The RedEagle Team

Login Account Pro Secure

Documentation

Summary

Introduction

Thank you for your purchase! The RedEagle team is proud to present you this fully secured login and account system. We worked a lot on this asset and we hope you will enjoy it ☺

Please leave a review on the Unity asset store to tell us your feelings about it, we will improve it according to your needs.

To install the asset, please read the document 'Instructions setup.pdf'.

The explanations below are detailed so you can understand what's going on in the global picture. There a few "technical" explanations here, since the code is fully documented, it's better to have the explanation and the code side by side. So in this document you will find conceptual information on how everything is organized to work together.

Special thanks to Magic73 for his windows store integration

Let's begin!

General

What you will find in this asset

In this asset you will find two systems, completely separated one to each other:

- The Unity login system
- The Web browser login system

The Unity login system allows your players to connect to your game, get their information from anywhere with their account.

IMPORTANT ABOUT THE WEBSITE PART:
The Web browser system is an improvement allowing your players to connect to their account from a website. It can be useful to create a high scores screen, a community forum, etc… with the same account for gaming and sharing. It is not an asset for a website but for Unity games, so our work is **focused on the game part**, the website is just a start for people willing to create a forum or anything else using user's accounts of the game. We would recommend you, if it's your case, to rather implement a HTTPS website to ensure security, that's the only way of making it secure, just be aware that it's not free and need work. Moreover not any server provider are supporting it.
The game part is completely secure (as long as you never divulgate the privateCertificate.crt file).

First of all the main purpose of this asset is to make your game development a lot easier by not having to deal with database access and the security.

We wanted this asset to be compatible with absolutely every PHP server, mySQL database, and every Unity games. You can add anything you want in your database, any improvement or information your game need, this asset will support it, if not, just send us an email, we are nice guys! ☺

RSA-AES presentation

To protect our data on network we need to encrypt all the communications made between client and the server. So if anyone listen to the communication, he would understand absolutely nothing.

To encrypt the data we use two algorithms:

- RSA
- AES

RSA algorithm is an asymmetric encryption. It means that two keys are used: a public one and a private one.

The public key can be sent to anyone, there is absolutely no information on it. The only utility of the public key is to be used for encryption (**not** decryption).

For a login/account system, we need a two-way communication:

- Client -> Server
- Server -> Client

In this configuration, only the server has the RSA private key. It means that the server is the only one to be able to decrypt information. The client is only able to **send** information, so the communication Client -> Server is working, but the Server -> Client is not.

We could use RSA for the Server -> Client communication too, with the client generating another RSA keys pair, but the real problem of RSA is the calculation's complexity. RSA is extremely secured, it's mathematically admitted that it's impossible to "guess" the private key, however, RSA algorithm is very slow. We couldn't accept such latency on servers for every single data sent. That's why AES algorithm makes part of the system.

AES is a symmetric encryption. It means we use the same key for encryption and decryption. But here is the problem: client and server needs to have the same AES key to communicate, so if we use only AES algorithm, how can the client send the AES key to the server without and ensure nobody gets it except the server itself? RSA is useful here.

We will use the combination of those two algorithms.

How the security works

1. The client (the player on your Unity game) sends a request to your server.
   He is not authenticated yet, so he asks to open a new SSL connection.

