ОСНОВНЫЕ ПОДХОДЫ К ПРОЦЕССУ ШИФРОВАНИЯ БАЗ ДАННЫХ НА ОСНОВЕ ПРОЦЕДУРЫ ГЕНЕРИРОВАНИЯ ОДНОРАЗОВЫХ КЛЮЧЕЙ

Лукманов Ильнар Ильгизович

ИСПОЛЬЗУЕМЫЕ СОКРАЩЕНИЯ И ОПРЕДЕЛЕНИЯ

SHA - Secure Hashing Algorithm
ГОК - Генератор одноразовых ключей
eToken - electronic token
ajax – асинхронная передача данных
jQuery – вспомогательная библиотека над java script
Соль[1] - уникальное слово, гарантирующее уникальность данных при шифровании

Основные подходы к процессу шифрования баз данных на основе процедуры генерирования одноразовых ключей

Шифрование данных чрезвычайно важно для защиты конфиденциальной информации. В этой статье приводится краткий обзор существующих подходов к процессу шифрования конфиденциальной информации.

Так как возможности интернета продолжают расти, большая часть отечественных предприятий осуществляют операции в темпе реального времени. Среди этого набора наиболее важных операций особо ответственными являются: интернет банк, онлайн оплата, электронные письма, обмен частными и служебными сообщениями и др., которые предусматривают обмен конфиденциальными данными и информацией сторонних лиц. Если эти данные окажутся в распоряжении злоумышленника, то это может нанести вред не только отдельному пользователю, но и всей компании, и ее онлайн бизнес - системе. Чтобы ликвидировать и обезопасить в будущем потенциальные угрозы, были реализованы меры по шифрованию сетевого трафика для защиты передачи личных данных на основе криптографических методов. Существуют три основные методы шифрования, используемых в большинстве систем сегодня: хеширование, симметричное и асимметричное шифрование.

1 Методы шифрования

1.1 Симметричное шифрование

При симметричном шифровании обычного текста, он кодируется (шифруется), таким образом, что становится нечитаемым. Это скремблирование данных производится с помощью ключа. Как только данные будут зашифрованы, их можно безопасно передавать на ресивер. У получателя, зашифрованные данные декодируются с помощью того же ключа, который использовался для кодирования. Таким образом ясно что ключ является наиболее важной частью симметричного шифрования. Он должен быть скрыт от посторонних, так как каждый у кого есть к нему доступ может расшифровать его. Вот почему этот тип шифрования также известен как "секретный ключ". В современных системах, ключ обычно представляет собой строку данных, которые получены в виде «надежного пароля», или из совершенно случайного источника. Он используется в процессе симметричного шифрования программного обеспечения, которое применяет его, чтобы засекретить входные данные. Скремблирование данных достигается с помощью симметричного алгоритма шифрования, такого как: стандарт шифрования данных (DES), расширенный стандарт шифрования (AES), или международный алгоритм шифрования данных (IDEA).

.2 Ассиметричное шифрование

Асимметричный ключ шифрования работает аналогично симметричному ключу, данный вид шифрования использует ключ для кодирования передаваемых сообщений. Однако, вместо того, чтобы использовать тот же ключ, для расшифровки этого сообщения он использует другой. Ключ, используемый для кодирования доступен любому и всем пользователям сети. Как таковой он известен как «общественный» ключ. С другой стороны, ключ, используемый для расшифровки, хранится в тайне, и предназначен для использования в частном порядке самим пользователем. Следовательно, он известен как «частный» ключ. Асимметричное шифрование также известно, как шифрование с открытым ключом. Поскольку, при таком способе, секретный ключ, необходимый для расшифровки сообщения не должен передаваться каждый раз, и он обычно известен только пользователю (приемнику), вероятность того, что злоумышленник сможет расшифровать сообщение значительно ниже. Diffie-Hellman и RSA являются примерами алгоритмов, использующих шифрование с открытым ключом.

.3 Хэширование

Методика хеширования использует алгоритм, известный как хэш-функция для генерации специальной строки из приведенных данных, известных как хэш. Этот хэш имеет следующие свойства:

- одни и те же данные всегда производит тот же самый хэш.
- невозможно, генерировать исходные данные из хэша в одиночку.

- Нецелесообразно пробовать разные комбинации входных данных, чтобы попытаться генерировать тот же самый хэш.

