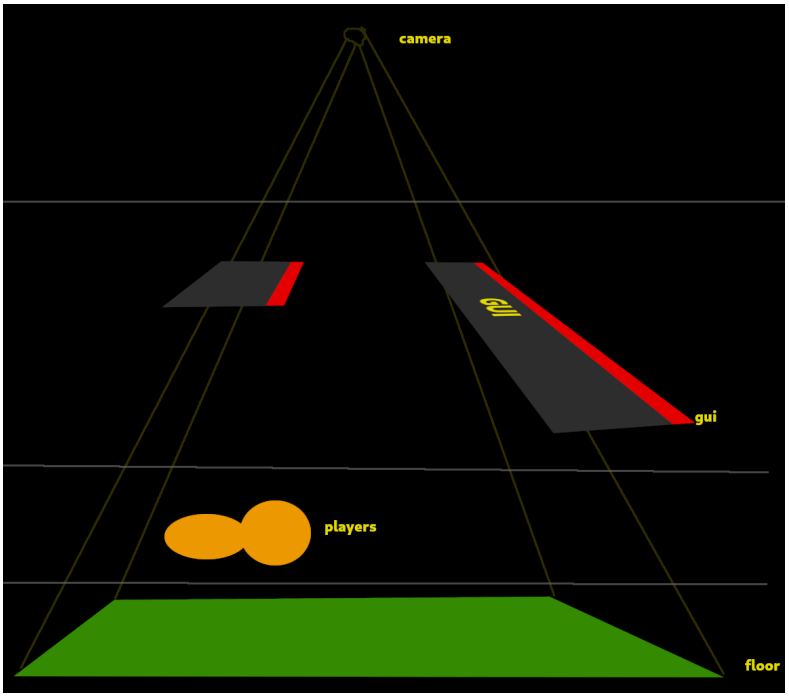
```
int n;
std::string raw_site;
while((n = recv(socket_desc, buffer, sizeof(buffer)+1, 0)) > 0){
 int i = 0;
         while (buffer[i] >= 32 || buffer[i] == '\n' || buffer[i] == '\r'){

           raw_site+=buffer[i];
           i += 1;
       }
}
std::cout<<raw_site<<std::endl;
close(socket_desc);
```

These final lines of the code above, fetch the response from the server, and organize it onto a string, ending the communication and closing the socket, to complete the call.

- Overall OpenGL :

OpenGL basically is working with 2 modes, the orthogonal mode, and the depth 3d mode, this is done, to keep the 2D style having the avatars on a 3D style, this is done, to keep smooth rotation over

the avatar, and basically that's the part for opengl, it does have an initialization, process which can be found on, the main thread, and it has multiple function for input, in which we find keyboard and mouse input, and finally the rendering loop which is constantly running and per each loop it configures the machine to the different modes, and draws the players, the way it draws is by first drawing the 2D environment, which consists on the grass and objects, then it draws the players, as 3D when mode changes, and at the end it uses ImGUI to render the user interface, to display terminal and chat, after that it flushes the buffer (cleans the buffers) and swaps them after a new drawn is made, basically buffer swapping changes the images drawn.
we may visualize what we are drawing as:



Which represents multiple layers of drawing, in which the camera is looking at the top, and this is basically done, to avoid using Raycasting for mouse detection, as raycasting is a slow process which requires a lot of resources, and will slow down the framerate of our program.

- General explanation:

Overall the program follows the same idea for Quim_player, for the data structures, this because it is efficient enough to manage a lot of data in low time, the program is divided into 2 threads, on the one hand, we have the Rendering thread, which is the one in charge of drawing the different things on the screen, and the second thread is represented by this code:

```
void Network_service(int _t){
   //  ----------------------------------------
   //  - REQUEST THE MESSAGES FROM THE SERVER -
   std::cout<<"net service has started. . . "<<std::endl;
   //  ----------------------------------------
   while (!started){}
   while(1){
       std::cout<<"net loop working...."<<std::endl;
       multiplayer_ng->recv_chat();
       std::cout<<"message fetched!"<<std::endl;
       if(strlen(message2_send.c_str()) > 1){
           multiplayer_ng->send_chat(message2_send);
       }
       std::cout<<"net loop ended..."<<std::endl;
       std::cout<<"player engine running . . . "<<std::endl;
       player_engine->Player_fetch_coords();
       manager_usr->Players_Update();
       if(has_clicked == true){
           player_engine->Player_post_position();
           has_clicked = false;
       }
       sleep(1);
   }
   std::cout<<"net service has died...."<<std::endl;
}
```