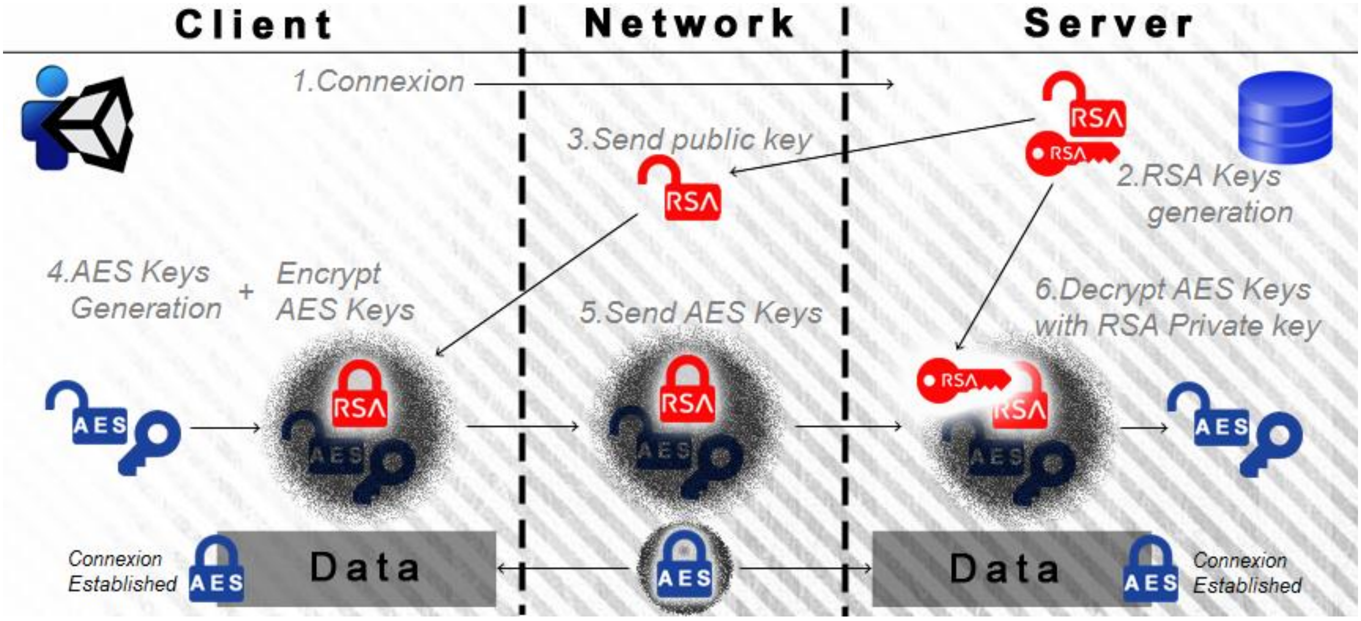
2. The server gets the request and generate a new SID (Session Identifier) and RSA keys pair:
   - The RSA public key
   - The RSA private key

3. The public key is sent to the client on the internet.

4. The client gets the public key (which is **not** encrypted of course) and generate an AES keys pair.
   (The key and the initial vector).
   With the RSA public key, the client encrypt the AES keys pair.

5. The encrypted AES keys pair is sent to the server.

6. The Server gets the encrypted AES keys pair and decrypt it with the RSA **private** key



Both sides (client and server) have the same AES keys pair now.

They can communicate by encrypting data with the AES keys and send them to each other.

The decryption is made with the same AES keys pair (which is kept secret) on both sides.

VERY IMPORTANT:
An attack called 'Man-in-the-middle' could attack such behavior since somebody could communicate with us, faking to be a 'real server'. To prevent this attack, our behavior will not generate a new RSA key pair (public and private) but will use certificate. When you install your game (and your server) you will generate a public certificate and private certificate.

- The public certificate is absolutely not a secret, everybody can encrypt information using the public key.
- The private certificate is an ABSOLUTE secret, it's the only way on earth (mathematically admitted) to decrypt an information encrypted with the public key.

Communication

Now, we know how to encrypt/decrypt data, send information without being read by anyone. Let's see how the communication is made.

The client is using the game, so he's using C# language.

The server is a PHP server.

The communication needs compatibility between the two languages. To ensure that, we will use base64 strings. Base64 strings are handled the same way on PHP and C#.

The database here will never communicate with the client directly. This means that the server get/set information in the database, and it's the only one allowed to do it. If the client wants to get something from the database, he asks the server for it (via PHP scripts).

The server use a SQL request to get/set the information. Then (encrypt it and) send it to the client as an answer.

Let's see how the server is working exactly!

Server

Server functionalities

What's really matter here is to understand that all the communications between any client and the server will use "actions".

The server has **two** types of scripts, the "public" ones, and the "private" ones.

The public ones are accessible from the outside, in fact, anyone who wants to get information from your server will be allowed to use them, they are public. So they need to check identity/IP/Session before doing anything.

