Федеральное государственное автономное

образовательное учреждение

Высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт космических и информационных технологий |
| институт |
| Программная инженерия |
| кафедра |

**ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ**

|  |
| --- |
| Аутентификация в Web-ориентированных информационных системах |
| тема |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Преподаватель | |  |  |  | Ю.В. Потылицина |
|  | |  | подпись, дата |  | инициалы, фамилия |
| Студент | КИ21-17/1Б, 032156940 |  |  |  | Н.А. Самарин |
|  | номер группы, зачётной книжки |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2023

**СОДЕРЖАНИЕ**

1 Цель.................................................................................................................... 3

2 Задачи................................................................................................................ 3

3 Теоретические сведения................................................................................... 3

3.1 Описание схемы аутентификации............................................................. 3

3.2 Описание протокола Диффи-Хеллмана.................................................... 4

3.3 Описание информационной системы........................................................ 4

4 Практическая часть.......................................................................................... 4

5 Вывод................................................................................................................. 5

ПРИЛОЖЕНИЕ А................................................................................................. 6

**1 Цель**

- ознакомиться с основами методов аутентификации;

- освоить механизмы аутентификации и авторизации в  
WEB-ориентированных информационных системах;

- ознакомится с базовыми моделями доступа в задачах защиты от  
несанкционированного доступа (НСД);

- изучить алгоритм Диффи-Хеллмана и особенности его применения в  
задачах разработки механизмов аутентификации;

- освоить использование хеш-функций в задачах безопасного хранения и  
передачи учётных данных пользователей;

- познакомиться на практике с технологических особенностями  
разработки механизмов аутентификации в web-ориентированных  
информационных системах.

**2 Задачи**

Разработать эффективную и безопасную (в заданных условиях действия  
нарушителя) механизм централизованной аутентификации пользователей и  
последующей авторизации в web-ориентированной ИС. Разработать её  
программный прототип, реализующий:

- разработанную Вами схему аутентификации/авторизации;

- одну из базовых моделей доступа (см. теоретические сведения) или  
некую модифицированную модель на её основе;

- минимальный функционал ИС для пользователей (тематика самой ИС  
оставляется на выбор студента).

**3 Теоретические сведения**

**3.1 Описание схемы аутентификации**

Процесс аутентификации и авторизации заключается в следующем:

Установка защищенного канала связи по протоколу Диффи-Хеллмана.

- Получение общего секрета по протоколу Диффи-Хеллмана;

- Шифрование передаваемых по каналу сообщений алгоритмом AES.

Аутентификация

- Запрос по защищённому каналу на аутентификацию по учетным  
данным;

- При успешном запросе авторизация считается пройденной, в cookie  
сохраняется user\_id.

У использования протокола Диффи-Хеллмана есть некоторые недостатки.  
Поскольку он не аутентифицирует ни одну из сторон в передаче, обмен  
ключами Диффи-Хеллмана подвержен атаке посредника. Также он не  
позволяет использовать сертификаты.

**3.2 Описание протокола Диффи-Хеллмана**

Протокол Диффи-Хеллмана (англ. Diffie-Hellman, DH) –  
криптографический протокол, позволяющий двум и более сторонам получить  
общий секретный ключ, используя незащищенный от прослушивания канал  
связи. Полученный ключ (обеими сторонами) в дальнейшем может  
использоваться для шифрования дальнейшего информационного обмена  
(образование защищённого канала) с помощью алгоритмов симметричного  
шифрования, а также для реализации методов аутентификации.

