Минобрнауки России

федеральное государственное бюджетное образовательное учреждение высшего образования «Санкт-Петербургский государственный технологический институт (технический университет)»

СПбГТИ(ТУ)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Направление подготовки | | 09.03.01 | Информатика и вычислительная техника | | | |
| Направленность программы | |  | Автоматизированные системы обработки информации и управления | | | |
|  | |  |  | | | |
| Факультет | |  | Информационных технологий и управления | | | |
| Кафедра | |  | Систем автоматизированного  проектирования и управления | | | |
| Учебная дисциплина | |  | Интернет-технологии | | | |
| Курс 3 |  | | | Группа | 404 |

**Отчет по контрольной работе № 3**

|  |  |
| --- | --- |
| **Тема:** | **Создание серверной части веб-сайта** |

Студент гр. 404 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. К. Азаров \_

(подпись, дата) (инициалы, фамилия)

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Иванов \_

(подпись, дата) (инициалы, фамилия)

Отметка о зачете \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись преподавателя)

Санкт-Петербург

2023

#### **1 Описание идеи проекта. Формирование функциональных требований к проекту**

В данной контрольной работе осуществляется продолжение развитие идеи проекта, описанной в контрольной работе №1.

**Цель проекта:** разработать интерфейса веб-приложения для шифрования текста.

**Функциональные требования к разрабатываемому веб-приложению:**

* Предоставлять доступ к сайту.
* Шифровать текст алгоритмами: AES, BLOWFISH, CAST, DES.
* Предоставлять администратору возможность смотреть частоту использования каждого алгоритма.

#### **2 Обзор и обоснование выбора архитектуры информационной системы, способа клиент-серверного взаимодействия**

Наиболее известные подходы к архитектуре:

* – Многослойная архитектура (Layered Architecture);
* – Многоуровневая архитектура (Tiered Architecture);
* – Сервис-ориентированная архитектура (Service Oriented Architecture — SOA);
* – Микросервисная архитектура (Microservice Architecture) [1].

Многослойная архитектура работает по принципу разделения ответственностей. Программное обеспечение разделено на слои, лежащие друг на друге, и каждый из них выполняет определенную обязанность. Преимущества: более простая реализация по сравнению с другими подходами; предлагает абстракцию благодаря разделению ответственностей между уровнями; изолирование защищает одни слои от изменений других; повышает управляемость программного обеспечения за счет слабой связанности. Недостатки: не предлагает большой масштабируемости; программное обеспечение, созданное с таким подходом, будет иметь монолитную структуру, усложняющую внесение модификаций; данные должны проходить по каждому слою, даже если нет необходимости передавать их с определенных слоев.

Многоуровневая архитектура разделяет комплекс программного обеспечения на уровни по принципу взаимодействия “клиент-сервер”. Архитектура может иметь один, два и больше уровней, разделяющих ответственности между поставщиком данных и потребителем. Этот подход использует шаблон Request Response для связи между уровнями. В отличие от многослойной архитектуры, он предлагает масштабируемость, которая может быть как горизонтальной (масштабирование сети с помощью высокопроизводительных узлов), так и вертикальной (масштабирование каждого узла путем повышения его производительности).

Сервис-ориентированная архитектура состоит из компонентов и приложений, которые связываются друг с другом с помощью четко определенных сервисов. Клиент отправляет запрос с использованием стандартного протокола и формата данных по сети. Этот запрос обрабатывается ESB (enterprise service bus — сервисная шина предприятия), которая считается сердцем сервис-ориентированной архитектуры и отвечает за оркестровку и маршрутизацию. С помощью сервисного репозитория ESB направляет запрос в специальный сервис, который может взаимодействовать с другими сервисами и базами данных, чтобы составить полезную нагрузку (данные) ответа.

