|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** \_***ИУ-КФ «Информатики и управления»*\_\_**\_\_\_\_\_\_\_\_\_\_\_\_

**КАФЕДРА** \_\_***ИУ6-КФ «Защита информации»***\_\_\_\_\_\_\_\_\_\_

**ЛАБОРАТОРНАЯ РАБОТА №3**

**ДИСЦИПЛИНА: «Защита информации»**

**ТЕМА: «Программная реализация асимметричного шифра. Алгоритм RSA»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. САПР Б.-81 | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Алпатов И.О. )  (Подпись) (Ф.И.О.)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Дунаев А.В. )  (Подпись) (Ф.И.О.) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Бурмистров А.В. )  (Подпись) (Ф.И.О.) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга , 2019

# Цель работы

Реализовать шифр с открытыми ключами. Алгоритм RSA.

# Задание

1. Изучить теоретические основы алгоритма RSA.

2. Разработать программу, реализующую предложенную методику шифрования.

3. Апробировать разработанную программу.

4. Оформить отчет по лабораторной работе.

# Теоретическая часть

Алгоритм RSA стоит у истоков асимметричной криптографии. Он был предложен тремя исседователями-математиками Рональдом Ривестом (R.Rivest) , Ади Шамиром (A.Shamir) и Леонардом Адльманом (L.Adleman) в 1977-78 годах.

Первым этапом любого асимметричного алгоритма является создание пары ключей: открытого и закрытого и распространение открытого ключа "по всему миру".

Сначала необходимо вычислить пару ключей (секретный ключ и открытый ключ). Для этого отправитель (автор) электронных документов вычисляет два больших простых числа Р и Q, затем находит их произведение

N = Р \* Q

и значение функции

j (N) = (Р-1)(Q-1).

Далее отправитель вычисляет число Е из условий:

Е £ j (N), НОД (Е, j (N)) = 1

и число D из условий:

D < N, Е\*D º 1 (mod j (N)).

Пара чисел (Е, N) является открытым ключом. Эту пару чисел автор передает партнерам по переписке для проверки его цифровых подписей. Число D сохраняется автором как секретный ключ для подписывания.

Обобщенная схема формирования и проверки цифровой подписи RSА показана на рисунке 1.

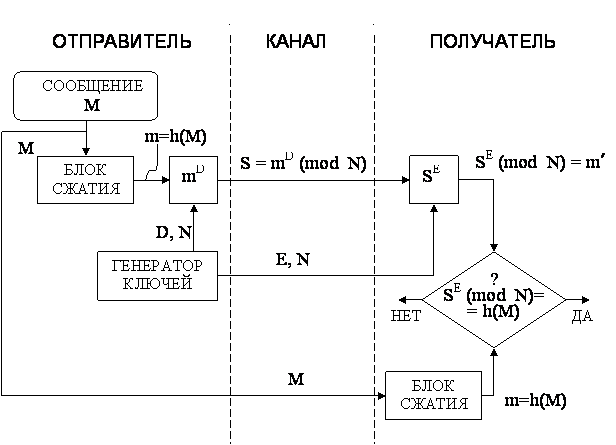


Рисунок 1- Обобщённая схема цифровой подписи RSA

Допустим, что отправитель хочет подписать сообщение М перед его отправкой. Сначала сообщение М (блок информации, файл, таблица) сжимают с помощью хэш-функции h(·) в целое число m:

m = h(М).

Затем вычисляют цифровую подпись S под электронным документом М, используя хэш-значение m и секретный ключ D:

S = mD (mod N).

Пара (М,S) передается партнеру-получателю как электронный документ М, подписанный цифровой подписью S, причем подпись S сформирована обладателем секретного ключа D.

После приема пары (М,S) получатель вычисляет хэш-значение сообидения М двумя разными способами. Прежде всего он восстанавливает хэш-значение m', применяя криптографическое преобразование подписи S с использованием открытого ключа Е:

m' = SE (mod N).

Кроме того, он находит результат хэширования принятого сообщения М с помощью такой же хэш-функции h(·):

m = h(М).

Если соблюдается равенство вычисленных значений, т.е.

SE ( mod N ) = h ( М ),

то получатель признает пару (М,S) подлинной. Доказано, что только обладатель секретного ключа D может сформировать цифровую подпись S по документу М, а определить секретное число D по открытому числу Е не легче, чем разложить модуль N на множители.

