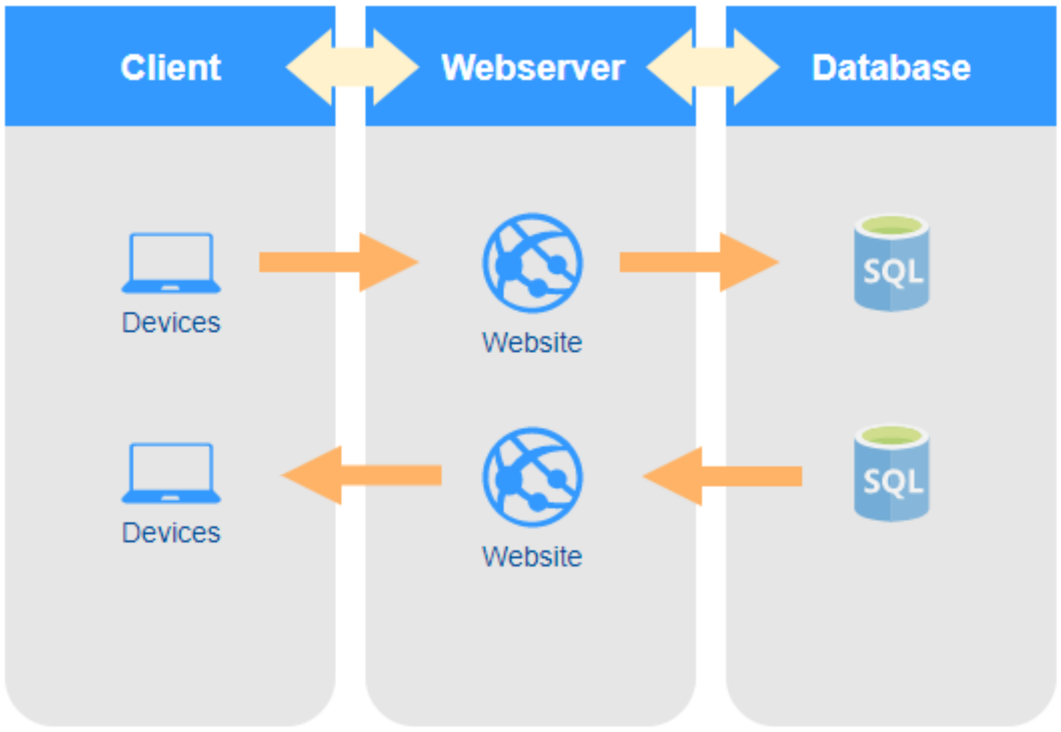
CIPHRA Implementation guide.

This guide describes how to implement CIPHRA in an existing system, and also provides tips on how to secure the registered clients' passwords.