And this above code what it does, is to make in a loop the call for the different API calls performed along the game, basically this is the network loop which every loop, it sends the messages, and receives them within the server, and then the results of the loops for this thread/service, are sent to the other rendering loop, and these start to draw the results from the server, and this allows to get the multiplayer engine.

on the cases for click or message send, if tests if there is something to do before, to reduce the amount of server calls on the API, to prevent the server from overloading.

Basically what the source code does is to send a message to the server if the message is using the mode 1 then it is a fetch message and the server will interpret this and send a response, then our program will fetch that response, and will set everything into global variables, so the rendering can display the results of the response, if the mode is not 1 then the server will be set to posting mode and will clean the request and make its job. the

program is divided into 2 services, and the rendering uses 2 modes which are the orthographic and perspective mode, these modes are used to display the player in 3D and the environment in 2D, it also permits the game to have a GUI, which the user can interact with, so, we have 1 thread making requests, and another doing the rendering, our rendering, is based on the quim_player structure, while the network is based, on socket systems, overall we can see the explained parts of the code as the essentials, as I repeat this model for all the services, because of its efficiency, and portability.

- PHP Server:

The messages on the PHP server are handled by a similar string, which only changes the MySQL request and the get arguments, but in general both use the same structure.

```
<?php
   function clean($string) {
       $string = str_replace(' ', '-', $string);
       $string = preg_replace('/[^A-Za-z0-9$\/=]/', '', $string);
       $string = str_replace('$', '', $string);
       $string = str_replace('\'', '', $string);
       $string = str_replace('\"', '', $string);
       return preg_replace('/-+/', '-', $string);
   }
   // --------------------------------
   require ('credentials.php');
   // --------------------------------
   $link = mysqli_connect('localhost', $chat_user, $chat_pass, $chat_datb);
   // --------------------------------
   if(!$link){
       die('couldn\'t connect to mysql');
   }
   // --------------------------------
   $mode = $_GET['dmode'];
   $mesg = $_GET['infput'];
   $uname= $_GET['unamed'];
   // --------------------------------
   $mesg = clean($mesg);
   $uname= clean($uname);
   if($mode == 1) {
       $answ = mysqli_query($link, "SELECT * FROM messages ORDER BY Id DESC
LIMIT 7;");
       echo '$';
       if (mysqli_num_rows($answ) > 0) {
           while($row = mysqli_fetch_assoc($answ)) {
               echo 'k ' . $row["username"] . ' ' . $row["dmessage"] . " $";
           }
```

```
        }
    }else{

        $dmes = clean($mesg);
        $uname= clean($uname);
        $query= 'INSERT INTO messages (username, dmessage) VALUES (\''
.$uname.'\',\''.$dmes.'\');';
        $answ = mysqli_query($link, $query);
    }
    //echo 'finished..\n';
    mysqli_free_result($answ);
    mysqli_close($link);
?>
```

Now I'm going to explain the code for the server management, which is written in PHP.

```
    function clean($string) {
        $string = str_replace(' ', '-', $string);
        $string = preg_replace('/[^A-Za-z0-9$\/=]/', '', $string);
        $string = str_replace('$', '', $string);
        $string = str_replace('\'', '', $string);
        $string = str_replace('\"', '', $string);
        return preg_replace('/-+/', '-', $string);
    }
```

The above code will be used to clean all the unnecessary characters, and prevent the SQL and PHP injection attacks, this is part of the system security, to prevent the unused characters, and finally, returns the real string sanitized.

```
// ---------------------------------
    require ('credentials.php');
    // ---------------------------------
    $link = mysqli_connect('localhost', $chat_user, $chat_pass, $chat_datb);
    // ---------------------------------
```

This part of the code basically makes the connection, it gets the credentials from the SQL database, and then it makes the connection, storing it on $link.

```
 // ---------------------------------
    $mode = $_GET['dmode'];
    $mesg = $_GET['infput'];
    $uname= $_GET['unamed'];
    // ---------------------------------
```