The private ones, however, are not accessible from the outside. They are only **used** by the public scripts. If someone is trying to use/see them an error is thrown.

So the server needs to check information received before doing anything, this means that a script needs to know how to check information, and which information must be checked depending on which action is made/asked.
You can see this script as a door, checking you, your identity, before granting you any access. This script is called "Server.php".

The other "public" PHP scripts are used to validate emails addresses etc… (Not really important here, they are easy to understand)

The script "Server.php"

The "Server.php" script:

- First, all the functions used by all scripts are declared here (see the first "include Function.php")
- Then, the session is instantiated or loaded (the session allows the server to save all information it needs to check his identity, keep useful information like AES keys, username, etc…)
- The connection to the database is made with information loaded from the "ServerSettings.php" we filled in the configuration form.
- If an action is specified in the $_POST array the server read and save it.
- The action is checked, assuming the specified action exist, the corresponding script is then included.

This is the global running process of the "Server.php" script.

Session and cryptography

How the Session is made?
It's very easy:

The Server.php script generate a new SID (Session Identifier) if no SID is received from the client.
Then the session is launched with this SID.

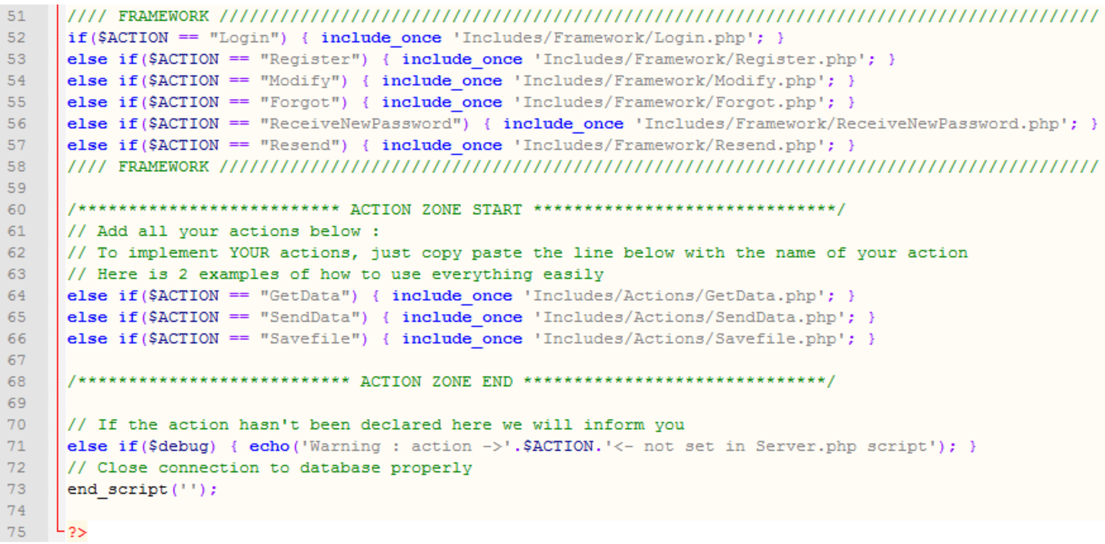
Caution though, the SID can be usurped, indeed, it's of course **not** encrypted. (If it was encrypted, how could the server know which client's AES keys it needs to use to decrypt the SID?)

It means anybody who wants to get the SID can. But it's no big deal since a session token is generated every time a user log in. The 'session token' is a secret key generated client side and will always be decrypted (with AES keys) and checked.

If the session token is correctly decrypted and correct -> the connection is granted.