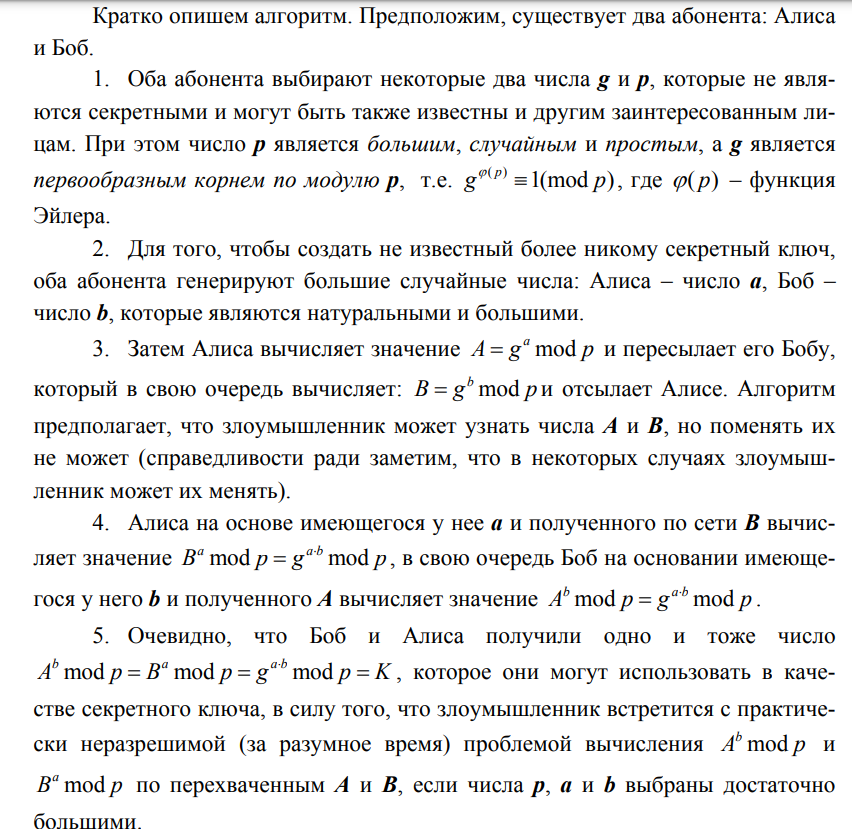


Рисунок 1 – Описание алгоритма протокола Диффи-Хеллмана

**3.3 Описание информационной системы**

Информационная система реализована на языке Python c использованием  
фреймворка Flask, который представляет из себя фреймворк для разработки  
веб-сайтов на языке Python со встроенным веб-сервером. Информационная  
система реализует систему хранения объектов с минимальным функционалом,  
поддерживающим добавление и удаление объектов. Основным объектом  
данных является сущность «собаки». В качестве базы данных используется  
SQLite, для работы с базой используется SQLAlchemy.

**4 Практическая часть**

Информационная система была разработана, листинги программного  
кода представлены в приложении А. Запросы клиента и ответы сервера при  
выполнение операций с информационной системой представлены на рисунках

ниже.



Рисунок 2 – Запросы клиента и ответы сервера при выполнении  
аутентификации



Рисунок 3 – Запросы клиента и ответы сервера при выполнении  
авторизации

**5 Вывод**

В процессе выполнения лабораторной работы были получены навыки  
разработки и реализации механизмов аутентификации и авторизации в  
WEB-ориентированных информационных системах. Изучен алгоритм  
Диффи-Хеллмана. Созданная схема выполняет поставленную задачу защиты  
информации. Программная реализация выполнена согласно нормам и  
стандартам выбранных для использования протоколов и алгоритмов.

**ПРИЛОЖЕНИЕ А**

Листинг А.1 – \_\_init\_\_.py

from flask import Flask  
from flask\_admin import Admin  
from flask\_migrate import Migrate  
from flask\_sqlalchemy import SQLAlchemy  
from sqlalchemy import MetaData  
from config import Config  
convention = {  
 "ix": 'ix\_%(column\_0\_label)s',  
 "uq": "uq\_%(table\_name)s\_%(column\_0\_name)s",  
 "ck": "ck\_%(table\_name)s\_%(constraint\_name)s",  
 "fk": "fk\_%(table\_name)s\_%(column\_0\_name)s\_%(referred\_table\_name)s",  
 "pk": "pk\_%(table\_name)s"  
}  
db = SQLAlchemy(metadata=MetaData(naming\_convention=convention))  
migrate = Migrate()  
admin\_app = Admin(name='KursoAgregator', template\_mode='bootstrap3')  
app = Flask(\_\_name\_\_)  
app.permanent\_session\_lifetime = datetime.timedelta(days=1)  
app.config.from\_object(Config)  
migrate.init\_app(app, db, render\_as\_batch=True)  
db.init\_app(app)  
admin\_app.init\_app(app)  
from app import models, admin, routes, diffi\_helman