Таким образом, основное различие между хэшированием и двумя другими формами шифрования данных заключается в том, что, как только данные зашифрованы (хешированы), они не могут быть получены обратно в первозданном виде (расшифрованы). Этот факт гарантирует, что даже если хакер получает на руки хэш, это будет бесполезно для него, так как он не сможет расшифровать содержимое сообщения. Message Digest 5 (MD5) и Secure Hashing Algorithm (SHA) являются двумя широко используемыми алгоритмами хеширования.

2 Выбор метода шифрования и пример его работы, для шифрования данных полученных от ГОК

2.1 Выбора метода шифрования

Проанализировав существующие методы шифрования данных, для шифрования полученной от устройства генератора одноразовых ключей информаций на стороне сервера (backend) было принято использовать SHA (а именно SHA-256) шифрования с добавлением “соли”[1], к первоначальному паролю.

Метод шифрования SHA 256 был выбран по следующим достоинствам:

- высокая скорость шифрования и практически невозможная расшифровка без ключей;
- минимальный риск появления коллизий (одинаковых образов).

Но, тем не менее, не смотря на высокую степень безопасности использования SHA 256 и добавления уникального слова к первоначальному паролю (“соль”) все же остается риск непосредственного взлома базы данных злоумышленником и перебора возможных комбинаций с hash суммой. Для решения этой проблемы было принято в момент шифрования вместе со словом “соль” использовать дополнительный локальный параметр.

Дополнительный локальный параметр это еще одно слово “соль” в комбинаций “пароль + соль”, только данный параметр будет храниться не в базе данных, а непосредственно в памяти приложения локальной машины – константа системы. В константы системы локальный параметр попадает из конфигурационного файла (только не из базы данных).

2.2 Примеры реализаций методов шифрования

Для демонстраций основных методов шифрования рассмотренных в главе 1 создадим web сервис реализующий алгоритмы шифрования принимающий запросы по принципу REST api. После создадим front обертку с небольшим UI интерфейсом на стороне клиента для отправки наших данный на сервер и вывод полученных шифров на экран клиенту.

В качестве конечной точки (end point) куда будут отправляться запросы, используем тестовый домен **ilnarlx5.bget.ru/service/hash/** находящийся на хостинг провайдере компаний Beget. Серверная часть осуществляющая шифрование данных и принимающая запросы с фронта будет написана интерпретируемом языке гипертекстовой разметки – PHP. В качестве системы управления базами данных будет использоваться MySQL. Фронт будет написан с применением технологий html, css, javaScript + jQuery, а для отправки данных на сервер используем технологию Ajax (асинхронная передача данных на сервер), что позволит передавать и получать данные из backend без перезагрузки страницы.

Определим основные REST api запросы и сведем их в таблицу 1.

ТАБЛИЦА 1.

ОСНОВНЫЕ REST API ЗАПРОСЫ

Название метода	Метод передачи	Конечная точка (URL) ilnarlx5.bget.ru/service/hash/	Входные параметры	Выходные параметры
Симметричное шифрование	POST	symmetric_encryption.php	{password: value}	String: зашифрованный пароль
Ассиметричное шифрование	POST	asymmetric_encryption.php	{password: value}	String: зашифрованный пароль
Хеширование	POST	hash_function.php	{password: value}	String: зашифрованный пароль
SHA + слово соль + локальный параметр	POST	custom_hash.php	{password: value}	String: зашифрованный пароль

2.2.1 Реализация симметричного шифрования

Реализуем симметричное шифрование инструментами и технологией описанной в параграфе 2.2, в качестве библиотеки шифрования воспользуемся строенным модулем в php – HASH, для этого:

оздадим файл symmetric_encryption.php в каталоге сервера /service/hash/ для этого воспользуемся встроенной панелью управления sprutio, предоставленной провайдером Beget, Рисунок 1:

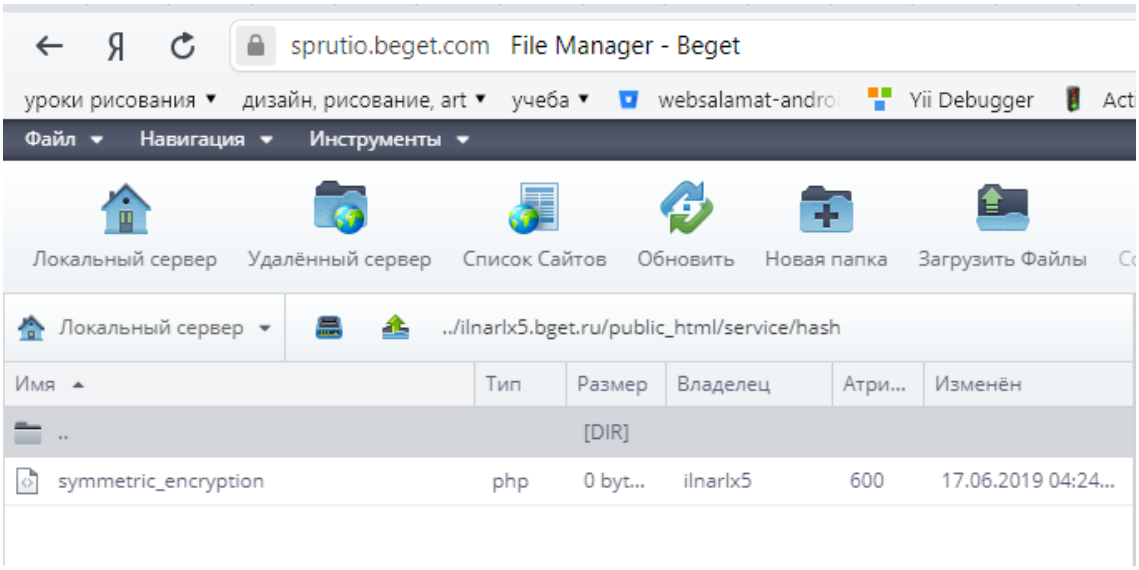
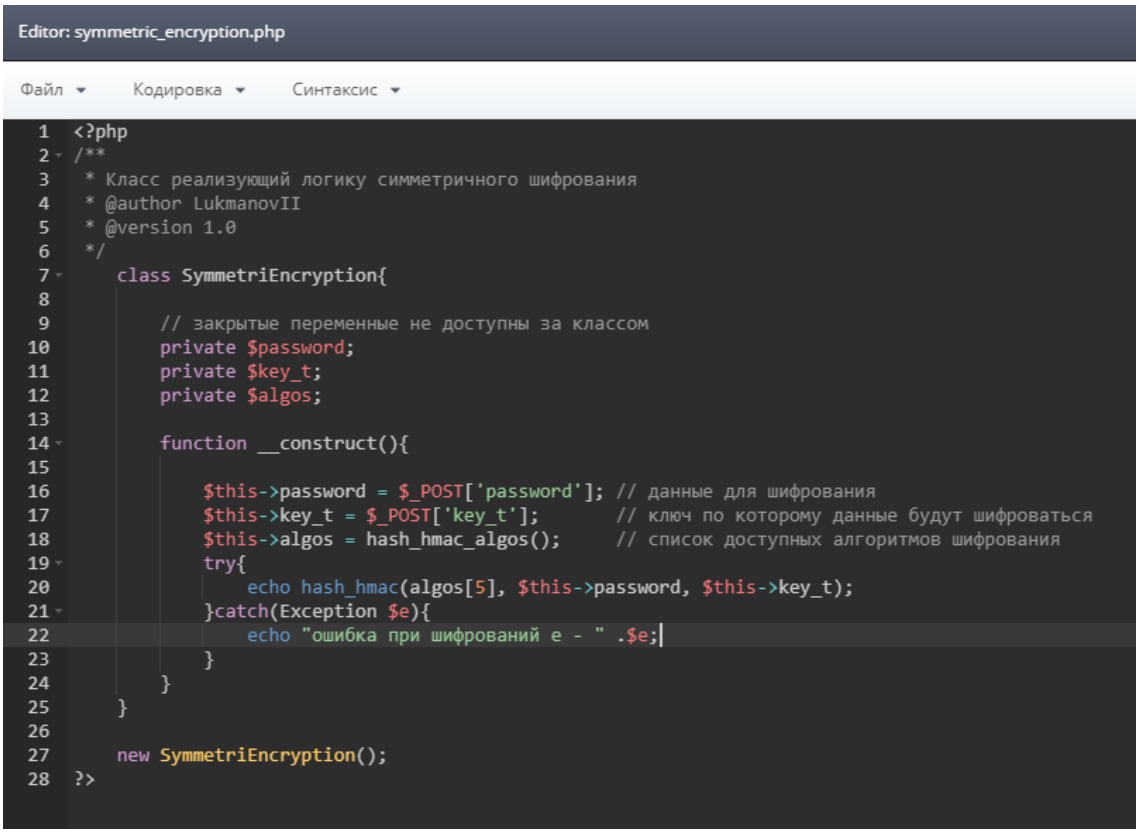