If a user (Alfred) gave its login information to another person, a friend for example (Donald). Alfred could be connected to his account, if Donald tried to connect, and succeed it, Alfred would be disconnected (an account can only be used by one person).
However **before** Donald disconnect Alfred, its IP would be checked and refused, except if Alfred allow this new IP connection by clicking the link in the email he would have receive. So Alfred won't be surprised to be disconnected once in a while, he would call his 'friend' Donald and yield at him for having disconnect him from the best game he ever played: yours! :p

Actions



```php
//// FRAMEWORK /////////////////////////////////////////////////////////////////////////////////////
if($ACTION == "Login") { include_once 'Includes/Framework/Login.php'; }
else if($ACTION == "Register") { include_once 'Includes/Framework/Register.php'; }
else if($ACTION == "Modify") { include_once 'Includes/Framework/Modify.php'; }
else if($ACTION == "Forgot") { include_once 'Includes/Framework/Forgot.php'; }
else if($ACTION == "ReceiveNewPassword") { include_once 'Includes/Framework/ReceiveNewPassword.php'; }
else if($ACTION == "Resend") { include_once 'Includes/Framework/Resend.php'; }
//// FRAMEWORK /////////////////////////////////////////////////////////////////////////////////////

/************************** ACTION ZONE START ******************************/
// Add all your actions below :
// To implement YOUR actions, just copy paste the line below with the name of your action
// Here is 2 examples of how to use everything easily
else if($ACTION == "GetData") { include_once 'Includes/Actions/GetData.php'; }
else if($ACTION == "SendData") { include_once 'Includes/Actions/SendData.php'; }
else if($ACTION == "Savefile") { include_once 'Includes/Actions/Savefile.php'; }

/*************************** ACTION ZONE END *****************************/

// If the action hasn't been declared here we will inform you
else if($debug) { echo('Warning : action ->'.$ACTION.'<- not set in Server.php script'); }
// Close connection to database properly
end_script('');

?>
```

As you can see if the action is specified in $_POST array it's checked, otherwise the script just leave and tells you that you missed something if you're in debug mode.

Notice the "else if" and not "if". Keep organization this way, for an action we include only one script which contains code and/or other includes. More readable this way.

The ACTION ZONE is where you can add your actions. For example, you could add an action called "Host game" with all the client's information, IP, game type, maximum players, etc… Another action "List games" who displays to the client which games are available, and their information. As you can see it's pretty easy to use ☺

Includes

You know it now, every action means ONLY one "action script" (which can include other scripts and/or code).

So, how an "action script" is made?

First, you have to try to create generics script. It means that rather than check client's information in every "action script" we've made a "tool script" checking it. What is it for? If something went wrong with your code, or if you want to change it, you know where it is, you have only one error to fix/change. And it makes code WAY more readable.

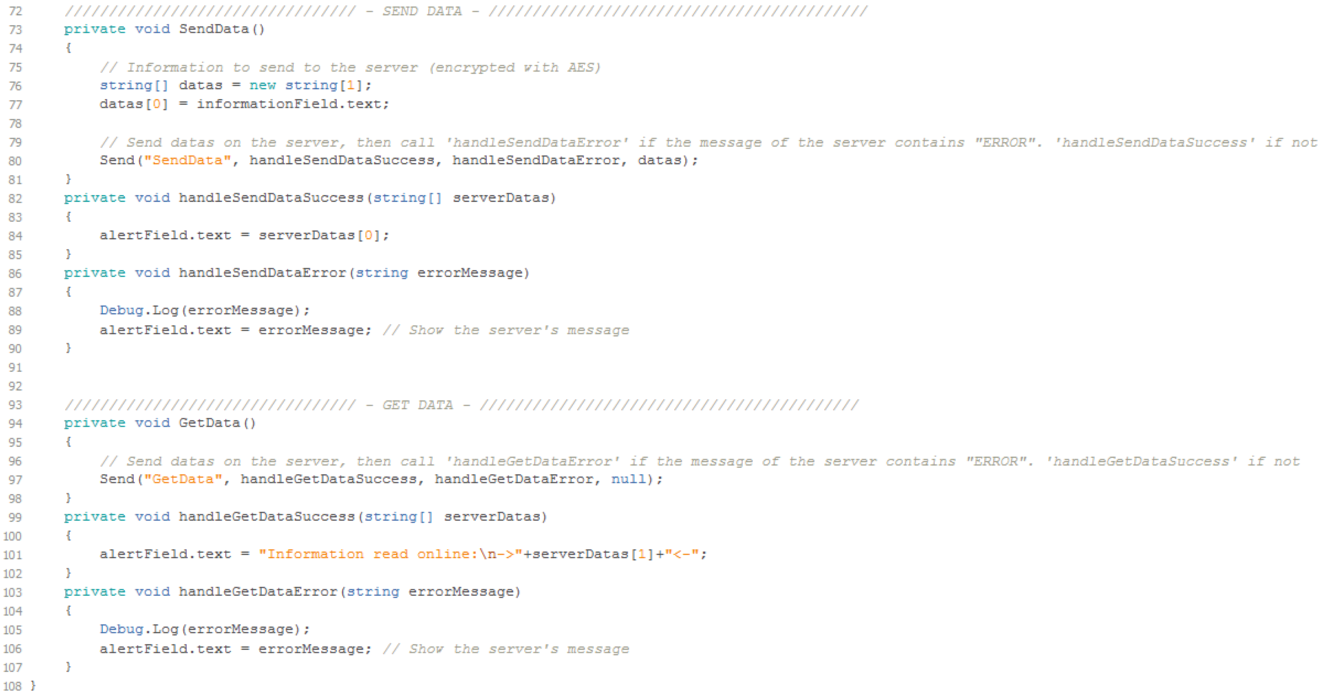
The main key is to use functions for everything, every action, get, set, everything, call them in your "tools scripts" which you use in your "actions scripts". (You can see this implementation as the strategy design pattern if you are an advanced programmer).