Листинг А.2 – admin.py

from flask\_admin.contrib.sqla import ModelView  
from app import admin\_app, db  
from app.models import User, Dog, Role  
class SecurityModelView(ModelView):  
 column\_display\_pk = True  
admin\_app.add\_view(SecurityModelView(User, db.session, endpoint='user'))  
admin\_app.add\_view(SecurityModelView(Dog, db.session, endpoint='dog'))  
admin\_app.add\_view(SecurityModelView(Role, db.session, endpoint='role'))

Листинг А.3 – diffi\_helman.py

import os  
from base64 import b64encode, b64decode  
import sympy  
import random  
from math import gcd  
from cryptography.exceptions import InvalidTag  
from cryptography.hazmat.backends import default\_backend  
from cryptography.hazmat.primitives import hashes  
from cryptography.hazmat.primitives.ciphers import Cipher, modes  
from cryptography.hazmat.primitives.ciphers.algorithms import AES  
def generate\_coprime\_with\_p(p):  
 while True:  
 s = random.randint(2, p - 1)  
 print(s)  
 if gcd(s, p) == 1:  
 return s  
def get\_g\_p():  
 p = sympy.randprime(10 \*\* 15, 10 \*\* 16)  
 g = generate\_coprime\_with\_p(p)

Окончание листинга А.3

return g, p  
def secret\_key\_client():  
 return random.randint(10 \*\* 15, 10 \*\* 16 - 1)  
def get\_shared\_server\_key(A, b, p):  
 return pow(A, b, p)  
def get\_shared\_client\_key(g, b, p):  
 return pow(g, b, p)  
def secret\_key\_server():  
 return random.randint(10 \*\* 15, 10 \*\* 16 - 1)  
def hash\_key(secret\_key):  
 secret\_key = str(secret\_key)  
 digest = hashes.Hash(hashes.SHA256(), backend=default\_backend())  
 digest.update(secret\_key.encode())  
 return digest.finalize()  
def encrypt(text, key):  
 # Convert the text to bytes  
 data = text.encode()  
 # Generate a random initialization vector (IV)  
 iv = os.urandom(12)  
 cipher = Cipher(AES(key), modes.GCM(iv),  
 backend=default\_backend())  
 encryptor = cipher.encryptor()  
 # Encrypt the data  
 encrypted\_data = encryptor.update(data) + encryptor.finalize()  
 # Append the tag to the encrypted data  
 encrypted\_data\_with\_tag = encrypted\_data + encryptor.tag  
 # Convert the encrypted data with tag and IV to Base64 strings  
 encrypted\_data\_with\_tag\_str = b64encode(encrypted\_data\_with\_tag).decode()  
 iv\_str = b64encode(iv).decode()  
 print("encrypt:", text, key, encrypted\_data\_with\_tag\_str, iv\_str)  
 return encrypted\_data\_with\_tag\_str, iv\_str  
def decrypt(encrypted\_text, key, iv):  
 print("before decrypt :", encrypted\_text, key, iv)  
 iv = b64decode(iv)  
 # Decode the Base64 string back into bytes  
 encrypted\_data\_with\_tag = b64decode(encrypted\_text)  
 print("decrypt 1 :", encrypted\_data\_with\_tag)  
 # Separate the encrypted data and the tag  
 encrypted\_data = encrypted\_data\_with\_tag[:-16]  
 tag = encrypted\_data\_with\_tag[-16:]  
 decoded = ""  
 try:  
 cipher = Cipher(AES(key), modes.GCM(iv, tag),  
 backend=default\_backend())  
 decryptor = cipher.decryptor()  
 # Decrypt the data  
 decrypted\_data = decryptor.update(encrypted\_data) + decryptor.finalize()  
 decoded = decrypted\_data.decode()  
 except InvalidTag:  
 print("Impossible to decrypt")  
 print("decrypt 2 :", decoded)  
 return decoded