Mикросервисная архитектура разрабатывается как набор небольших сервисов, каждый из которых работает в собственном процессе и связывается с легковесными механизмами, обычно API для HTTP-ресурса. Преимущества:

предлагает слабую связанность благодаря высокой степени изоляции; повышает модульность; cбой в одном сервисе не затронет всю систему, поскольку они изолированы; предлагает высокую гибкость и масштабируемость; простота модификации может ускорить итерации; позволяет реализовать улучшенную систему обработки ошибок; решает проблемы с потоками данных, которые бывают у многослойной архитектуры. Недостатки: повышенный риск сбоя при обмене данными между сервисами; большим количеством сервисов трудно управлять; требует решения таких проблем, как задержки в сети, балансировка нагрузки и прочих трудностей, свойственных распределенной архитектуре; нуждается в комплексном тестировании в распределенной среде; на реализацию потребуется гораздо больше времени.

Для реализации проекта выбрана трёхуровневая архитектура приложения, представленная на рисунке 1.

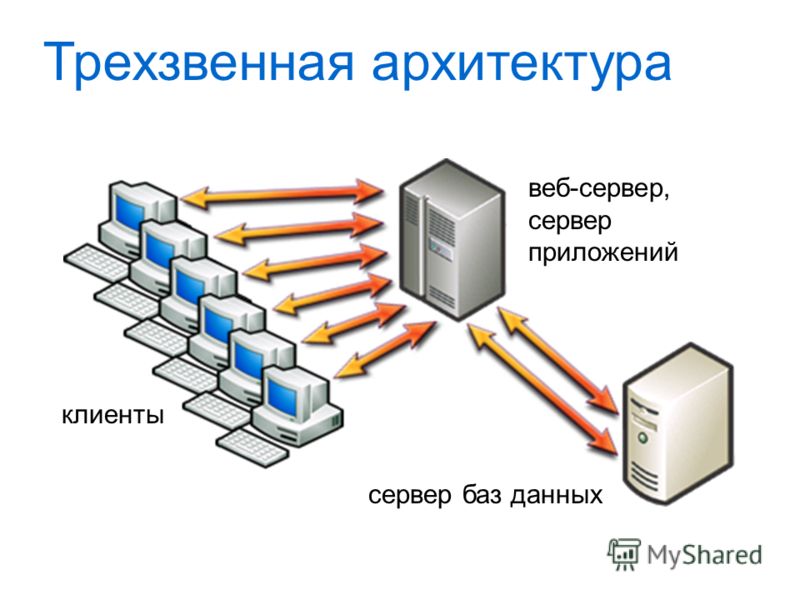


Рисунок 1 – Схема трехуровневой архитектура

Расчёт и визуализация результата будет производиться на стороне клиента, так как cхема трехуровневой архитектуры результатов на стороне сервера вносит дополнительную задержку ответа на действия пользователя.

Хранение данных будет осуществляться на стороне севера в серверной системе управления базами данных – это снизит требования к клиентскому оборудованию (требуется меньший объем накопителя).

Обработка запросов на получение статических файлов и данных из системы управления базами данных будет производиться на сервере.

Клиент и сервер осуществляют запросы посредством протокола HTTP. В целях обеспечения большей безопасности рекомендуется использовать расширение протокола HTTPS для поддержки шифрования данных.

Далее, необходимо выбрать систему управления базами данных. Для сравнения будут приведены PostgreSQL и MySQL, так как они являются наиболее популярными среди разработчиков.

Таблица 1 – Сравнение систем управления базами данных MySQL и PostgreSQL

|  |  |  |
| --- | --- | --- |
| Критерий | MySQL | PostgreSQL |
| Открытый код | + | + |
| ACID | Частично | + |
| Высокая производительность | + | Только при чтении данных |
| Поддержка NoSQL функции | - | + |
| Хранимые процедуры | - | + |

Для реализации в проекте хранения данных выбрана СУБД PostgreSQL обладающая весомыми преимуществами в рамках, а именно:

– имеет поддержку хранимых процедур, которые повысят скорость выполнения запросов, в отличие от MySQL;

– является более производительным и имеет полную поддержку ACID (atomicity, consistency, isolation, durability) [2].