GetData.php and SendData.php scripts are perfect examples to show you how to create custom actions.

Client communication

We made communication between your C# scripts and PHP scripts so easy, you will never see how complicated this is ;)

Here are the example1 code, you just have to use the method "Send" which takes 4 parameters:
- The action name
- The method called we the server answer and send the information
- The method to call we the server throw an error
- The array containing all the data you want to send (you can send none by just sending null)



```
    ///////////////////////////////// - SEND DATA - ///////////////////////////////////////////
    private void SendData()
    {
        // Information to send to the server (encrypted with AES)
        string[] datas = new string[1];
        datas[0] = informationField.text;

        // Send datas on the server, then call 'handleSendDataError' if the message of the server contains "ERROR". 'handleSendDataSuccess' if not
        Send("SendData", handleSendDataSuccess, handleSendDataError, datas);
    }
    private void handleSendDataSuccess(string[] serverDatas)
    {
        alertField.text = serverDatas[0];
    }
    private void handleSendDataError(string errorMessage)
    {
        Debug.Log(errorMessage);
        alertField.text = errorMessage; // Show the server's message
    }


    ///////////////////////////////// - GET DATA - ///////////////////////////////////////////
    private void GetData()
    {
        // Send datas on the server, then call 'handleGetDataError' if the message of the server contains "ERROR". 'handleGetDataSuccess' if not
        Send("GetData", handleGetDataSuccess, handleGetDataError, null);
    }
    private void handleGetDataSuccess(string[] serverDatas)
    {
        alertField.text = "Information read online:\n->"+serverDatas[1]+"<-";
    }
    private void handleGetDataError(string errorMessage)
    {
        Debug.Log(errorMessage);
        alertField.text = errorMessage; // Show the server's message
    }
}
```

Database communication

See the script called "Function.php" absolutely every database communication is made in functions! After all, only "simple" actions are made to communicate with the database, better get it in a function in case a field is added/removed/changed.

You will find in "Function.php" a function called "update_table()". It is useful because it makes your code very readable for updates you make, only specified fields are updated and it's clear to see what you are doing. However is can be tricky to use without an example, you can see one in the function
function modify($mail, $username, $password)

Just a tip to debug: All the errors in CAPS (like this "CHECK USERNAME EXISTS FUNCTION DATABASE ERROR!") are database problems, often a request string problem, a bad fields name or something, try to execute the request that causing the error in phpMyAdmin (you will find the problem very quickly if it's just a syntax problem or a table logic).