Logical implementation of CIPHRA

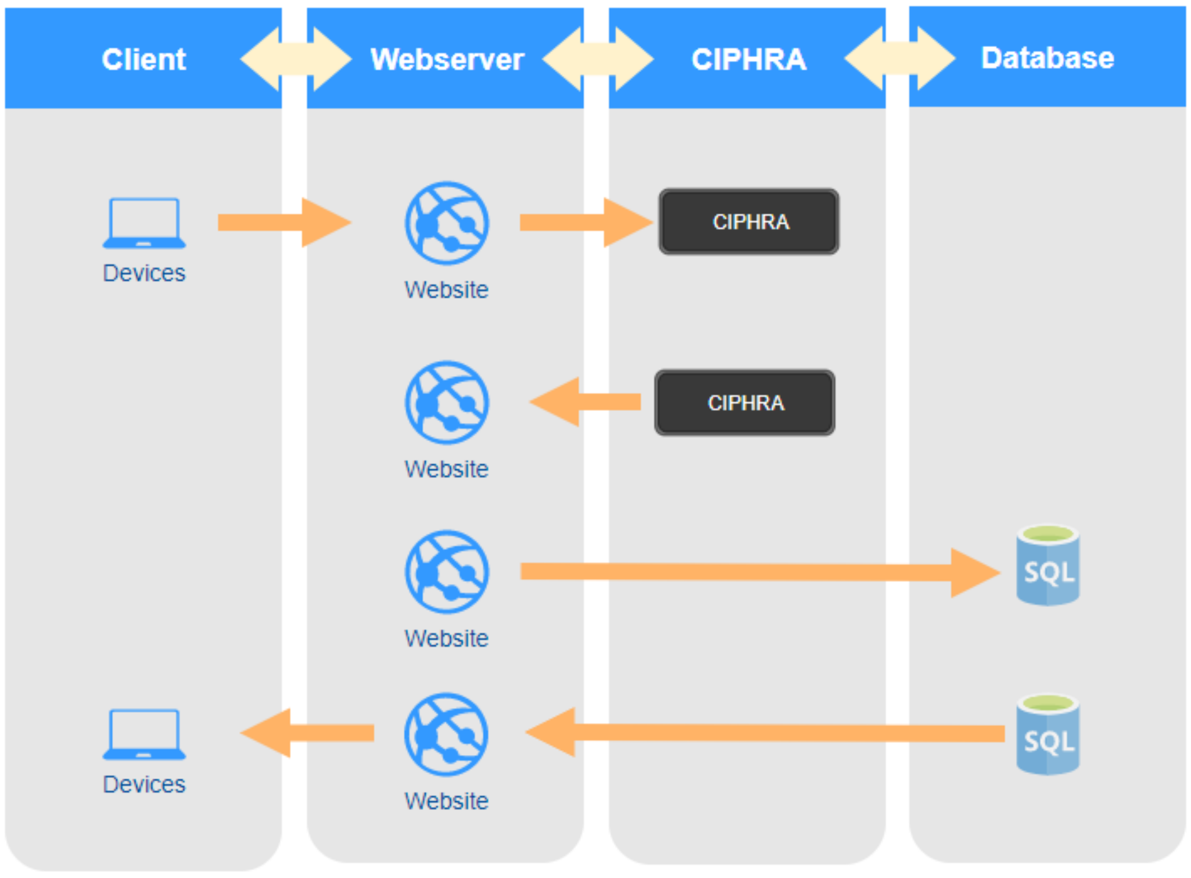
To understand where and how to implement the CIPHRA we need to take a look at the logic.
First, let's take a look at a common implementation of a username and password system. The image describes a simple webservice but can be any type of application.



In above illustration we can see that the client is sending a login or registration request to a webserver. The webserver looks up or registers credentials within a database. The database sends its result to the webserver which in turn validates the response and grants or denies access to the client.