Листинг А.4 – models.py

from datetime import datetime  
from werkzeug.security import generate\_password\_hash, check\_password\_hash  
from app import db

Окончание листинга А.4

class User(db.Model):  
 id = db.Column(db.Integer, primary\_key=True)  
 username = db.Column(db.String(128), nullable=False)  
 email = db.Column(db.String(128), nullable=False, default="")  
 password\_hash = db.Column(db.String(256), nullable=False)  
 date = db.Column(db.DateTime, default=datetime.utcnow, comment='date of regi  
 station')  
 last\_seen = db.Column(db.DateTime, default=datetime.utcnow, comment='last se  
 en user in online')  
 role\_id = db.Column('role\_id', db.Integer, db.ForeignKey('role.id', ondelete  
 ='CASCADE'))  
 role = db.relationship('Role', backref='owners')  
 def \_\_repr\_\_(self) -> str:  
 return f'User {self.id}, Username: {self.username}, email: {self.email},  
 date: {self.date},' \  
 f' last\_seen: {self.last\_seen}'  
 def set\_password(self, password):  
 self.password\_hash = generate\_password\_hash(password)  
 def check\_password(self, password):  
 return check\_password\_hash(self.password\_hash, password)  
class Role(db.Model):  
 id = db.Column(db.Integer, primary\_key=True)  
 name = db.Column(db.String(64))  
class Dog(db.Model):  
 id = db.Column(db.Integer, primary\_key=True)  
 name = db.Column(db.String(128), nullable=False)  
 description = db.Column(db.String(256))  
 breed = db.Column(db.String(64))

Листинг А.5 – routes.py

import flask  
from flask import render\_template, jsonify, request, redirect, url\_for, make\_res  
ponse, Response  
from werkzeug.local import LocalProxy  
from app import diffi\_helman, db  
from app.diffi\_helman import hash\_key  
from app.models import Dog, User, Role  
from main import app  
current\_user = LocalProxy(lambda: User.query.filter\_by(id=flask.session.get('use  
r\_id')).first())  
@app.before\_request  
def make\_session\_permanent():  
 global current\_user  
 if not Role.query.filter\_by(name='Пользователь').first():  
 role = Role(name='Пользователь')  
 role1 = Role(name='Админ')  
 db.session.add(role)  
 db.session.add(role1)  
 db.session.commit()  
 current\_user = LocalProxy(lambda: User.query.filter\_by(id=flask.session.get(  
 'user\_id')).first())  
 print(current\_user)  
 flask.session.get('user\_id')  
 flask.session.permanent = True  
@app.route('/get', methods=['GET'])  
def get\_g\_and\_p():  
 """