This above code, fetches the different parameters from our client, that we will be using to perform the different operations.

```
 $uname= clean($uname);
    if($mode == 1) {
```

```
        $answ = mysqli_query($link, "SELECT * FROM messages ORDER BY Id DESC
LIMIT 7;");
        echo '$';
        if (mysqli_num_rows($answ) > 0) {
            while($row = mysqli_fetch_assoc($answ)) {
                echo 'k ' . $row["username"] . ' ' . $row["dmessage"] . " $";
            }
        }
    }else{

        $dmes = clean($mesg);
        $uname= clean($uname);
        $query= 'INSERT INTO messages (username, dmessage) VALUES (\''
.$uname.'\',\''.$dmes.'\');';
        $answ = mysqli_query($link, $query);
    }
```

now this part of the code is where we perform the operations, if the mode is 1, then we will enter into fetch mode, it selects the last 7 messages sent to the database, and organize them in the packet, please notice that the $ character is to jump the line, and it has the purpose of parsing. Basically this part sends the messages to the player, and if the mode is not 1, then the system will INSERT the message into the database, after a secondary cleanup, and that will make the routine to send our messages.

```
    mysqli_free_result($answ);
    mysqli_close($link);
```

finally we do a cleanup to prevent memory leaks.

Conclusions:

As a conclusion, the Rijndael algorithm is one of the most secure algorithms that we may find, but it is important to keep into account that it may still have vulnerabilities, overall it is important to keep the information encrypted to prevent the different, people, with malicious intention from reading our messages, data encryption plays a crucial role in ensuring the security of information transmitted over the internet. AES (Advanced Encryption Standard) is a widely accepted encryption algorithm that has replaced older algorithms like DES (Data Encryption Standard).

Rijndael, the algorithm on which AES is based, underwent some modifications and improvements to become AES. These changes primarily focused on enhancing the readability of the standard. AES encryption provides enhanced communication security against various attacks, with the Man-in-the-Middle attack being one of the most common.

While CBC mode offers advantages such as different ciphertext outputs for identical plaintext blocks, it has been found to have security vulnerabilities. Padding oracle attacks can exploit weaknesses in the CBC mode if data is not properly verified. To mitigate this attack, message integrity verification using techniques like SHA256 sums and HMAC signatures with a secret key is recommended.

I learned that Rjindael provides a good level of security inside a chat, and also learned multiple security measures that may protect our code from intruders as good practices for encryption like avoid using the CBC mode, and also I learned the way that Rijndael works, by performing multiple loops on the permutations, and also derivation of keys, having multiple sizes of its blocks, and also I consider interesting the part of the oracle padding attack.

AES supports encryption key sizes of 128, 192, and 256 bits, operating on data blocks of 128 bits. It is a symmetric algorithm, meaning the same key is used for both encryption and decryption. The algorithm takes into account various goals, including resistance to known attacks, source code compactness, speed on multiple computing platforms

The encryption process in Rijndael/AES involves multiple matrix operations in successive rounds. The number of rounds depends on the block size, with larger block sizes requiring more rounds. Byte replacement, swap operations, and XOR operations are fundamental to the algorithm, generating 10 128-bit keys stored in 4x4 tables.

Bibliography:

* Network Socket. (n.d.). In Wikipedia. Retrieved July 2, 2023, from https://en.wikipedia.org/wiki/Network_socket

1. Socket in Computer Network. (n.d.). In GeeksforGeeks. Retrieved July 2, 2023, from https://www.geeksforgeeks.org/socket-in-computer-network/
2. What is Socket? (n.d.). In TutorialsPoint. Retrieved July 2, 2023, from https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm
3. Getting started with OpenGL. (n.d.). In Learn OpenGL. Retrieved July 2, 2023, from https://learnopengl.com/Getting-started/OpenGL
4. The Khronos Group Inc. (n.d.). In OpenGL. Retrieved July 2, 2023, from https://www.opengl.org/