Рисунок 1. Создание файла symmetric_encryption.php

2) В данном файле напишем программный код на языке php, листинг которого представлен ниже Рисунок 2




```php
<?php
/**
 * Класс реализующий логику симметричного шифрования
 * @author LukmanovII
 * @version 1.0
 */
    class SymmetriEncryption{

        // закрытые переменные не доступны за классом
        private $password;
        private $key_t;
        private $algos;

        function __construct(){

            $this->password = $_POST['password']; // данные для шифрования
            $this->key_t = $_POST['key_t'];       // ключ по которому данные будут шифроваться
            $this->algos = hash_hmac_algos();     // список доступных алгоритмов шифрования
            try{
                echo hash_hmac(algos[5], $this->password, $this->key_t);
            }catch(Exception $e){
                echo "ошибка при шифрований е - " .$e;
            }
        }
    }

    new SymmetriEncryption();
?>
```

Класс на языке php реализующий логику симметричного шифрования
Рисунок 2. Листинг класса SymmetricEncrypting на языке php

Как видно из листинга при написаний исходного код был применен подход Обьектно ориентировочного программирования. На строчках 7-13 создается класс **SymmetricEncrypting** и задаются закрытые переменны: **password** – переменная будет служить для хранения переданного пароля с фронта, **key_t** – переменная для

хранения ключа по которому будет шифроваться пароль, **algos** – ассоциативный массив для хранения доступных методов шифрования библиотеки HASH. Строчка 14 (далее - Стр. [номер строчки]) создается конструктор класса внутри, которого будет происходить получения данных с фронта, шифрование данных и отправка полученного значения обратно. Стр. [16-17] инициализация переменных путем записи в них значений из глобального массива POST, стр 18 получение доступных методов шифрования библиотеки HASH путем вызова функций **hash_hmac_algos().** Стр.[20] путем вызова функций **hash_hmac**() производиться шифрование данных и одновременная отправка полученного результат на фронт с помощью функций **echo**(). Более подробно о встроенных функциях HASH можно узнать из официальной документации: ***https://www.php.net/manual/ru/book.hash.php***

3) Для проверки работоспособности функций напишем клиентскую часть сервиса (front). Для создания UI интерфейса воспользуемся библиотекой bootstrap.

3.1) Создадим файл index.php. Данный файл будет являться точкой работы с сервисом.