Кроме того, можно строго математически доказать, что результат проверки цифровой подписи S будет положительным только в том случае, если при вычислении S был использован секретный ключ D, соответствующий открытому ключу Е. Поэтому открытый ключ Е иногда называют "идентификатором" подписавшего

# Результаты работы

В рамках данной работы, на языке C# было реализовано клиент-серверное приложение чата, реализующее шифрование RSA. Скриншоты готового приложения представлены на рисунках 2 и 3. Алгоритм последовательности авторизации пользователей представлен на рисунке 4.

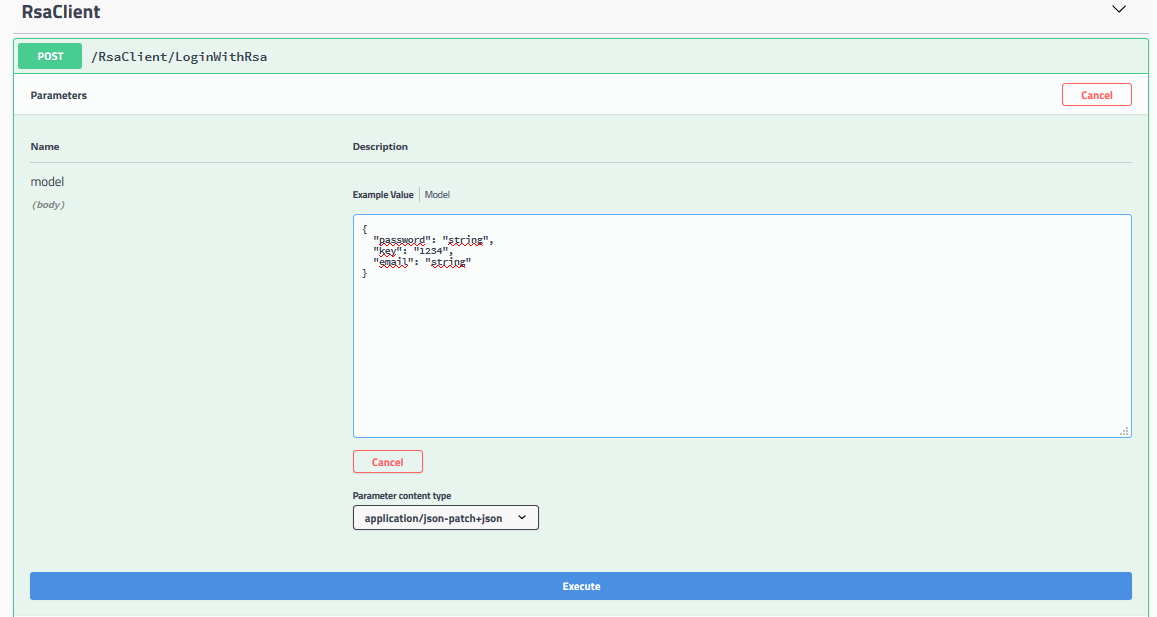


Рисунок 2 – Форма логина в приложение

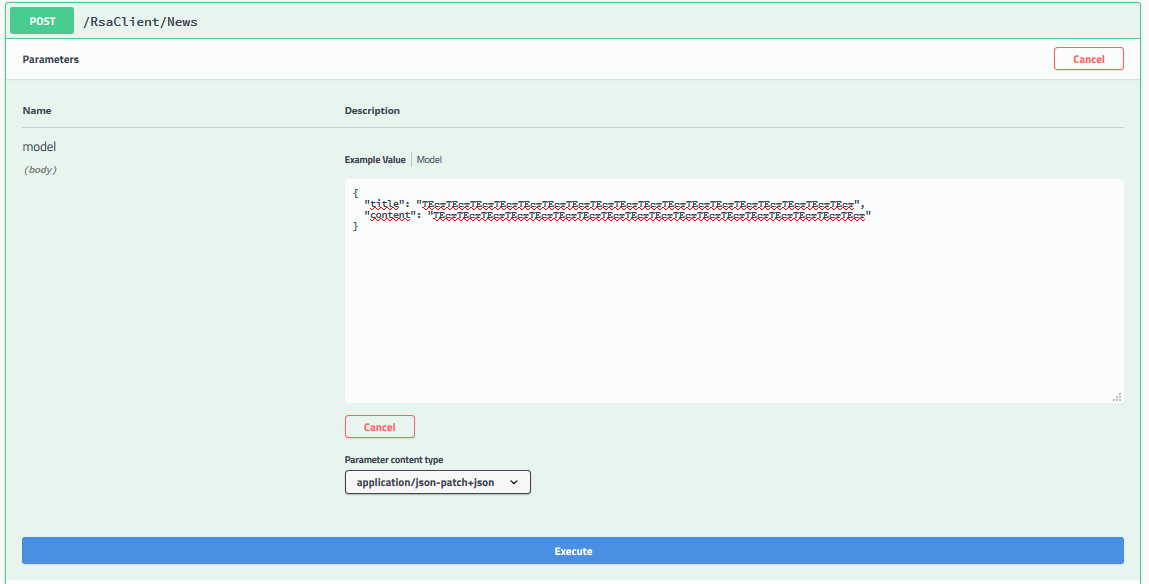


Рисунок 3 – Интерфейс приложения

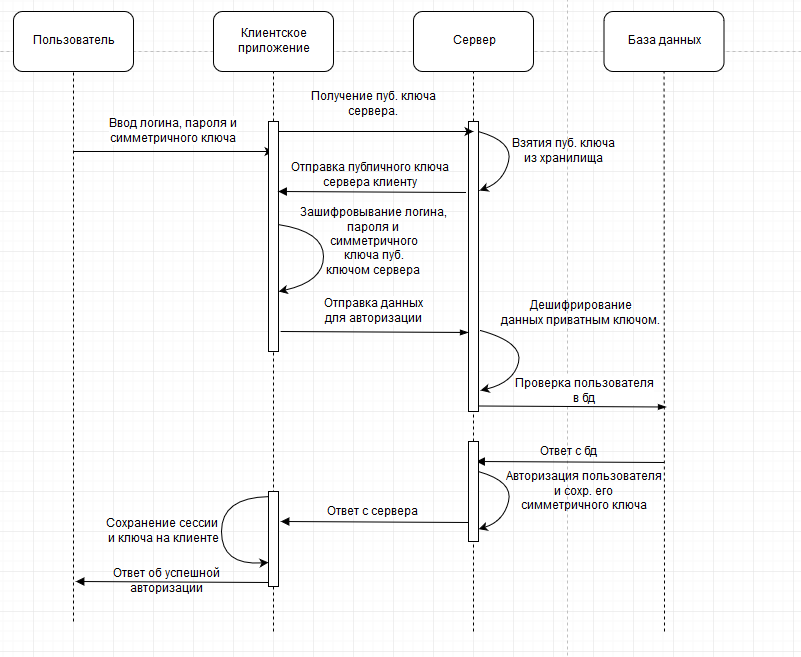