Продолжение листинга А.5

Ручка выдающая значения g, p  
 :return:  
 """  
 g, p = diffi\_helman.get\_g\_p()  
 flask.session['g'] = g  
 flask.session['p'] = p  
 return jsonify({'g': g, 'p': p})  
@app.context\_processor  
def inject\_user():  
 """  
 Внедряет поле пользователя в Jinja если пользователь авторизован  
 :return:  
 """  
 return dict(user=current\_user if current\_user else False)  
@app.route('/send\_key', methods=['POST'])  
def send\_key():  
 """  
 Получает на вход открытый ключ клиента создает закрытый ключ  
 собирает открытый ключ и его отправляет клиенту  
 :return:  
 """  
 A = int(request.json['A'])  
 flask.session['A'] = A  
 g, p = flask.session.get('g'), flask.session.get('p')  
 b = diffi\_helman.secret\_key\_server()  
 B = diffi\_helman.get\_shared\_client\_key(g, b, p)  
 s\_server = diffi\_helman.get\_shared\_server\_key(A, b, p)  
 flask.session['s\_server'] = s\_server  
 return jsonify({'B': B})  
@app.route('/', methods=['GET'])  
def index():  
 """  
 Главная страница  
 """  
 dogs = Dog.query.all()  
 return render\_template('index.html', dogs=dogs)  
@app.route('/add', methods=['GET'])  
def add():  
 """  
 Форма добавления  
 """  
 print("asda", request.method)  
 if not flask.session.get(app.config['USER\_FIELD']):  
 return redirect(url\_for('login'))  
 return render\_template('insert.html')  
@app.route('/add', methods=['POST'])  
def add\_dog():  
 if not request.json:  
 return redirect(url\_for('add'))  
 name = request.json['name']  
 desc = request.json['description']  
 breed = request.json['breed']  
 iv = request.json['iv']  
 key = flask.session['s\_server']  
 key\_hash = hash\_key(key)  
 if iv:  
 name = diffi\_helman.decrypt(name, key\_hash, iv)  
 desc = diffi\_helman.decrypt(desc, key\_hash, iv)

Продолжение листинга А.5

breed = diffi\_helman.decrypt(breed, key\_hash, iv)  
 dog = Dog(name=name, description=desc, breed=breed)  
 db.session.add(dog)  
 db.session.commit()  
 return redirect(url\_for('index'))  
 return render\_template('insert.html')  
@app.route('/login', methods=['GET', 'POST'])  
def login():  
 if current\_user:  
 return redirect(url\_for('index'))  
 if request.method == 'POST':  
 if not request.json:  
 return redirect(url\_for('register'))  
 if flask.session.get(app.config['USER\_FIELD']):  
 return redirect(url\_for('index'))  
 user\_login = request.json['username']  
 password = request.json['password']  
 iv = request.json['iv']  
 key = flask.session['s\_server']  
 key\_hash = hash\_key(key)  
 if user\_login and password and iv:  
 username = diffi\_helman.decrypt(user\_login, key\_hash, iv)  
 password = diffi\_helman.decrypt(password, key\_hash, iv)  
 user = User.query.filter\_by(username=username).first()  
 if user is None:  
 return Response(status=404)  
 if not user.check\_password(password):  
 return Response(status=401)  
 flask.session[app.config['USER\_FIELD']] = user.id  
 return redirect(url\_for('index'))  
 return render\_template('login.html')  
@app.route('/logout', methods=['GET'])  
def logout():  
 flask.session.pop(app.config['USER\_FIELD'])  
 return redirect(url\_for('index'))  
@app.route('/register', methods=['GET', 'POST'])  
def register():  
 if current\_user:  
 return redirect(url\_for('index'))  
 print(request.method)  
 if request.method == 'POST':  
 if not request.json:  
 return redirect(url\_for('register'))  
 user\_login = request.json['username']  
 password = request.json['password']  
 admin = request.json['has\_admin']  
 iv = request.json['iv']  
 key = flask.session['s\_server']  
 key\_hash = hash\_key(key)  
 print(request.json)  
 if user\_login and password and iv:  
 username = diffi\_helman.decrypt(user\_login, key\_hash, iv)  
 password = diffi\_helman.decrypt(password, key\_hash, iv)  
 has\_admin = diffi\_helman.decrypt(admin, key\_hash, iv)  
 if User.query.filter\_by(username=username).first():  
 return Response(status=406)  
 user = User(username=username)  
 if has\_admin == 'true':