#### **3 Разработка функциональной структуры системы с учётом клиент-серверной архитектуры. Постановка задачи разработки**

Функциональная структура веб-приложения для шифрования, дешифрования данных и вывода частоты использования алгоритмов администратору представлена на рисунке 2. Функциональная структура представлена в виде модели клиент-сервер. Клиент может отправлять на сервер HTTP запросы Ruquest и получать от сервера HTTP ответы Response.

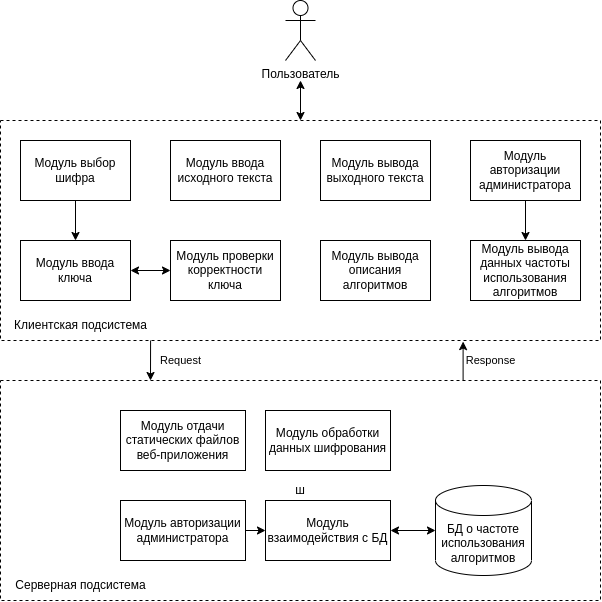


Рисунок 2 – Функциональная структура веб-приложения

#### **4 Разработка UML-диаграммы вариантов использования программы**

На рисунке 3 представлена UML-диаграммы вариантов использования веб-приложения.

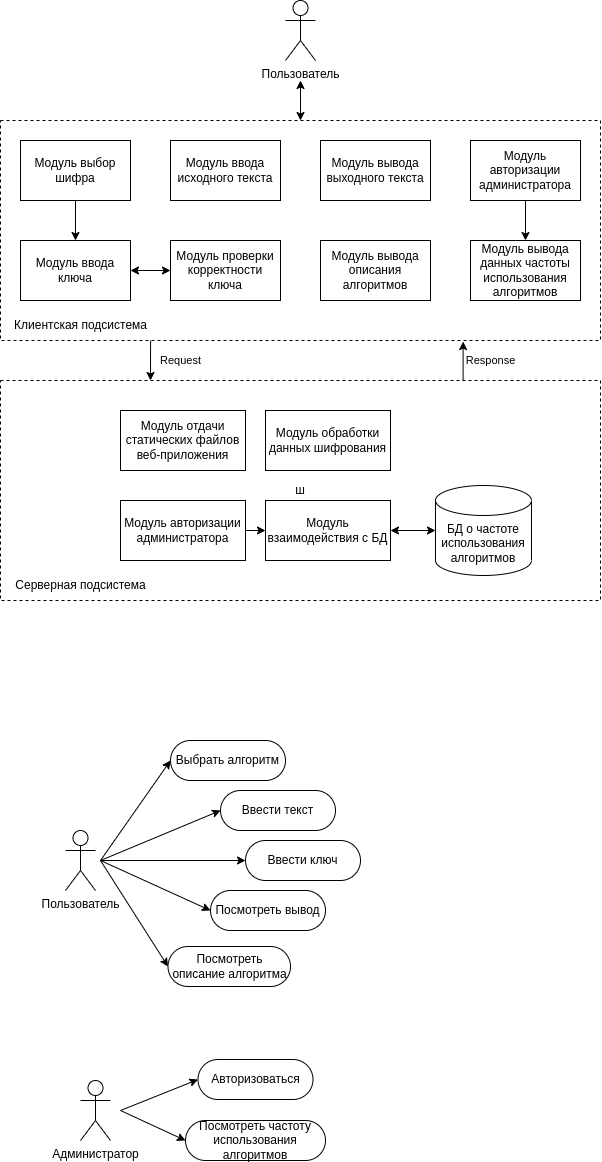


Рисунок 3 – UML-диаграммы вариантов использования веб-приложения

#### **5 Разработка блок-схемы алгоритма решения задачи**

На рисунке 4 представлена блок-схема алгоритма шифрования.

