Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.04 – «Программная инженерия»

Профиль: «Разработка программно-информационных систем»

Дисциплина: «Защита информации»

Семестр 5

ОТЧЕТ

по лабораторной работе №3

Тема: «Алгоритм Эль-Гамаля»

Вариант 25

|  |  |  |
| --- | --- | --- |
|  | Выполнил: Студент группы РИС-23-2б  Д.Е. Колосов | |
|  |  | подпись, дата |
|  | Проверил: Ст. преподаватель кафедры ИТАС  В. Г. Шереметьев | |
|  |  | подпись, дата |

Пермь, 2025

**Цель работы**

Получить практические навыки по использованию несимметричных алгоритмов шифрования, на примере использования алгоритма Эль-Гамаля.

**Задание**

Выполнить шифрование текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамаля, используя в качестве x и g простые числа с разрядностью не меньшей двенадцати и p и k не менее двадцати, выполнив условие случайности x и g для каждого нового шифрования.

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Основные принципы работы

Шифр Эль-Гамаля — это алгоритм с открытым ключом, основанный на проблеме дискретного логарифмирования. Он используется как для шифрования, так и для формирования цифровой подписи. Мы рассмотрим часть, связанную с шифрованием.

1. Основные концепции

Асимметричное шифрование:

Используется два ключа: открытый (для шифрования) и закрытый (для дешифрования).

Проблема дискретного логарифма:

Если есть уравнение y = g^x mod p, то при известных y, g и p вычислить x является вычислительно сложной задачей для больших простых чисел p. Это основа безопасности Эль-Гамаля.

2. Генерация ключей

Перед шифрованием необходимо сгенерировать пару ключей.

Выбор общего параметра (p) - выбирается большое простое число p.

По заданию: разрядность p не менее 20 бит. (Прим.: обычно на практике используют 2048 бит и более, 20 бит — это очень мало для реальной безопасности, но достаточно для учебного задания).

Выбор генератора (g) - выбирается число g — первообразный корень по модулю p. Это такое число, что его степени по модулю p будут давать все числа от 1 до p-1.

По заданию: g — простое число, разрядностью не менее 12 бит.

Выбор закрытого ключа (x) - выбирается случайное число x, такое что 1 < x < p-1.

По заданию: x — простое число, разрядностью не менее 12 бит.

Вычисление открытого ключа (y) - вычисляется y = g^x mod p.

Открытый ключ (public key): тройка (p, g, y). Ее можно свободно передавать.

Закрытый ключ (private key): число x. Его хранят в секрете.

3. Процесс шифрования

Предположим, отправляется сообщение M.

Представление сообщения - сообщение M должно быть числом, меньшим p. Если сообщение длинное, его разбивают на блоки, каждый из которых представляется числом в промежутке [0, p-1].

Выбор случайного сессионного ключа (k) - отправитель выбирает случайное число k, такое что 1 < k < p-1. Это число должно быть уникальным для каждого сеанса шифрования (важное условие безопасности!).

По заданию: k — случайное число, не менее 20 бит.

Шифротекст представляет собой пару чисел (a, b), вычисляемых по формулам:

a = g^k mod p

b = M \* y^k mod p

Отправитель передает получателю пары чисел (a, b) – зашифрованное сообщение.

4. Процесс дешифрования

Получив пару (a, b) и имея свой закрытый ключ x, получатель вычисляет исходное сообщение:

Вычисляет s = a^x mod p. Заметим, что s = (g^k)^x = g^(kx) mod p. Отправитель, при вычислении b, использовал y^k = (g^x)^k = g^(xk) = s. Таким образом, и отправитель, и получатель используют одно и то же значение s.

Нахождение обратного элемента: получатель вычисляет s^(-1) — обратный элемент к s по модулю p. Это такое число, что s \* s^(-1) ≡ 1 (mod p). Для нахождения можно использовать расширенный алгоритм Евклида.

Восстановление сообщения - исходное сообщение вычисляется как M = b \* s^(-1) mod p.