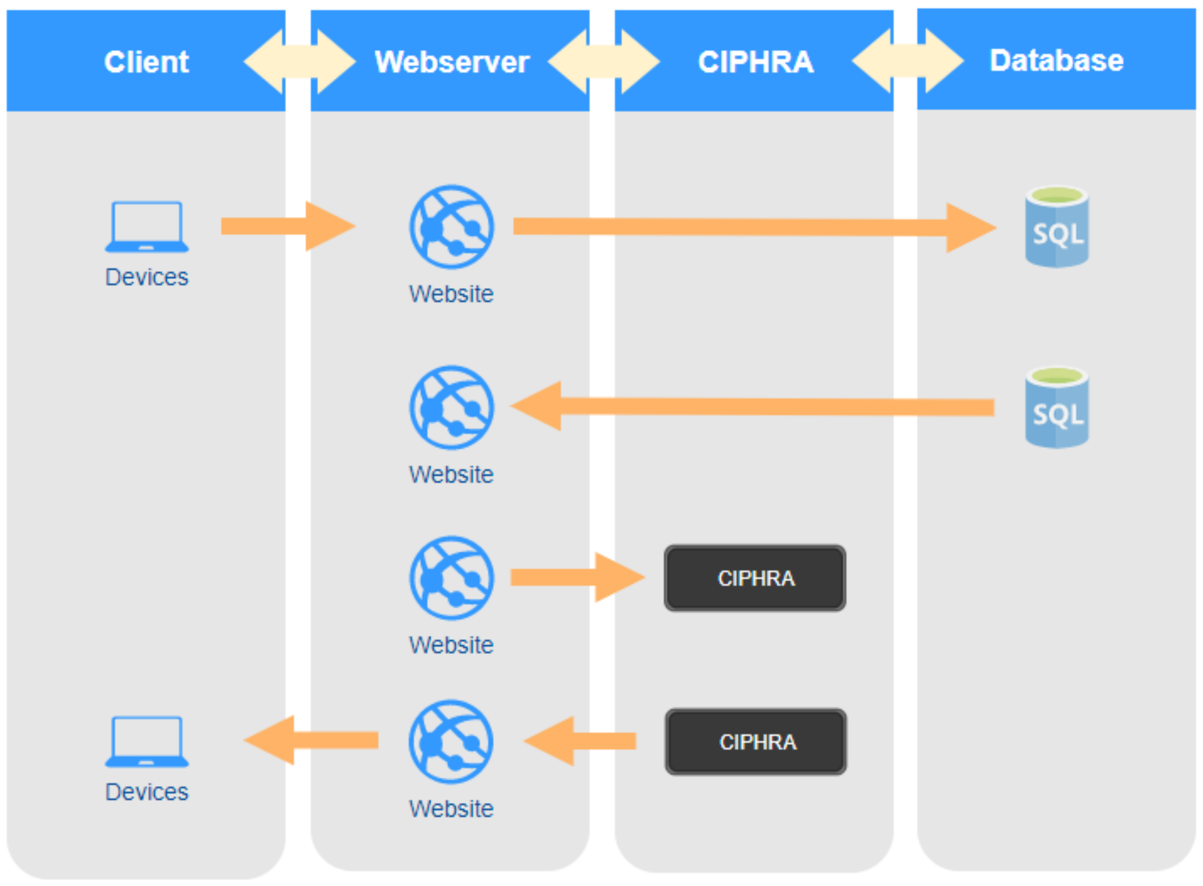
Now, let’s look on how to how this procedure works once we have implemented the CIPHRA into our system. To understand how to work with the CIPHRA we need to divide the site into 2 processes. We can call them registration and validation processes.

Registration process



As in the first image the client sends a request to the webserver. In this scenario the user is registering a new account. Instead of directly interact with a database the webserver now passes on the password from the client to the CIPHRA. The CIPHRA processes the password with its unique PUF algorithm and sends back the result to the webserver. The webserver can now save the registration information to the database using the CIPHRA response hash instead of the user’s password. The webserver gets a response from the SQL server indicating that all has been saved ok and can now return the result to the client.