Recurring development debugging

On the server, as you develop, you will eventually find those errors:

- "Header already sent by…" it's because you have an output (echo php function) occurring somewhere in your code or inclusion. You will find more information at http://stackoverflow.com/questions/8028957/how-to-fix-headers-already-sent-error-in-php

- Server can't be reached. Your server doesn't have the correct path in your "AccountServerSettings.cs" class or you have a compilation problem on your server, run the installation process to make sure all the steps are validated.

- You are on localhost and nothing works? Check all your PHP script, ensure they all start with <?php and end with ?>

- Same solution if your server is showing you some code on your page, you missed something to let your server understand you are talking in PHP language.

- Bad length exception or a decryption exception? Ensure you don't try to decrypt an empty string. Are your keys correct?

- Some data are not empty, but they should be, especially passwords? Are you trying to hash with SHA256 encryption too? Because HASH([EMPTY]) gives you a string, not empty!

- You are trying to build a Webplayer and you get a crossdomain.xml error? Create a text file, rename it crossdomain.xml (remove the .txt if any at the end of course) and just paste this:

  ```
  <?xml version="1.0"?>
  <cross-domain-policy>
  <allow-access-from domain="*" />
  </cross-domain-policy>
  ```

  IMPORTANT: save the xml file as… And select the encoding ANSI
  Place this little xml file at your server root (yes the root), it's working now.

Client

Client functionalities

On the client you will find 6 scenes. Each of them has a role:

- Login -> Connects and validates IP addresses, this scene is the "menu" when the user is not connected
- Register
- Forgot -> Allows users to get their information back
- Modify -> Allows users to change their information
- Example1 -> The first example shows you how easily send and get data on your server (securely)
- Example2 -> The second example shows you how to send and get a file on your server (securely)

In every scene you will see a "manager". This script handles every UI (i.e. User Interface) events made on the scene.

The Login scene

At start the Login scene establishes the SSL security (RSA&AES), in the same time a session token (a secret session key) is generated to check if the communication is made between the 'real' server and client.

You can see how the security is established by studying the "LoginManager" class, and the class it extends (like all other managers) 'ServerManager'.

Coroutines

What are Coroutines made for?

Coroutines are asynchronous. This means that the sender launch them on a different thread than the main process. This way the client continue to use everything is the scene without being frost while waiting for the server answer.

This way we can send a request to the server without being frost until the server answer.

UserSession

If you check the "UserSession" class you will see it's not attached to any object of any scene. It's a "static" class. This "object" (since you can see it as a singleton pattern) allows you to save user information. You can add any field you want in here as soon as it's static, it won't be cleared between scenes.

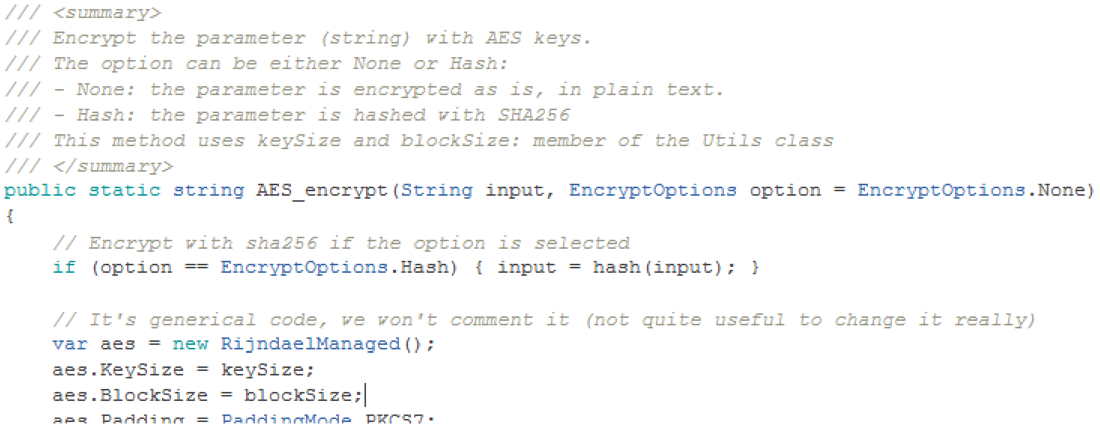
Advice: If it happens that your UserSession is cleared, check if you didn't called the "clearSession" method somewhere.