Почему это работает:

b \* s^(-1) ≡ (M \* y^k) \* (a^x)^(-1) ≡ (M \* (g^x)^k) \* ((g^k)^x)^(-1) ≡ (M \* g^(xk)) \* (g^(xk))^(-1) ≡ M (mod p)

Реализация:

Сгенерировать простое число p длиной не менее 20 бит (например, используя тест Миллера-Рабина).

Сгенерировать простое число g длиной не менее 12 бит, которое будет первообразным корнем по модулю p.

Сгенерировать простое число x длиной не менее 12 бит, где 1 < x < p-1.

Вычислить y = g^x mod p.

Преобразовать сообщение в последовательность байт (в кодировке UTF-8).

Разбить эту последовательность на блоки числового представления. Размер блока должен быть таким, чтобы максимальное число, представляемое блоком, было меньше p. Если p имеет длину 20 бит (~1 миллион), вы можно взять, например, блоки по 2 байта (макс. значение 65535).

Шифрование:

Сгенерировать новое случайное число k (не менее 20 бит), удовлетворяющее условиям.

Для каждого блока сообщения M\_i:

Вычислить a\_i = g^k mod p.

Вычислить b\_i = M\_i \* y^k mod p.

Пара (a\_i, b\_i) является шифротекстом для блока M\_i.

Полный шифротекст — это последовательность пар (a\_1, b\_1), (a\_2, b\_2), ..., (a\_n, b\_n).

**ХОД РАБОТЫ**

Стоит обратить внимание на зависимость длины шифруемого сообщения от длины ключа – от того, сколько разрядов выделено на число p. Она выражается следующей формулой:

Макс. размер сообщения (байты) = (битовая\_длина\_p / 8)

Таким образом, если на p выделить 20 бит (минимум по заданию), то максимальная длина шифруемого сообщения будет 2 байта (8 бит). Каждая русская буква занимает 2 байта, английская – 1. Цифры 0-9 занимают 1 байт каждая.

Чтобы шифровать данным методом большие сообщения можно увеличить разрядность числа p, либо разбить текст сообщения на блоки допустимой для шифрования малым p длины и шифровать сообщение поблочно.

Главное окно программы представлено на рисунке 1.