Рисунок 4 - Алгоритм последовательности авторизации пользователей

Алгоритм передачи сообщения от клиента к серверу и последующая отправка клиентам представлена на рисунке 5.

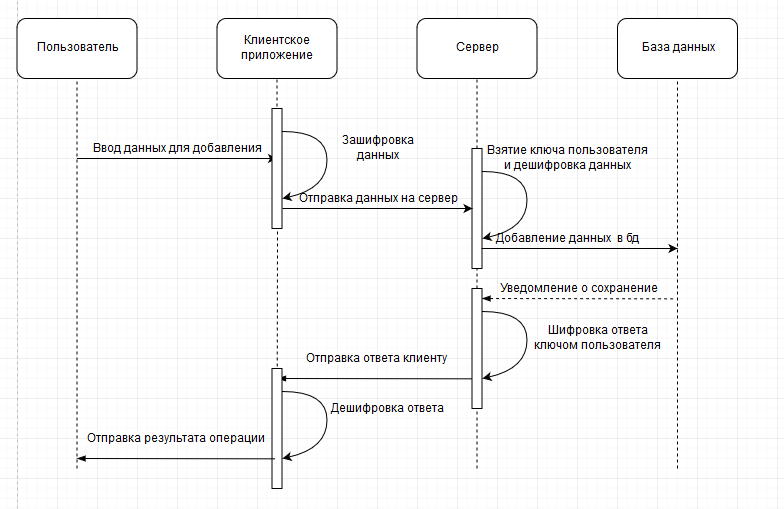


Рисунок 5 - Алгоритм передачи сообщения от клиента к серверу

# Вывод

В результате данной работы были изучены виды ассиметричного шифрования. Были получены практические навыки по реализации RSA шифрования в многопользовательском клиент-серверном приложении чата. Приложение на языке JavaScript реализует выбранный метод шифрования сообщений при общении пользователей. Приложение было апробировано на нескольких машинах и пользователях. Замеченные в процессе апробирования недостатки и уязвимости могут быть выявлены и устранены в процессе дальнейшей разработки.