Validation Process

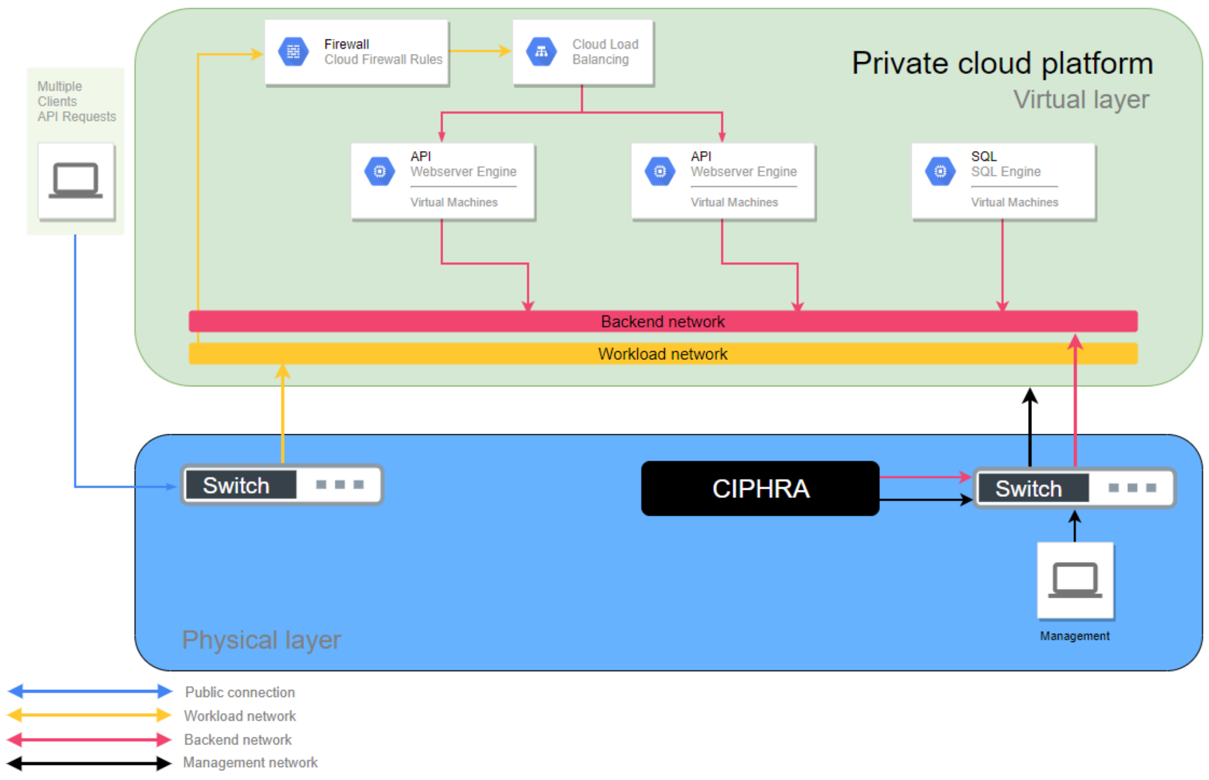


In this scenario the user already has an account and is trying to login. This time when the webserver receives client login request it talks to the SQL server and asks for the account password hash and salt. The SQL server sends the information back to the webserver. The webserver now passes on the password from the client together with the salt from the database to the CIPHRA. The CIPHRA processes the password with its unique PUF algorithm and sends back the result to the webserver. The webserver can now continue to validate the user by checking whether the CIPHRA's response matches the password hash retrieved from the database. After validation is complete the webserver can now send its response to the client.

Now let's move on to a physical implementation of above solution.

Physical implementation of CIPHRA

Today most datacenters and offices use some form of virtualization platform to handle their servers. We are now going to look at how to apply a CIPHRA appliance to an environment like this.



The networks in this figure are virtual networks except for the Public connection. These networks can be VLAN, VxLan or similar type of virtual networks. As in the example, CIPHRA needs to be connected to the backend network so it can respond to the requests from the Webserver/API. The CIPHRA only supports VLAN as a virtual network. If the backend network consists of any other type of technique a conversion must be done to VLAN. CIPHRA is also connected to a management network. From this network you can manage your CIPHRA. Please consult the CIPHRA manual on how to configure it correctly.

Multiple CIPHRA.

If you have multiple CIPHRA appliances in your environment, it is recommended to load balance the traffic to them. For this to be achieved the CIPHRA appliances need to be configured with the same Device key. To load balance, you can either use a hardware/software system for load balancing or modify your own code to load balance among the CIPHRA appliances.

Tips

1. Always use a secure connection between your client and the API/Webserver
2. Try to pass on the client's credentials to the CIPHRA as soon as possible.
3. Always remove the user's password from your variables as soon as you got response from CIPHRA.
4. Put the CIPHRA API on separate VLAN from management.

API endpoints

To communicate with the CIPHRA you utilize a REST API. The response from the CIPHRA is in JSON format. There are two API endpoints available for use.