Окончание листинга А.5

user.role = Role.query.filter\_by(name='Админ').first()  
 else:  
 user.role = Role.query.filter\_by(name='Пользователь').first()  
 user.set\_password(password)  
 db.session.add(user)  
 db.session.commit()  
 flask.session[app.config['USER\_FIELD']] = User.query.filter\_by(usern  
 ame=username).first().id  
 return redirect(url\_for('index'))  
 return render\_template('register.html')  
@app.route('/delete', methods=['POST'])  
def delete():  
 if not request.is\_json:  
 return redirect(url\_for('index'))  
 if not current\_user:  
 return redirect(url\_for('login'))  
 user = User.query.filter\_by(id=flask.session.get('user\_id')).first()  
 if user.role.name != 'Админ':  
 return redirect(url\_for('index'))  
 iv = request.json['iv']  
 key = flask.session['s\_server']  
 key\_hash = hash\_key(key)  
 dog\_id = diffi\_helman.decrypt(request.json['dog\_id'], key\_hash, iv)  
 db.session.delete(Dog.query.filter\_by(id=dog\_id).first())  
 db.session.commit()  
 return redirect(url\_for('index'))

Листинг А.6 – config.py

import os  
basedir = os.path.abspath(os.path.dirname(\_\_file\_\_))  
class Config(object):  
 SQLALCHEMY\_DATABASE\_URI = 'sqlite:///' + os.path.join(basedir, 'appdb')  
 SECRET\_KEY = os.environ.get('SECRET\_KEY') or 'zxczxc'  
 SQLALCHEMY\_TRACK\_MODIFICATIONS = False  
 USER\_FIELD = 'user\_id'

Листинг А.7 – main.py

from app import app  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.config['SESSION\_TYPE'] = 'filesystem'  
 app.run(debug=True)

Листинг А.8 – crypto.js

B = 0  
key = localStorage.getItem('key')  
if (key) {  
 S\_Client = key  
}  
function getRandomArbitrary(min, max) {  
 return bigInt(parseInt(Math.random() \* (max - min) + min));  
}  
let random = getRandomArbitrary(Math.pow(10, 15), Math.pow(10,16)-1)  
function validate\_fields(username, password, error\_field) {  
 if(!username) {  
 error\_field.innerText = 'Вы не ввели логин'

Продолжение листинга А.8

}  
 if(!password) {  
 error\_field.innerText = 'Вы не ввели пароль'  
 }  
}  
a = random  
function fetchData() {  
 const xhr = new XMLHttpRequest();  
 const url = '/get';  
 xhr.open('GET', url, true);  
 // Обработчик события загрузки  
 xhr.onload = function () {  
 if (xhr.status === 200) {  
 const data = JSON.parse(xhr.responseText);  
 console.log(data)  
 const g = bigInt(data.g);  
 const p = bigInt(data.p);  
 console.log(typeof g)  
 const requestBody = JSON.stringify({A: g.modPow(a, p)});  
 xhr.open('POST', '/send\_key', false);  
 xhr.setRequestHeader('Content-Type', 'application/json');  
 // Обработчик события загрузки  
 xhr.onload = function () {  
 if (xhr.status === 200) {  
 const responseData = JSON.parse(xhr.responseText);  
 B = bigInt(parseInt(responseData.B));  
 S\_Client = B.modPow(a, p)  
 localStorage.setItem('key', S\_Client)  
 console.log(`B = ${B}, S\_Client = ${S\_Client}`);  
 } else {  
 console.error(`Ошибка при отправке данных. Статус: ${xhr.sta  
 tus}`);  
 }  
 };  
 // Обработчик сетевых ошибок  
 xhr.onerror = function () {  
 console.error('Произошла сетевая ошибка');  
 };  
 // Отправляем запрос  
 xhr.send(requestBody);  
 console.log(`Получены значения g=${g} и p=${p}`);  
 } else {  
 console.error(`Ошибка при получении данных. Статус: ${xhr.status}`);  
 }  
 };  
 // Обработчик сетевых ошибок  
 xhr.onerror = function () {  
 console.error('Произошла сетевая ошибка');  
 };  
 // Отправляем запрос  
 xhr.send();  
}  
if(!key) {  
 fetchData()  
}  
async function processForm() {  
 event.preventDefault()  
 username = document.forms[0].username.value