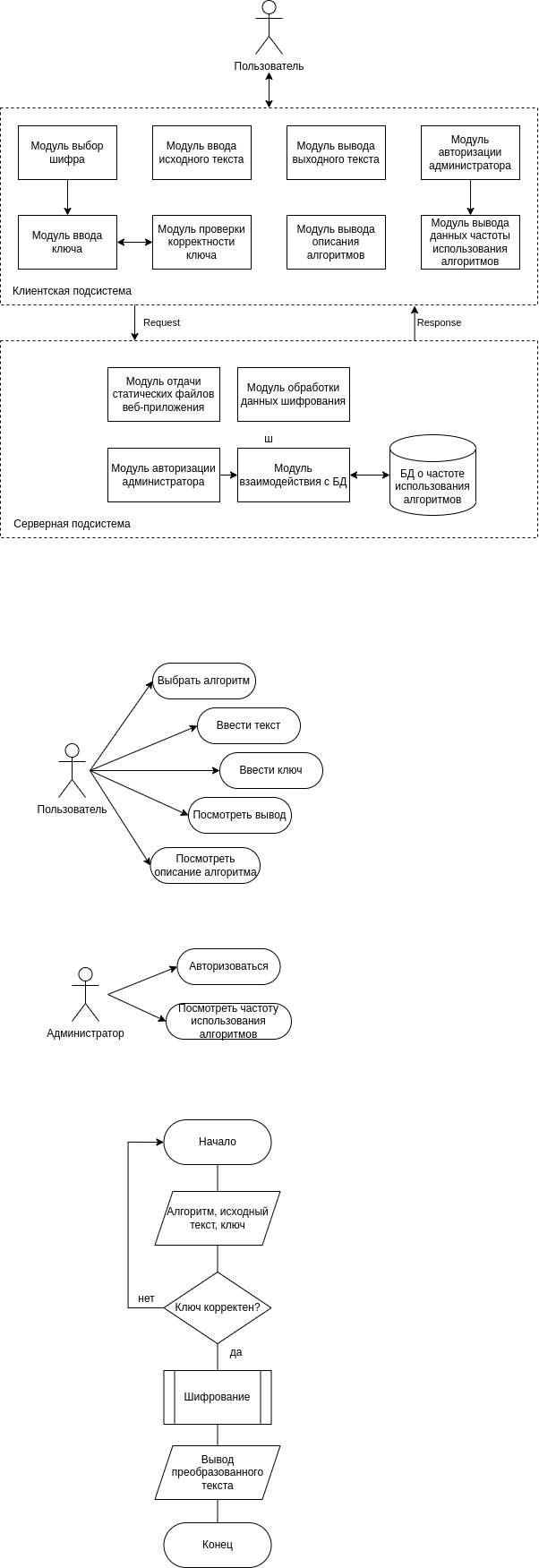


Рисунок 4 – Блок-схема алгоритма загрузки данных

#### **6 Проектирование серверного интерфейса взаимодействия с клиентом и СУБД**

В таблице 2 представлено описание API обращения клиента к серверу по основным запросам.

Таблица 2 – Описание API обращения клиента к серверу по основным запросам

|  |  |  |
| --- | --- | --- |
| Метод HTTP запроса | Относительный путь запроса | Назначение запроса |
| GET | / | Страница шифрования |
| POST | / | Шифрование текста |
| GET | /admin-login | Страница авторизация администратора |
| POST | /admin-login | Отображение таблицы частоты использования алгоритмов |

Для взаимодействия с выбранной СУБД используется Node js express.

[]

[]

#### **7 Разработка базы данных информационной системы**

Структура БД представлена в таблице 4.

