|  |  |
| --- | --- |
| **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 №2**

**ДИСЦИПЛИНА: «Защита информации»**

**ТЕМА: «Программная реализация блочного симметричного шифра»**

|  |  |  |
| --- | --- | --- |
| 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

Разработать алгоритм и реализовать клиент серверную программу, реализующую шифр Гронсфельда.

# Задание

1. Изучить теоретические основы симметричного шифрования.

2. Разработать алгоритм программы, реализующей шифр сложной замены.

3. Реализовать программу по разработанному алгоритму.

4. Апробировать разработанную программу.

5. Оформить отчет по лабораторной работе.

# Теоретическая часть

Шифр Гронсфельда — [полиалфавитный подстановочный шифр](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B8%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82%D0%BD%D1%8B%D0%B9_%D1%88%D0%B8%D1%84%D1%80" \o "Полиалфавитный шифр) создан графом Гронсвельдом (руководителем первой дешифровальной службы Германии) в XVII веке. Шифр можно считать усовершенствованием шифра Цезаря (надежность) и Виженера / Бофора (скорость).

Длина ключа (*K*) должна быть равной длине исходного текста. Для этого циклически записывают ключ до тех пор, пока его длина не будет соответствовать длине исходного текста.

1. Шифрование:

Каждый символ *Mi* открытого текста *M* нужно на *Ki* (соответствующий символ ключа *K*) шагов сдвинуть вправо.  
Или пользуясь таблицей Гронсфельда (*Tx y*, где *x* — номер строки, а *y* — номер столбца и отсчет ведется с нуля):   
каждый символ *Ci* шифротекста *C* находится на пересечении столбца *y*, первый (заголовочный) символ которого равен соответствующему символу открытого текста *Mi*, и *Ki*-й (соответствующей цифры ключа) строки — (*TKi y*)



Рисунок 1 – Таблица Гронсфельда.

1. Дешифрование:

Каждый символ (*Ci*) зашифрованого текста *C* нужно на *Ki* (соответствующий символ ключа *K*) шагов сдвинуть влево.  
Или пользуясь таблицей Гронсфельда (*Tx y*, где *x* — номер строки, а *y* — номер столбца и отсчет ведется с нуля):   
нужно в *Ki* (*i*-ая цифра ключа *K*) строке найти символ, который равен соответствующему символу шифротекста (*TKi y = Ci*), и первый (заголовочный) элемент столбца будет *i*-ый символ открытого текста.

# Результаты работы

В рамках данной работы, на языке C# было реализовано клиент-серверное приложение чата, реализующее шифрование сообщений шифром Грондсфельда. Скриншот готового приложения представлен на рисунке 2.