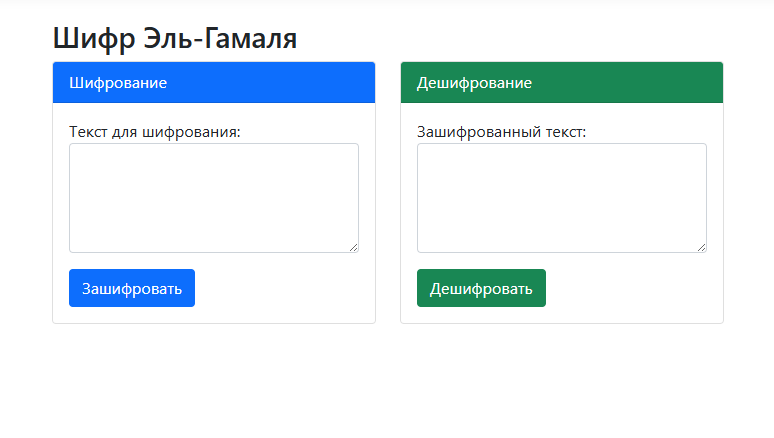


Рисунок 1 – главное окно программы

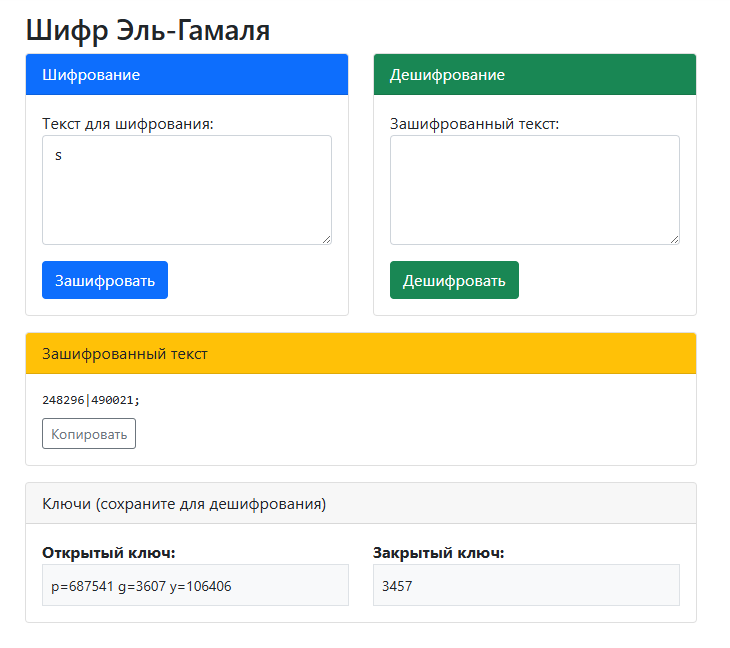


Рисунок 2 – шифрование одной латинской буквы – один блок (если шифровать каждый байт)

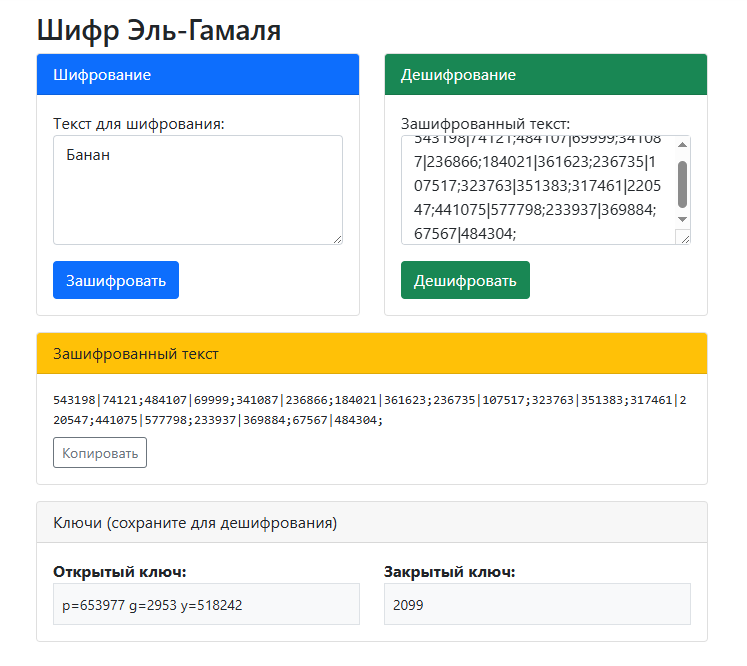


Рисунок 3 – шифрование слова на кириллице

Используя те же ключи, можно выполнить расшифровку данных.