Продолжение листинга А.8

password = document.forms[0].password.value  
 error\_field = document.getElementById('error')  
 validate\_fields(username, password, error\_field)  
 let {key, iv} = await convertKey(S\_Client);  
 console.log(key)  
 let encryptedName = await encrypt(username, key, iv);  
 let encryptedPassword = await encrypt(password, key, iv);  
 let data = {  
 iv: btoa(String.fromCharCode.apply(null, iv)),  
 username: encryptedName,  
 password: encryptedPassword  
 };  
 console.log(JSON.stringify(data))  
 const response = await fetch('/login', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify(data)  
 })  
 const responseData = await response  
 showError(error\_field, responseData.status, responseData.url)  
}  
async function convertKey(secretKey){  
 const encoder = new TextEncoder();  
 const secretKeyBuffer = encoder.encode(secretKey);  
 const hashedKey = await window.crypto.subtle.digest('SHA-256', secretKeyBuff  
 er);  
 // Import the hashed key as a CryptoKey  
 const key = await window.crypto.subtle.importKey(  
 'raw',  
 hashedKey,  
 'AES-GCM',  
 false,  
 ['encrypt', 'decrypt']  
 );  
 const iv = window.crypto.getRandomValues(new Uint8Array(12));  
 console.log(key)  
 return {key, iv};  
}  
async function encrypt(text, key, iv) {  
 // The data to encrypt  
 const data = new TextEncoder().encode(text);  
 // Encrypt the data  
 const encryptedData = await window.crypto.subtle.encrypt(  
 {  
 name: 'AES-GCM',  
 iv: iv  
 },  
 key,  
 data  
 );  
 // Convert the encrypted data (including the tag) to a Base64 string  
 const encryptedDataArray = new Uint8Array(encryptedData);  
 console.log(encryptedDataArray)  
 return btoa(String.fromCharCode.apply(null, encryptedDataArray));  
}  
async function register(event) {

Продолжение листинга А.8

console.log('vvvvvvvv')  
 event.preventDefault()  
 username = document.forms[0].username.value  
 password = document.forms[0].password.value  
 has\_admin = document.forms[0].has\_admin.checked.toString()  
 error\_field = document.getElementById('error')  
 validate\_fields(username, password, error\_field)  
 let {key, iv} = await convertKey(S\_Client);  
 console.log(key)  
 let encryptedName = await encrypt(username, key, iv);  
 let encryptedPassword = await encrypt(password, key, iv);  
 let encryptedAdmin = await encrypt(has\_admin, key, iv);  
 let data = {  
 iv: btoa(String.fromCharCode.apply(null, iv)),  
 username: encryptedName,  
 password: encryptedPassword,  
 has\_admin: encryptedAdmin  
 };  
 console.log(JSON.stringify(data))  
 const response = await fetch('/register', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify(data)  
 })  
 const responseData = await response  
 showError(error\_field, responseData.status, responseData.url)  
}  
async function addDog(event) {  
 event.preventDefault()  
 name = document.forms[0].name.value  
 description = document.forms[0].description.value  
 breed = document.forms[0].breed.value  
 error\_field = document.getElementById('error')  
 let {key, iv} = await convertKey(S\_Client);  
 let encryptedName = await encrypt(name, key, iv);  
 let encryptedDesc = await encrypt(description, key, iv);  
 let encryptedBreed = await encrypt(breed, key, iv);  
 let data = {  
 iv: btoa(String.fromCharCode.apply(null, iv)),  
 name: encryptedName,  
 description: encryptedDesc,  
 breed: encryptedBreed,  
 };  
 console.log(JSON.stringify(data))  
 const response = await fetch('/add', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify(data)  
 })  
 const responseData = await response  
 window.location.href = responseData.url;  
}  
function showError(error\_field, status, url) {  
 if(status === 200) {

Окончание листинга А.8

window.location.href = url;  
 }  
 if(status === 403) {  
 window.location.href = url;  
 }  
 if(status === 406) {  
 error\_field.innerText = 'Такой пользователь уже существует'  
 }  
 if(status === 404) {  
 error\_field.innerText = 'Такого пользователя не существует'  
 }  
 if(status === 401) {  
 error\_field.innerText = 'Неверный пароль'  
 }  
}