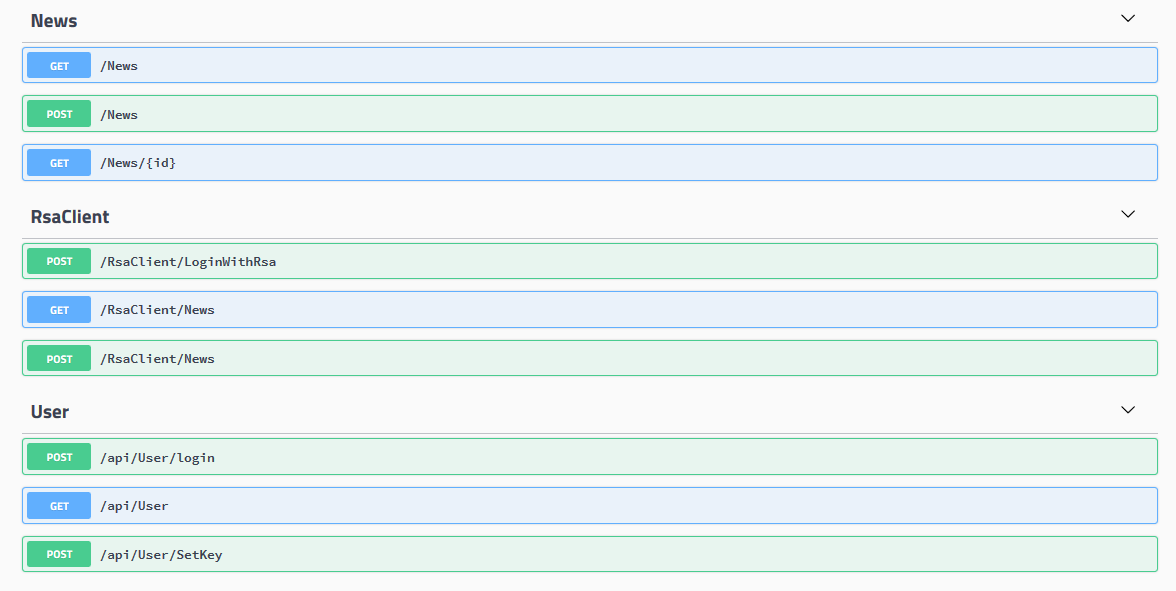


Рисунок 2 – Интерфейс готовой программы

На рисунке 3 представлен алгоритм сохранения ключа пользователя.

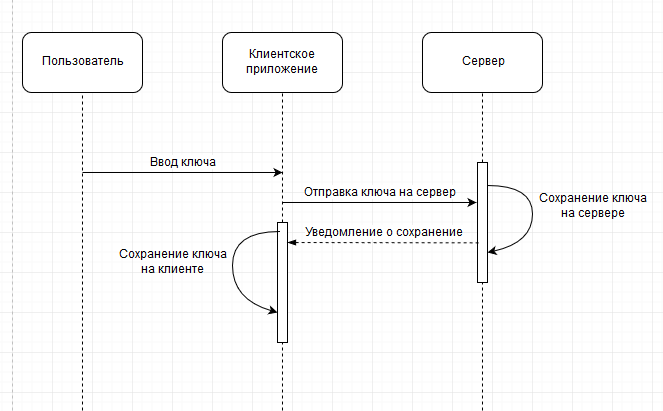


Рисунок 3 - Алгоритм сохранения ключа пользователя

Алгоритм передачи сообщения от клиента к серверу и последующая отправка клиентам представлена на рисунке 4.

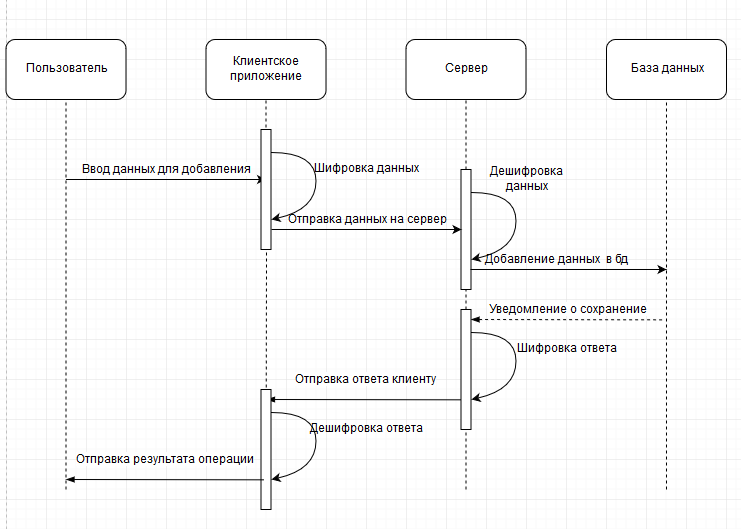


Рисунок 4 - Алгоритм передачи сообщения от клиента к серверу

# Вывод

В результате данной работы были изучены теоретические основы видов шифрования. Были получены практические навыки по реализации шифров сложной замены, а именно, Шифра Гронсфельда. Был разработан алгоритм шифра сложной замены и реализовано клиент-серверное приложение на языке C#, которое реализует выбранный метод шифрования сообщений при общении пользователей.