Таблица 4 – Структура сущностей

|  |  |  |
| --- | --- | --- |
| Поле(я) | Тип данных | Значение |
| frequency\_algo(algo\_name, frequency) | text, int | Название алгоритма, частота использования |

#### **8 Реализация с использованием JS проекта**

Далее представлен листинг программы, где реализуется обращение к серверу за данными с помощью JS.

[script.js]

window.onload = init

function init()

{

// Изменение шифра

var topMenuChilds = document.getElementById("topmenu").children

for (let el of topMenuChilds)

{

el.addEventListener("click", \_ => changeCipherType(el.textContent))

}

// Генерация ключа

let generateKeyBtn = document.getElementById("generate\_key")

generateKeyBtn.addEventListener("click", setGenerateKeyOnTexBox)

// let keyTextBox = document.getElementById("key\_box")

// keyTextBox.setCustomValidity("Что-то не правильно")

// Кодирование

let cipherBtn = document.getElementById("ciph\_btn")

cipherBtn.addEventListener("click", \_ => ciphText(false))

let decipherBtn = document.getElementById("deciph\_btn")

decipherBtn.addEventListener("click", \_ => ciphText(true))

}

function changeCipherType(name)

{

let header = document.getElementById("cipher\_name")

header.textContent = name

let shownAlghoritm = document.getElementsByClassName("show")[0]

shownAlghoritm.className = "hide"

let alghoritmManual = document.getElementById(name + "-alghoritm")

alghoritmManual.className = "show"

}

function setGenerateKeyOnTexBox()

{

let keyTextBox = document.getElementById("key\_box")

keyTextBox.value = generateKey()

}

function generateKey()

{

let shownAlghoritm = document.getElementById("cipher\_name").textContent

let count = getAlgoKeyLenght(shownAlghoritm)

let min = Math.pow(10, 16);

let max = Math.pow(10, 17)

let val = "";

for(let i = 0; i < 10; ++i)

{

val += Math.floor(Math.random() \* (max - min) + min).toString(36)

}

return val.slice(-count)

}

function getAlgoKeyLenght(name)

{

let count = 0;

switch (name)

{

case "AES":

count = 128 //128/192/256 бит

break

case "BLOWFISH":

count = 32 //от 32 до 448 бит

break

case "CAST":

count = 64 //40-128 бит

break

case "DES":

count = 64 //56 бит + 8 проверочных бит

break

}

let v = count / 8

return count/8

}

function ciphText(isDecipher)

{

console.log(isDecipher)

let keyText = document.getElementById("key\_box")

let typeCiph = document.getElementById("cipher\_name").textContent

if (!checkKeyValidity(keyText.value, typeCiph))

{

keyText.setCustomValidity(`Ключ должен содержать ${getAlgoKeyLenght(typeCiph)} символов`);

keyText.reportValidity()

return

}

let inText = document.getElementById("in\_text").value

let outText = document.getElementById("out\_text")

let req =

{

isDeciph: isDecipher,

type: typeCiph,

text: inText,

key: keyText.value

}

fetch("/",

{

method: 'POST',

headers: { 'Content-Type': 'application/json;charset=utf-8' },

body: JSON.stringify(req)

})

.then(res => res.text())

.then(text => outText.textContent = text)

}

function checkKeyValidity(key, ciphType)

{

return key.length == getAlgoKeyLenght(ciphType)

}

[script.js]