# Приложение А

**Общая часть**

/// <summary>

/// Сервис для работы с RSA

/// </summary>

public class RsaService

{

private readonly ILogger<RsaService> \_logger;

public RsaService(ILogger<RsaService> logger)

{

\_logger = logger;

}

public string Encrypt(object model, RsaPublicKey publicKey)

{

var text = JsonConvert.SerializeObject(model);

\_logger.LogInformation($"Rsa: \nВходная модель: {text}");

var enc = new RSACryptoServiceProvider();

var rsaParametr = publicKey.GetRsaParameters();

enc.ImportParameters(rsaParametr);

return Convert.ToBase64String(enc.Encrypt(Encoding.ASCII.GetBytes(text), false));

}

public T Decrypt<T>(string data, RSAParameters privateKey)

{

var enc = new RSACryptoServiceProvider();

enc.ImportParameters(privateKey);

var resultData = Encoding.ASCII.GetString(enc.Decrypt(Convert.FromBase64String(data), false));

\_logger.LogInformation($"Rsa: \nРезультат: {resultData}");

return JsonConvert.DeserializeObject<T>(resultData);

}

public string Decrypt(string data, RSAParameters privateKey)

{

var enc = new RSACryptoServiceProvider();

enc.ImportParameters(privateKey);

var resultData = Encoding.ASCII.GetString(enc.Decrypt(Convert.FromBase64String(data), false));

\_logger.LogInformation($"Rsa: \nРезультат: {resultData}");

return resultData;

}

public static RsaKeys GetKeyPair(int length = 2048)

{

var encProvider = new RSACryptoServiceProvider(length);

var publicKey = encProvider.ExportParameters(false);

var privateKey = encProvider.ExportParameters(true);

return new RsaKeys(publicKey, privateKey);

}

}

public class RsaKeys

{

public RSAParameters PublicKey { get; set; }

public RSAParameters PrivateKey { get; set; }

/// <inheritdoc />

public RsaKeys(RSAParameters publicKey, RSAParameters privateKey)

{

PublicKey = publicKey;

PrivateKey = privateKey;

}

}

public class RsaPublicKey

{

public string Modulus { get; set; }

public string Exponent { get; set; }

/// <inheritdoc />

public RsaPublicKey(byte[] modulus, byte[] exponent)

{

Modulus = Convert.ToBase64String(modulus);

Exponent = Convert.ToBase64String(exponent);

}

public RsaPublicKey(RSAParameters publicKey)

: this(publicKey.Modulus, publicKey.Exponent)

{

}

public RsaPublicKey()

{

}

public RSAParameters GetRsaParameters()

{

return new RSAParameters

{

Modulus = GetModulus(),

Exponent = GetExponent()

};

}

public byte[] GetModulus()

{

return Convert.FromBase64String(Modulus);

}

public byte[] GetExponent()

{

return Convert.FromBase64String(Exponent);

}

}

/// <summary>

/// Модель для пердачи служебной инфы

/// </summary>

public class ServiceInfo

{

public string Password { get; set; }

/// <summary>

/// Симетричный ключ

/// </summary>

public string Key { get; set; }

public string Email { get; set; }

}

**Серверная часть**

/// <summary>

/// Возврат пуб. ключа

/// </summary>

/// <param name=""></param>

/// <returns></returns>

[HttpGet("RsaPublicKey")]

public async Task<RsaPublicKey> GetPublicKey()

{

return new RsaPublicKey(\_rsaServerKeys.PublicKey);

}

/// <summary>

/// Устанавливает на сервере сим. ключ

/// и авторизация пользователя

/// </summary>

/// <param name="publicKey"></param>

/// <returns>Возвращает публичный ключ сервера</returns>

[HttpPost("LoginWithRsa"), AllowAnonymous]

public async Task SetKey(ServiceInfo serviceInfo)