# Приложение А

**Общая часть**

**//**Сервис для работы с гронсфельдом

public class GronfeldEncrypt

{

/// <summary>

/// Шифрование

/// </summary>

/// <param name="alf">Алфавит</param>

/// <param name="key">Ключ</param>

/// <param name="data">Данные для шифрованияы</param>

/// <returns></returns>

public string EncryptGronsfeld(string alf, string key, string data)

{

data = Convert.ToBase64String(Encoding.UTF8.GetBytes(data));

var length = data.Length/alf.Length + 2;

var alfLocal = GetAlfFullString(alf, alf.Length \* length);

var keyLocal = GetKeyFullString(key, data.Length);

var resultEncData = "";

//Encrypt

for (int i = 0; i < data.Length; i++)

{

var dataChar = data[i];

var indexInEng = alfLocal.IndexOf(dataChar);

indexInEng += int.Parse(keyLocal[i].ToString()) + i;

resultEncData += alfLocal[indexInEng].ToString();

}

return resultEncData;

}

/// <summary>

/// Дешифрование

/// </summary>

/// <param name="alf">Алфавит</param>

/// <param name="key">Ключ</param>

/// <param name="encryptData">Данные для дешифровки</param>

/// <returns></returns>

public string DecryptGronsfeld(string alf, string key, string encryptData)

{

var resultEncData = "";

var length = encryptData.Length/alf.Length + 2;

var alfLocal = GetAlfFullString(alf, alf.Length \* length);

var keyLocal = GetKeyFullString(key, encryptData.Length);

for (int i = 0; i < encryptData.Length; i++)

{

var dataChar = encryptData[i];

var indexInEng = alfLocal.LastIndexOf(dataChar);

indexInEng -= (int.Parse(keyLocal[i].ToString()) + i);

resultEncData += alfLocal[indexInEng].ToString();

}

resultEncData = Encoding.UTF8.GetString(Convert.FromBase64String(resultEncData));

return resultEncData;

}

private string GetAlfFullString(string key, int count)

{

var resultKey = "";

for (int i = 0, keyIndex = 0; i < count; i++, keyIndex++)

{

if (keyIndex == key.Length)

keyIndex = 0;

resultKey += key[keyIndex];

}

return resultKey;

}

private string GetKeyFullString(string key, int count)

{

var resultKey = "";

for (int i = 0, keyIndex = 0; i < count; i++, keyIndex++)

{

if (keyIndex == key.Length)

keyIndex = 0;

resultKey += key[keyIndex];

}

return resultKey;

}

}

public class GronsfeldOptions

{

/// <summary>

/// Алфавит

/// </summary>

public string Alp { get; set; }

public int KeyLenght { get; set; }

}

public class GronsfeldService

{

private readonly GronfeldEncrypt \_encrypt;

private readonly GronsfeldOptions \_options;

public GronsfeldService(GronfeldEncrypt encrypt, GronsfeldOptions options)

{

\_encrypt = encrypt;

\_options = options;

}

public string Encrypt(string data, string key)

{

return \_encrypt.EncryptGronsfeld(\_options.Alp, key, data);

}

public string Decrypt(string data, string key)

{

return \_encrypt.DecryptGronsfeld(\_options.Alp, key, data);

}

}

public class NewsInfo

{

public string Title { get; set; }

public string Content { get; set; }

}

**Серверная часть**

//Установка ключа

[HttpPost("Key"), Authorize]

public void SetKey(string key)

{

var email = HttpContext.GetEmail();

\_logger.LogInformation($"Пользователь:{email} установил себе ключ:{key}");

if(!key.All(char.IsDigit))

throw new ArgumentException("Ключ не валидный.");

\_memoryCache.Set(email, key, TimeSpan.FromDays(1));