На рисунке 6-7 представлены результаты работы сайт

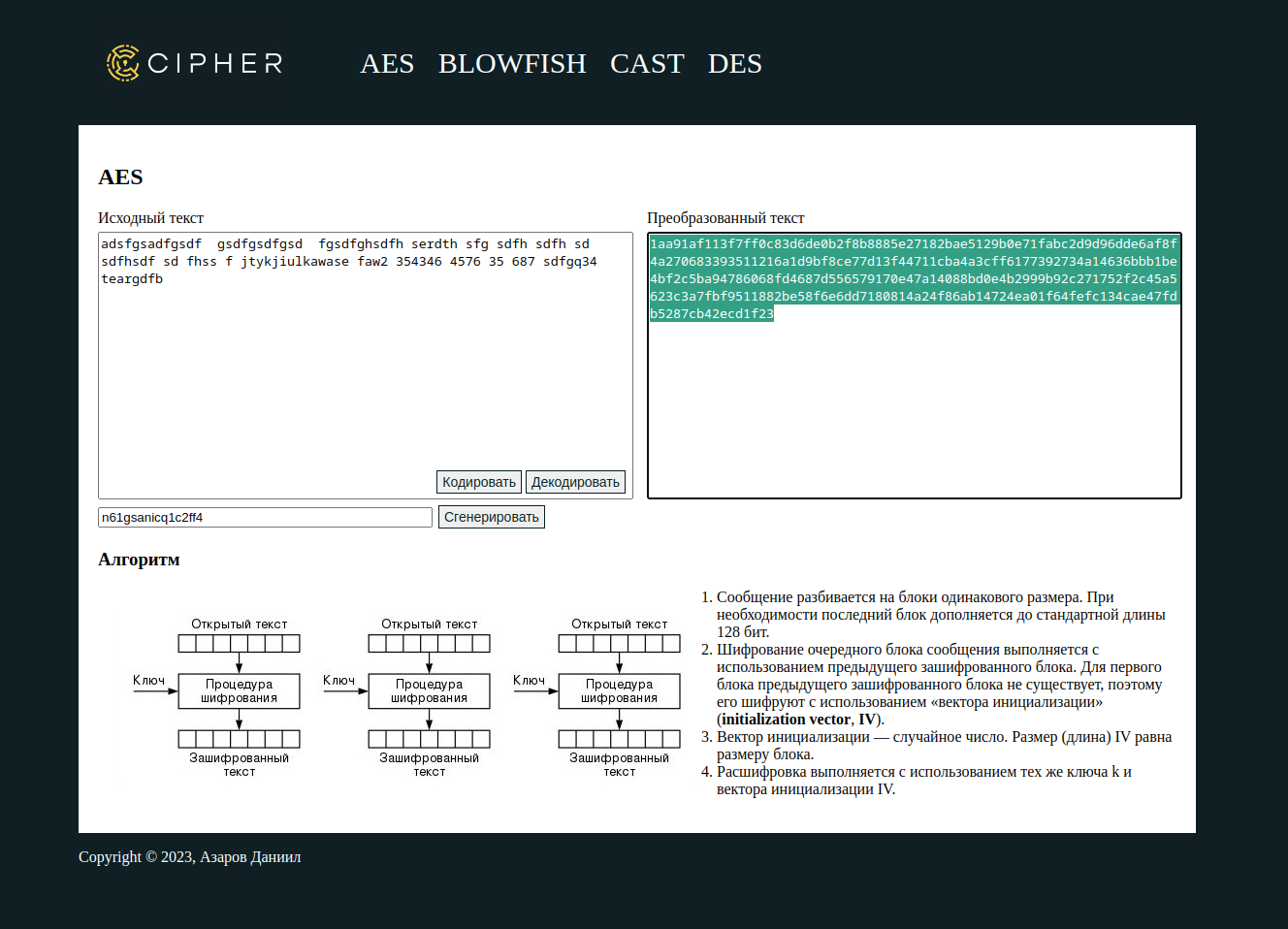


Рисунок 6 – Результат работы сайта

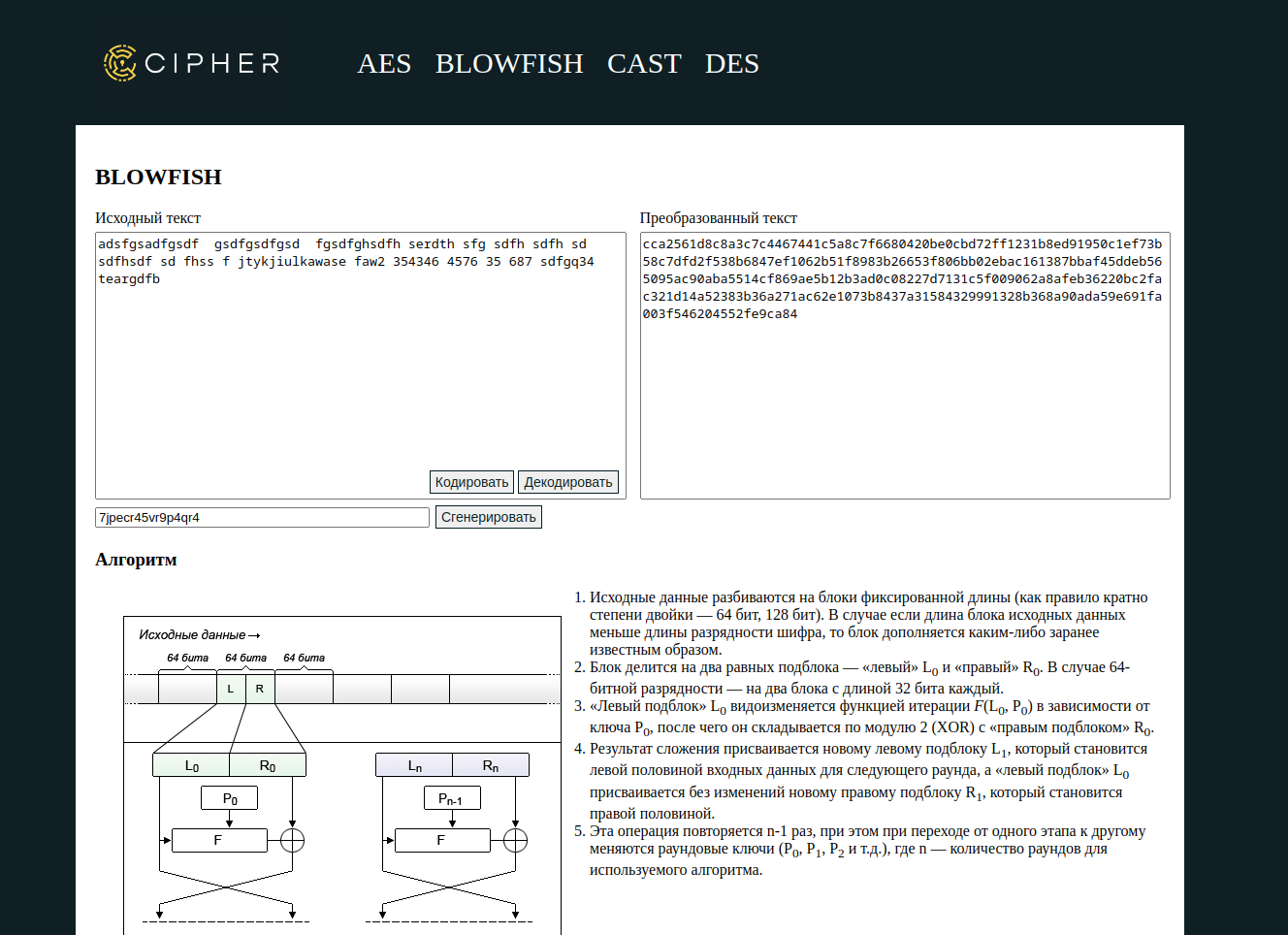


Рисунок 7 – Результат работы сайта

# **СПИСОК ЛИТЕРАТУРЫ**

1 4 типа архитектуры программного обеспечения | by Daria Sidorova | NOP::Nuances of Programming | Medium : сайт. – Хельсинки, 2021 – . – URL: https://medium.com/nuances-of-programming/4-типа-архитектуры-программного-обеспечения-917133174724 (дата обращения: 27.02.22).

2 Что такое SQL? Чем отличаются MySql и PostgreSQL?. – Москва, 2019 – . – URL: https://webhost1.ru/information/blog/2021-05-24-sql-osnovnye-sql-subd.html (дата обращения: 29.03.2023).