{

\_logger.LogInformation($"Входная модель:{JsonConvert.SerializeObject(serviceInfo)}");

var email = serviceInfo.Email;

var decryptKey = \_rsaService.Decrypt(serviceInfo.Key, \_rsaServerKeys.PrivateKey).Replace("\"", "");

\_logger.LogInformation($"Дешифрованный приватным ключом, Ключ гронсфельда:{decryptKey}");

var password = \_rsaService.Decrypt<string>(serviceInfo.Password, \_rsaServerKeys.PrivateKey);

\_logger.LogInformation($"Дешифрованный пароль:{password}");

var user = \_userAccount.Login(new UserLogin

{

Email = email,

Password = password

});

if(user==null)

throw new NullReferenceException("Такого пользователя не существует.");

\_memoryCache.Set(email + "3", new GronsKeyModel {Key = decryptKey});

var claims = new List<Claim>

{

new Claim(ClaimTypes.Name, serviceInfo.Email),

new Claim(ClaimsIdentity.DefaultNameClaimType, serviceInfo.Email),

new Claim(ClaimTypes.Email, serviceInfo.Email)

};

var claimsIdentity = new ClaimsIdentity(

claims,

CookieAuthenticationDefaults.AuthenticationScheme);

await HttpContext.SignInAsync(

CookieAuthenticationDefaults.AuthenticationScheme,

new ClaimsPrincipal(claimsIdentity));

}

/// <summary>

/// Созд. новости

/// </summary>

/// <param name="model"></param>

/// <returns></returns>

[HttpPost("News")]

public async Task<string> CreateNews(NewsInfo model)

{

var key = \_memoryCache.Get<GronsKeyModel>(HttpContext.GetEmail() + "3").Key;

\_logger.LogInformation($"Ключ пользователя {HttpContext.GetEmail()} : {key}");

\_logger.LogInformation("Входная модель новости: " + JsonConvert.SerializeObject(model));

model.Title = \_gronsfeldService.Decrypt(model.Title, key);

model.Content = \_gronsfeldService.Decrypt(model.Content, key);

\_logger.LogInformation("Входная модель новости после дешифровки: " + JsonConvert.SerializeObject(model));

var result = await \_newsService.Create(model);

return \_gronsfeldService.Encrypt(result.ToString(), key);

}

/// <summary>

/// Возвращение новостей

/// </summary>

/// <returns></returns>

[HttpGet("News")]

public async Task<List<NewsModel>> Get()

{

var key = \_memoryCache.Get<GronsKeyModel>(HttpContext.GetEmail() + "3").Key;

\_logger.LogInformation($"Ключ пользователя {HttpContext.GetEmail()} : {key}");

var result = await \_newsService.Get();

foreach (var newsModel in result)

{

newsModel.Content = \_gronsfeldService.Encrypt(newsModel.Content, key);

newsModel.Title = \_gronsfeldService.Encrypt(newsModel.Title, key);

}

\_logger.LogInformation($"Отправляемая модель: {JsonConvert.SerializeObject(result)}");

return result;

}

**Клиентская часть**

[HttpPost("LoginWithRsa")]

public async Task SetKey(ServiceInfo model)

{

await \_rsaHttpService.LoginWithRsa(model);

var claims = new List<Claim>

{

new Claim(ClaimTypes.Email, model.Email),

new Claim(ClaimsIdentity.DefaultNameClaimType, model.Email)

};

var claimsIdentity = new ClaimsIdentity(

claims,

CookieAuthenticationDefaults.AuthenticationScheme);

await HttpContext.SignInAsync(

CookieAuthenticationDefaults.AuthenticationScheme,

new ClaimsPrincipal(claimsIdentity));

}

[HttpPost("News")]

public async Task<Guid> CreateNews(NewsInfo model)

{

return await \_rsaHttpService.CreateNews(HttpContext.GetEmail(), model);

}

[HttpGet("News")]

public async Task<List<NewsModel>> Get()

{

return await \_rsaHttpService.Get(HttpContext.GetEmail());

}

/// <summary>

/// Сервис для работы с клиентом

/// </summary>

public class RsaHttpService

{

private readonly HttpClient \_httpClient;

private readonly IMemoryCache \_memoryCache;

private readonly RsaService \_rsaService;

private GronsfeldService \_gronsfeldService;

private readonly ILogger<NewsService> \_logger;

public RsaHttpService(HttpClient httpClient,

IMemoryCache memoryCache,

RsaService rsaService, ILogger<NewsService> logger,

GronsfeldService gronsfeldService)