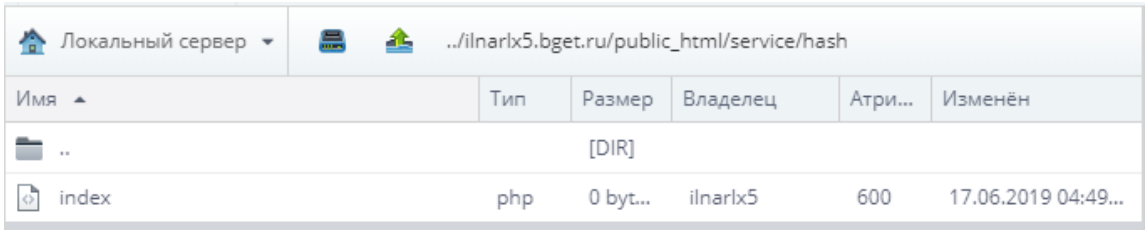


Рисунок 3. Создание файла index.php

3.2) Напишем каркас нашего сервиса на стороне фронта:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8" />
      <title>Шифрование данных</title>
      <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
      <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0i
      <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfR
      <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-UO2eT0CpHqdSJQ6hJty5KV
      <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoI
   </head>
   <body>
      <div class="container">
         <div class="row">
            <div class="col">
               <blockquote class="blockquote">
                  <p class="mb-0 h4">Web сервис шифрования данных</p>
                  <p class="h6">Данный сервис носит ознакомительный характер, <br>
                     не рекомендуется использовать его в практических целях
                  </p>
                  <br>
                  <p class = "h6">
                     <em>
                        Авторы сервиса:
                  <ul>
                  <li class = "h6">Лукманов Ильнар Ильгизович - студент УГАТУ</li>
                  <li class = "h6">Жернаков Сергей Владимирович -  преподаватель высшей школы, ученый. Доктор технических наук (2006),
                  </ul>
                  </em>
                  </p>
```

Рисунок 4. Html каркас сервиса

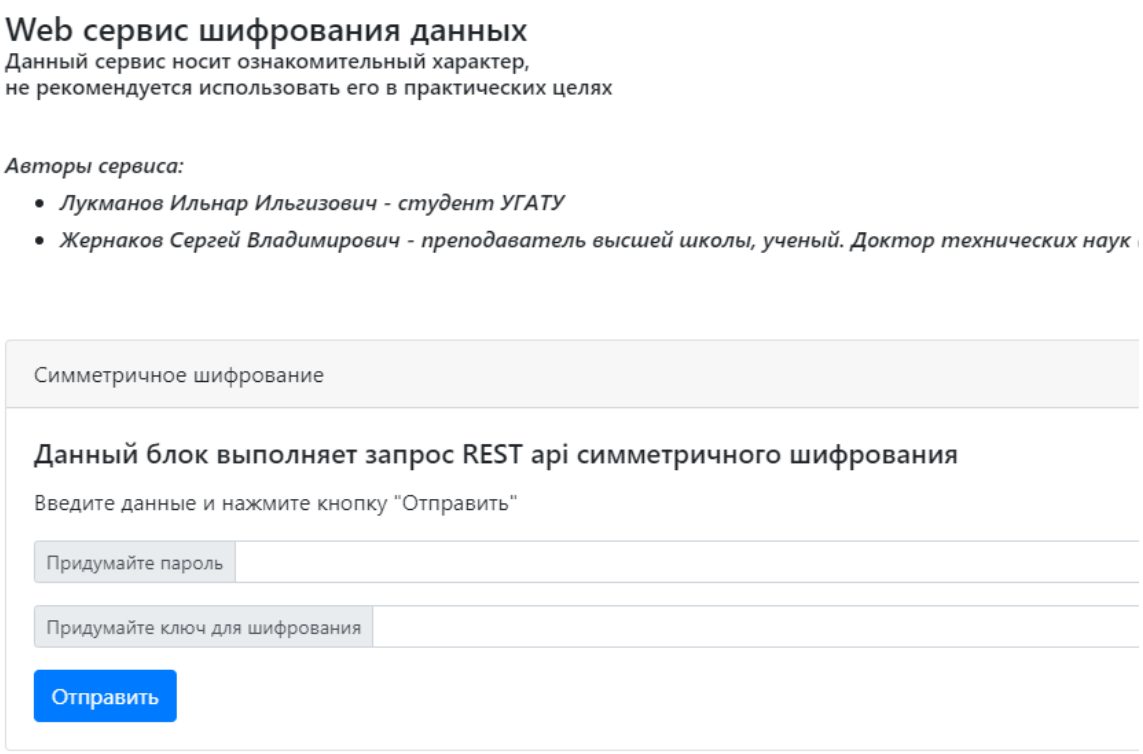


Рисунок 6. UI внешний вид сервиса

3.3) Напишем программу на языке java script с использованием вспомогательной библиотеки обертки над java script – jQuery.

Editor: main.js

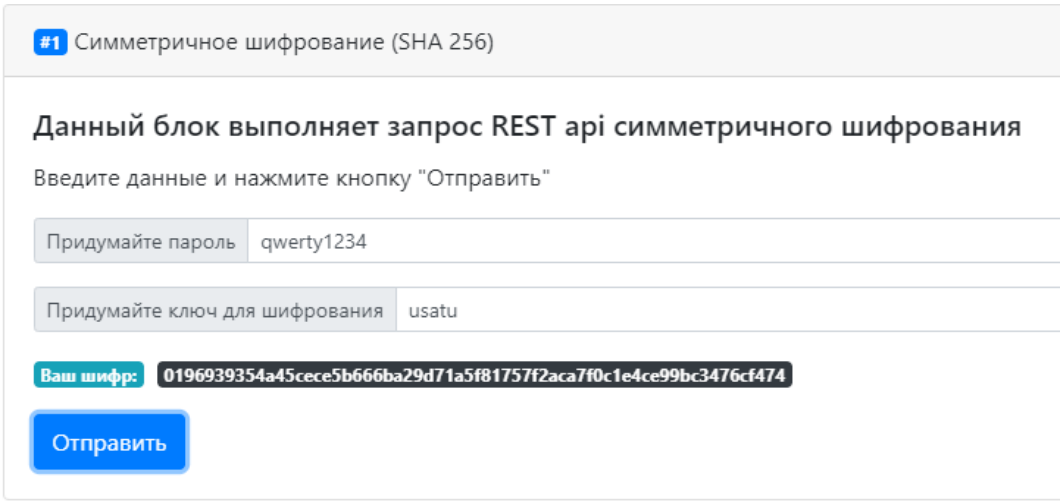
Файл Кодировка Синтаксис