}

//Метод приемы новостей

[JsonEncryptResultFilter]

[HttpPost]

public async Task<Guid> Create(string data, [ModelBinder(typeof(CustomJsonModelBinder))] NewsInfo model)

{

return await \_newsService.Create(model);

}

/// Дешифрока данных от клиента

public async Task BindModelAsync(ModelBindingContext bindingContext)

{

var request = bindingContext.HttpContext.Request;

var type = bindingContext.ModelType;

if (bindingContext == null)

{

throw new ArgumentNullException(nameof(bindingContext));

}

var query = request.Query["data"].ToString().Replace(" ", "+");

\_logger.LogInformation($"Данные с запроса:{query}");

var key = \_memoryCache.Get<string>(bindingContext.HttpContext.GetEmail());

var decryptModel = \_gronsfeldService.Decrypt(query, key);

var resultData = JsonConvert.DeserializeObject(decryptModel, type);

\_logger.LogInformation(

$"Входные данные:{query}\nКлюч пользователя:{key}\nРезультат дешифровки:{JsonConvert.SerializeObject(resultData)}");

bindingContext.Result = ModelBindingResult.Success(resultData);

}

//Шифрока данных клиенту

public void OnResultExecuting(ResultExecutingContext context)

{

var serviceProviders = context.HttpContext.RequestServices.CreateScope().ServiceProvider;

\_logger = serviceProviders.GetService<ILoggerProvider>()

.CreateLogger("JsonEncryptResultFilter");

\_gronsfeldService = serviceProviders.GetService<GronsfeldService>();

\_memoryCache = serviceProviders.GetService<IMemoryCache>();

var objResult = (ObjectResult) context.Result;

var key = \_memoryCache.Get<string>(context.HttpContext.GetEmail());

var encryptData = JsonConvert.SerializeObject(objResult.Value);

var resultData = \_gronsfeldService.Encrypt(encryptData, key);

context.Result = new ContentResult()

{

Content = resultData,

StatusCode = 200,

ContentType = "text/plain"

};

\_logger.LogInformation($"Конец результата");

}

**Клиентская часть**

[Authorize]

[HttpPost("SetKey")]

public async Task SetKey(string key)

{

var email = HttpContext.User.FindFirst(ClaimTypes.Email).Value;

await \_userClient.SetKey(email, key);

}

//Установка ключа

public async Task SetKey(string email, string key)

{

if (!key.All(char.IsDigit))

throw new ArgumentException("Ключ должен состоять только из чисел");

var result = await \_httpClient.PostAsync($"user/key?key={key}", null);

result.EnsureSuccessStatusCode();

var model = \_memoryCache.Get<CacheKeyCookieModel>(email);

model.Key = key;

\_memoryCache.Set(email, model);

}

/// <summary>

/// Создание новости

/// </summary>

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

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

/// <returns></returns>

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

{

var keyModel = \_memoryCache.Get<CacheKeyCookieModel>(email);

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

var json = JsonConvert.SerializeObject(model);

\_logger.LogInformation($"Отправляемая модель:{json}");

var encryptStr = \_gronsfeldService.Encrypt(json, keyModel.Key);

var res = await \_httpClient.PostAsync($"news?data={encryptStr}", null);

var content = await res.Content.ReadAsStringAsync();

\_logger.LogInformation($"Ответ с сервера:{content}");

var afterDecrypt = \_gronsfeldService.Decrypt(content, keyModel.Key);

\_logger.LogInformation($"После дешефровки:{content}");

return JsonConvert.DeserializeObject<Guid>(afterDecrypt);

}

public class CacheKeyCookieModel

{

public static string GetCookieName = "Set-Cookie";

public static string SetCookie = "Cookie";

public string Key { get; set; }

public IEnumerable<string> Cookie { get; set; }

/// <inheritdoc />

public CacheKeyCookieModel(string key, IEnumerable<string> cookie)

{

Key = key;

Cookie = cookie;

}

}