{

\_httpClient = httpClient;

\_memoryCache = memoryCache;

\_rsaService = rsaService;

\_logger = logger;

\_gronsfeldService = gronsfeldService;

}

public async Task LoginWithRsa(ServiceInfo model)

{

\_logger.LogInformation($"Пользователь {model.Email} ввел данные:{JsonConvert.SerializeObject(model)}");

var res = await \_httpClient.GetAsync($"rsa/RsaPublicKey");

res.EnsureSuccessStatusCode();

var serverPublicKey = await res.Content.ReadAsAsync<RsaPublicKey>();

\_logger.LogInformation($"Rsa:Публичный ключ с сервера: {await res.Content.ReadAsStringAsync()}");

var encryptPassword = \_rsaService.Encrypt(model.Password, serverPublicKey);

\_logger.LogInformation($"Зашифрованный пароль гронсфельдом:{encryptPassword}");

var encryptKey = \_rsaService.Encrypt(model.Key, serverPublicKey);

\_logger.LogInformation($"Зашифрованный Ключ пуб. ключом сервера:{encryptKey}");

var gronsKey = model.Key.ToString();

model.Key = encryptKey;

model.Password = encryptPassword;

var authResponse = await \_httpClient.PostAsync($"rsa/LoginWithRsa", JsonContent.Convert(model));

authResponse.EnsureSuccessStatusCode();

\_logger.LogInformation($"Ответ успешен");

\_memoryCache.Set(model.Email + ":serverPubKey", serverPublicKey);

var cookie = authResponse.Headers.GetValues(CacheKeyCookieModel.GetCookieName);

\_memoryCache.Set(model.Email + "3", new CacheKeyCookieModel(gronsKey, cookie), TimeSpan.FromDays(1));

\_logger.LogInformation($"Куки установлены");

}

public async Task<Guid> CreateNews(string email, NewsInfo model)

{

var userModel = \_memoryCache.Get<CacheKeyCookieModel>(email + "3");

\_logger.LogInformation($"Ключ пользователя {email} : {userModel.Key}");

\_httpClient.DefaultRequestHeaders.Add(CacheKeyCookieModel.SetCookie, userModel.Cookie);

\_logger.LogInformation($"Отправляемая модель на сервер перед шифрованием:{JsonConvert.SerializeObject(model)}");

model.Title = \_gronsfeldService.Encrypt(model.Title, userModel.Key);

model.Content = \_gronsfeldService.Encrypt(model.Content, userModel.Key);

\_logger.LogInformation($"Отправляемая модель на сервер после шифрования:{JsonConvert.SerializeObject(model)}");

var response = await \_httpClient.PostAsync($"rsa/News", JsonContent.Convert(model));

var responseString = await response.Content.ReadAsStringAsync();

var newsGuid = \_gronsfeldService.Decrypt(responseString, userModel.Key);

return Guid.Parse(newsGuid);

}

public async Task<List<NewsModel>> Get(string email)

{

var userModel = \_memoryCache.Get<CacheKeyCookieModel>(email + "3");

\_logger.LogInformation($"Ключ пользователя {email} : {userModel.Key}");

\_httpClient.DefaultRequestHeaders.Add(CacheKeyCookieModel.SetCookie, userModel.Cookie);

\_logger.LogInformation($"Запрос на новости отпарвляется.");

var response = await \_httpClient.GetAsync($"rsa/News");

var responseList = await response.Content.ReadAsAsync<List<NewsModel>>();

\_logger.LogInformation($"Пришедший список: {JsonConvert.SerializeObject(responseList)}");

foreach (var item in responseList)

{

item.Content = \_gronsfeldService.Decrypt(item.Content, userModel.Key);

item.Title = \_gronsfeldService.Decrypt(item.Title, userModel.Key);

}

\_logger.LogInformation($"Пришедший список после дешифровки: {JsonConvert.SerializeObject(responseList)}");

return responseList;

}

}

public class ClientRsaKeys

{

public RsaPublicKey RsaPublicKeyServer { get; set; }

public RsaPublicKey RsaPublicKeyUser { get; set; }

public RSAParameters RsaPrivateKeyUser { get; set; }

}