GET /health

You can use this endpoint to check the health of the CIPHRA. The endpoint can also be used to validate the CIPHRA in a load balancing scenario. It returns one property named HEALTH which can contain 3 different values: OK, DEGRADED and FAILED.

Input example
GET /health
input property <none>
Response example:
{"HEALTH":"OK"}

POST /mkhash

Use this endpoint to hash passwords. The endpoint takes one mandatory input property named password and one optional named salt. When using the mandatory option alone the response comes with two properties: HASH and SALT. If you add the optional property to the input, you will only get property HASH in return.

example using password property only as input
POST /mkhash
input property password=1234
Response example:
{"HASH":"5e15e49106885550957060ff55e912d900e22834bc5f7546ebaa1d3ca9a1129b","SALT":"0004bf2da6f7693dd65e73cc45f4b7f722c6454d04058eeec352af33cd5e7b9e"}

example using password and salt as input
POST /mkhash
*input property password=1234&salt=*0004bf2da6f7693dd65e73cc45f4b7f722c6454d04058eeec352af33cd5e7b9e
Response example:
{"HASH":"5e15e49106885550957060ff55e912d900e22834bc5f7546ebaa1d3ca9a1129b"}

Programming example in C#

This programming example is to give you a guideline on how to implement the CIPHRA in your system using the mkhash endpoint. In the example we have two classes for your JSON responses and one class to process our requests.

```
public class mkhashResponse
    {
        public string HASH { get; set; }
        public string SALT { get; set; }
    }
```

```
public class healthResponse
    {
        public string HEALTH { get; set; }
    }
```

```
class CIPHRA
    {
        private string ciphra_url;
        public CIPHRA(string endpoint)
        {
            ciphra_url = endpoint;
        }

        public string Registration(string password)
        {
            string inputData = "password=" + password;
            return sendreq(inputData);
        }

        public string Validation(string password, string salt)
        {
            string inputData = "password=" + password + "&salt=" + salt;
            return sendreq(inputData);
        }

        private string sendreq(string input)
        {
            string resp;
            var req = WebRequest.Create(ciphra_url + "/mkhash");
            req.Method = "POST";
            byte[] byteArray = Encoding.UTF8.GetBytes(input);
            req.ContentLength = byteArray.Length;
            Stream dataStream = req.GetRequestStream();
            dataStream.Write(byteArray, 0, byteArray.Length);
            dataStream.Close();
            var Wresponse = (HttpWebResponse)req.GetResponse();
            using (var sr = new StreamReader(Wresponse.GetResponseStream()))
            {
                resp = sr.ReadToEnd();
            }
            return resp;
        }
    }
```

The next step is now how to use these classes. The first of this is using it during the registration process. Once you have received the password from the client you can now pass it on to the CIPHRA for hashing. In this example the user has sent you the password 1234

Registration process

```
//Instantiate the CIPHRA class and pass on the url to CIPHRA.
CIPHRA CIPHRAdemo = new CIPHRA("https://ciphraurl");

// Call the Registration process where the user password is 1234
// The JSON response is stored in response
string strResponse = CIPHRAdemo.Registration("1234");

//Convert JSON response to object
mkhashResponse resp = JsonConvert.DeserializeObject<mkhashResponse>(strResponse);
```

We now have the result in the object variable *resp*. You can now continue to store the HASH and the SALT in the database.

Validation process

```
//Instantiate the CIPHRA class and pass on the url to CIPHRA.
CIPHRA CIPHRAdemo = new CIPHRA("https://ciphraurl");

// Call the Validation process where the user password is 1234
// The JSON response is stored in response
string strResponse = CIPHRAdemo.Validation("userpassword", "salt");

//Convert JSON response to object
mkhashResponse resp = JsonConvert.DeserializeObject<mkhashResponse>(strResponse);
```

The validation process is very similar to the registration process but this time we also pass on the SALT we stored in database earlier. We utilize the same response class but this time only the HASH will be in the object *resp*. We can now continue the validation process by comparing the stored database hash value to the one found in response from CIPHRA.