5. Block Cipher Mode of Operation. (n.d.). In Wikipedia. Retrieved July 2, 2023, from https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation
6. What is CBC (Cipher Block Chaining)? (n.d.). In Educative. Retrieved July 2, 2023, from https://www.educative.io/answers/what-is-cbc
7. Crypto++ Tagged Questions. (n.d.). In Stack Overflow. Retrieved July 2, 2023, from https://stackoverflow.com/questions/tagged/crypto%2B%2B
8. Crypto++ Library. (n.d.). In Crypto++. Retrieved July 2, 2023, from https://www.cryptopp.com/
9. An Introduction to the Advanced Encryption Standard (AES). (n.d.). In Medium. Retrieved July 2, 2023, from https://medium.com/swlh/an-introduction-to-the-advanced-encryption-standard-aes-d7b72cc8de97
10. How to Initialize a Vector in a Constructor? (n.d.). In freeCodeCamp. Retrieved July 2, 2023, from https://www.freecodecamp.org/news/cpp-vector-how-to-initialize-a-vector-in-a-constructor/
11. Initialization Vector. (n.d.). In Wikipedia. Retrieved July 2, 2023, from https://en.wikipedia.org/wiki/Initialization_vector
12. What is an Initialization Vector (IV)? (n.d.). In TechTarget. Retrieved July 2, 2023, from https://www.techtarget.com/whatis/definition/initialization-vector-IV
13. Rijndael. (n.d.). In TechTarget. Retrieved July 2, 2023, from https://www.techtarget.com/searchsecurity/definition/Rijndael
14. Rijndael. (n.d.). In Britannica. Retrieved July 2, 2023, from https://www.britannica.com/topic/Rijndael
15. strcmp() Function in C. (n.d.). In Tutorialspoint. Retrieved July 2, 2023, from https://www.tutorialspoint.com/c_standard_library/c_function_strcmp.htm
16. Proper Way of Closing a Unix Socket. (n.d.). In Stack Overflow. Retrieved July 2, 2023, from https://stackoverflow.com/questions/64454060/proper-way-of-closing-a-unix-socket
17. MySQL UPDATE Statement. (n.d.). In w3schools. Retrieved July 2, 2023, from https://www.w3schools.com/mysql/mysql_update.asp
18. Insert Query - Check if Record Exists, if Not, Insert It. (n.d.). In Stack Overflow. Retrieved July 2, 2023, from https://stackoverflow.com/questions/15898599/insert-query-check-if-record-exists-if-not-insert-it
19. Best Way to Test if a Row Exists in a MySQL Table. (n.d.). In Tutorialspoint. Retrieved July 2, 2023, from

https://www.tutorialspoint.com/best-way-to-test-if-a-row-exists-in-a-mysql-table
20. HTTP Request by Sockets in C. (n.d.). In Stack Overflow. Retrieved July 2, 2023, from https://stackoverflow.com/questions/17685466/http-request-by-sockets-in-c
21. HKDF_h not found in Crypto++ Library. (n.d.). In Stack Overflow. Retrieved July 2, 2023, from https://stackoverflow.com/questions/52899602/hkdf-h-not-found-in-crypto-library
22. AES Encryption Key vs IV. (n.d.). In Stack Overflow. Retrieved July 2, 2023, from https://stackoverflow.com/questions/9049789/aes-encryption-key-versus-iv
23. Linking C++ Code - Undefined Reference to 'student::student(std::cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)'. (n.d.). In Stack Overflow. Retrieved July 2, 2023, from https://stackoverflow.com/questions/71231006/linking-c-code-undefined-reference-to-studentstudentstd-cxx11basic-s
24. Convert String to Hexadecimal and Vice Versa. (n.d.). In Stack Overflow. Retrieved July 2, 2023, from https://stackoverflow.com/questions/3381614/c-convert-string-to-hexadecimal-and-vice-versa
25. Encode std::string with Hex Values. (n.d.). In Stack Overflow. Retrieved July 2, 2023, from https://stackoverflow.com/questions/22317978/encode-stdstring-with-hex-values
26. How to Replace All Occurrences of a Character in a String? (n.d.). In Stack Overflow. Retrieved July 2, 2023, from https://stackoverflow.com/questions/2896600/how-to-replace-all-occurrences-of-a-character-in-string
27. C++ Multithreading. (n.d.). In TutorialsPoint. Retrieved July 2, 2023, from https://www.tutorialspoint.com/cplusplus/cpp_multithreading.htm