Advice2: If you add a field in the UserSession class, add it in the "clearSession" method too (This way it will be cleared when you call the clearSession method).

Utils

You will find in this class all methods you would need to communicate with the server, it contains the “tools” methods.

Pay attention to a method in particular: Utils.AES_encrypt()



```
/// <summary>
/// Encrypt the parameter (string) with AES keys.
/// The option can be either None or Hash:
/// - None: the parameter is encrypted as is, in plain text.
/// - Hash: the parameter is hashed with SHA256
/// This method uses keySize and blockSize: member of the Utils class
/// </summary>
public static string AES_encrypt(String input, EncryptOptions option = EncryptOptions.None)
{
    // Encrypt with sha256 if the option is selected
    if (option == EncryptOptions.Hash) { input = hash(input); }

    // It's generical code, we won't comment it (not quite useful to change it really)
    var aes = new RijndaelManaged();
    aes.KeySize = keySize;
    aes.BlockSize = blockSize;
    aes.Padding = PaddingMode.PKCS7;
```

You can see that the second parameter is optional. If you pass, in the second parameter, the value EncryptOptions.Hash, your data ‘input’ will be hashed with SHA256.

NO SALT IS ADDED client side, because its useless, the salt is added to the password by the server, just before saving the password, so the server save:

passwordReceived = SHA256(password)

SHA256(passwordReceived + salt) -> saved in the database.

The salt is saved too (in another field WITHOUT encryption)

SHA256

SHA256 is a one-way encryption: it’s a hash function. This means that you can ‘encrypt’ the data but never decrypt it.

What is that for?

For passwords mostly, you want to compare them, but you don’t want them to be written anywhere, it’s not a data it’s a security. You won’t see it anywhere is a screen, you won’t “use” it, “show” it, so you want to check it without saving it.

In case an administrator is reading the database, he shouldn’t have access to all the users’ passwords right?

When you encrypt with SHA256 you are sure that a result can be made by only one data.
Mathematically: For a character combination C, with a SHA256(C)=M, it does not exist a character combination C’ verifying SHA256(C’)=M. And it does not exist a function F verifying F(M)=C.

So you can compare password encrypted in SHA256, if they are equals, you know the user knows the real password.

Salt is added to avoid dictionary attack. If someone hash a lot of ‘usual used’ passwords and save it’s combination after SHA256 hash, he would obtain a dictionary he could reverse to find the ‘not hashed’ password. So salt add a random string to make this dictionary so long to generate it couldn’t be possible to guess any password.

Database

The Account table

Account table is useful to save user information.

You can see that no foreign key is used (both way). It's only because, this way, the setup is easier, it's not yelling any error depending on the order you install tables. But you can add foreign keys of course.

Email and username are unique (keep it this way, it's very important). Except if you know exactly what you are doing but it can take a long time to change the security system. So just keep the constraints email address is unique, and username is unique.

The IP table

This table is very simple, it just indicate which IP is linked to which account, and if the IP has been validated (with IP password or via email).