```
1  jQuery(document).ready(function($) {
2
3      $('#getSymmEncBut').on('click', function(){
4          $.ajax({
5              url: "symmetric_encryption.php",
6              type: "POST",
7              data: {password: $('#sycEnctPass').val(), key_t: $('#sycEnctKey').val()},
8              success: function(data){
9                  $('#resultSyncEnc').text(data);
10             }
11         });
12     });
13
```

Рисунок 7. Исходный код java script файлы main.js

Основная задача данного программного кода оправить ajax запрос на сервер когда клиент нажимает кнопку “отправить” в блоке симметричное шифрование. Для отправки используется технология асинхронного запроса Ajax. Стр.[3] привязка к событию нажатия кнопки. Стр. [4-10] Отправка запроса на сервер с помощью обьекта $.ajax библиотеки jQuery, и вывод полученного результат в консоль браузера.

3.4) Протестируем наш REST, отправив запрос на сервер



REST запрос http://ilnarlx5.bget.ru/service/hash/symmetric_encryption.php
Рисунок 8. Значение, полученное от сервера при запросе шифрования.

2.2.2 Реализация шифрования SHA с применением слова соль и локального параметра

Данная логика шифрования данных была озвучена в главе 2.1 принцип ее реализаций как сервиса, схож с тем, что написано в главе 2.2.1, единственным отличием является программный код шифрования на стороне сервера, рассмотрим его более подробно. На рисунке 9 представлен листинг файла **custom_hash.php**

```php
class CustomEncryption{

    // закрытые переменные не доступны за классом
    private $password;
    private $sol;
    private $localSol = "SpRiNg";
    private $resultEnc;
    private $key_t;
    private $algos = array();

    function __construct(){

        $this->password = $_POST['password']; // данные для шифрования
        $this->sol = $_POST['sol'];

        $this->resultEnc = $this->password.$this->sol.$this->localSol;
        $this->key_t = $this->resultEnc.$this->sol;

        $this->algos = hash_hmac_algos();     // список доступных алгоритмов шифрования
        try{
            echo hash_hmac('sha256', $this->resultEnc, $this->key_t).'<br><br> Локальный параметр:  '.$this->localSol;
        }catch(Exception $e){
            echo "ошибка при шифрований e - " .$e;
        }
    }
}

new CustomEncryption();
?>
```

шифрования данных SHA 256 + соль + локальный параметр
Рисунок 9. Листинг программы

Для шифрования используется все та же функция hash_hmac(), с единственным отличием, теперь в качестве входного значения используется конкатенированное поле пароля сл словом соль и локальным параметром стр.[22] и в качестве ключа шифрования выступает соль + локальный параметр стр.[23]

В итоге на фронт мы получаем значение следующего вида:

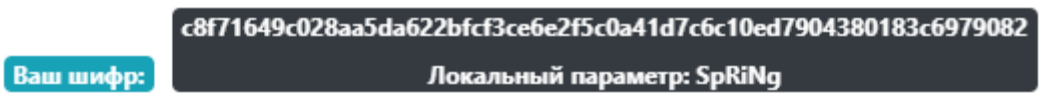


Рисунок 10. Результат REST запроса: http://ilnarlx5.bget.ru/service/hash/custom_hash.php

Заключение

В заключение можно сказать, что эти методы могут быть использованы для эффективного кодирования данных в нечитаемый формат, который может гарантировать, что они останутся безопасными. Большинство современных систем обычно используют комбинацию этих методов шифрования наряду с сильной реализацией алгоритмов для повышения безопасности. В дополнение к безопасности, эти системы также предоставляют множество дополнительных преимуществ, таких как проверка удостоверения пользователя, и обеспечение того, что полученные данные не могут быть подделаны.