The Attempts table

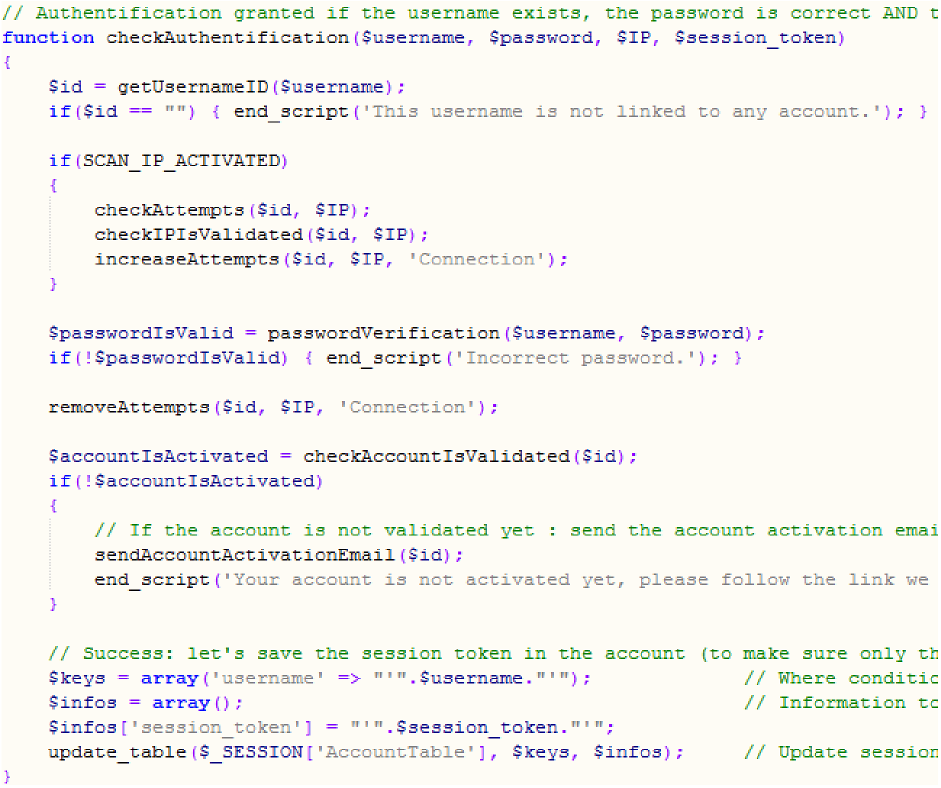
The Attempts table is useful in order to prevent brute force attacks. If someone try a lot of combination of login and password, he has a chance to find a correct combination and usurp someone else account. To prevent this attack, every time someone tries to connect to do an action, an attempt is created (you can specify different actions names to be block, or you could just leave always the same action type for all actions, like “global”).

If the action is performed because all the verifications has been made and are valid, all the actions’ attempts are removed for this account and this IP (remember to remove attempts when you create some).

However if the action can’t be performed because of a lack of a data or an incorrect one, attempts are not removed, and next time the attempts’ number will be incremented.

As soon as the attempt’s number is too high, the action is blocked for this account and this IP.

You can see an example in the checkAuthentification PHP function (in Function.php):



```
// Authentification granted if the username exists, the password is correct AND t
function checkAuthentification($username, $password, $IP, $session_token)
{
    $id = getUsernameID($username);
    if($id == "") { end_script('This username is not linked to any account.'); }

    if(SCAN_IP_ACTIVATED)
    {
        checkAttempts($id, $IP);
        checkIPIsValidated($id, $IP);
        increaseAttempts($id, $IP, 'Connection');
    }

    $passwordIsValid = passwordVerification($username, $password);
    if(!$passwordIsValid) { end_script('Incorrect password.'); }

    removeAttempts($id, $IP, 'Connection');

    $accountIsActivated = checkAccountIsValidated($id);
    if(!$accountIsActivated)
    {
        // If the account is not validated yet : send the account activation emai
        sendAccountActivationEmail($id);
        end_script('Your account is not activated yet, please follow the link we
    }

    // Success: let's save the session token in the account (to make sure only th
    $keys = array('username' => "'".$username."'");             // Where conditic
    $infos = array();                                            // Information tc
    $infos['session_token'] = "'".$session_token."'";
    update_table($_SESSION['AccountTable'], $keys, $infos);     // Update session
}
```

Notice the order:

- Check if attempts for this action are not too high (if so -> error, you are blocked, contact administrators)
- Increase action attempts (the function creates a new attempt line if no one exists)
- Perform all your data verifications
- Remove attempts for this (account, IP, actionName)

Conclusion

Thank you again for your purchase, please rate our asset on the Asset Store, it's a great help for us ☺

If you have any question, a bug report, any improvement to suggest, or you need any help, feel free to contact us at:
redeagleteamcontact@gmail.com

(Our website is just a domain for now but we are working on it, however we prefer to focus our work on the asset than our website, we have full time jobs so it's hard to take into account all users suggestions, we do our best ☺

The RedEagle Team