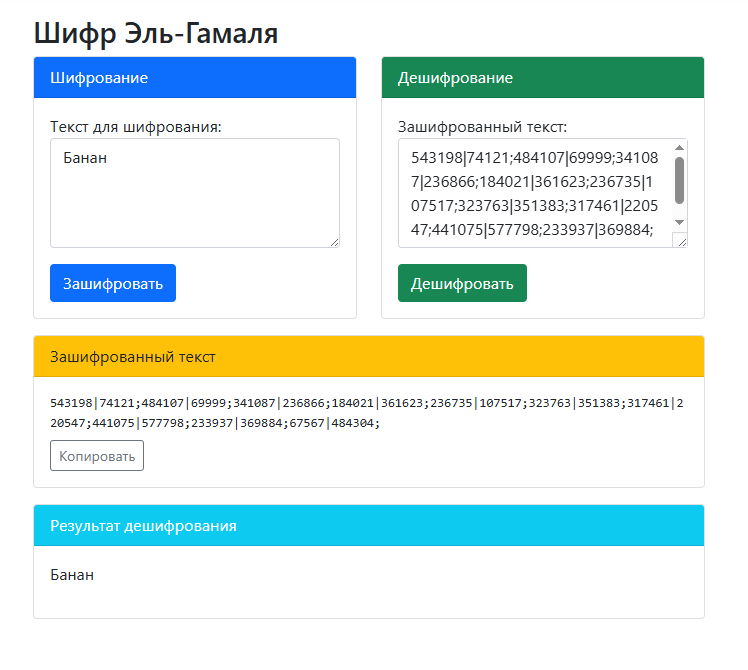


Рисунок 4 – расшифрование сообщения

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

Index.cshml

@page

@model IndexModel

<div class="container mt-4">

<h2>Шифр Эль-Гамаля</h2>

<form method="post">

<div class="row">

<!-- Левая часть - Шифрование -->

<div class="col-md-6">

<div class="card">

<div class="card-header bg-primary text-white">

Шифрование

</div>

<div class="card-body">

<div class="mb-3">

<label>Текст для шифрования:</label>

<textarea class="form-control" asp-for="Model.InputText" rows="4"></textarea>

</div>

<button type="submit" class="btn btn-primary" name="action" value="encrypt">

Зашифровать

</button>

</div>

</div>

</div>

<!-- Правая часть - Дешифрование -->

<div class="col-md-6">

<div class="card">

<div class="card-header bg-success text-white">

Дешифрование

</div>

<div class="card-body">

<div class="mb-3">

<label>Зашифрованный текст:</label>

<textarea class="form-control" asp-for="Model.EncryptedText" rows="4"></textarea>

</div>

<button type="submit" class="btn btn-success" name="action" value="decrypt">

Дешифровать

</button>

</div>

</div>

</div>

</div>

</form>

<!-- Зашифрованный текст -->

@if (!string.IsNullOrEmpty(Model.Model.EncryptedText))

{

<div class="card mt-3">

<div class="card-header bg-warning">

Зашифрованный текст

</div>

<div class="card-body">

<div style="max-height: 200px; overflow-y: auto; font-family: monospace; font-size: 0.8em;">

@Model.Model.EncryptedText

</div>

<button type="button" class="btn btn-sm btn-outline-secondary mt-2" onclick="copyToClipboard('encryptedText')">

Копировать

</button>

</div>

</div>

}

<!-- Ключи -->

@if (!string.IsNullOrEmpty(Model.Model.PublicKey))

{

<div class="card mt-3">

<div class="card-header">

Ключи (сохраните для дешифрования)

</div>

<div class="card-body">

<div class="row">

<div class="col-md-6">

<strong>Открытый ключ:</strong>

<div class="border p-2 bg-light">

<small>@Model.Model.PublicKey</small>

</div>

</div>

<div class="col-md-6">

<strong>Закрытый ключ:</strong>

<div class="border p-2 bg-light">

<small>@Model.Model.PrivateKey</small>

</div>

</div>

</div>

</div>

</div>

}

@if (!string.IsNullOrEmpty(Model.Model.DecryptedText))

{

<div class="card mt-3">

<div class="card-header bg-info text-white">

Результат дешифрования

</div>

<div class="card-body">

<p>@Model.Model.DecryptedText</p>

</div>

</div>

}

@if (!string.IsNullOrEmpty(Model.Model.Error))

{

<div class="alert alert-danger mt-3">

@Model.Model.Error

</div>

}

</div>

@section Scripts {

<script>

function copyToClipboard() {

const text = `@Model.Model.EncryptedText`;

navigator.clipboard.writeText(text).then(() => {

alert('Текст скопирован!');

});

}

</script>

}

Index.cshtml.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.RazorPages;

using ElGamalApp.Models;

using ElGamalApp.Services;

namespace ElGamalApp.Pages

{

public class IndexModel : PageModel

{

private readonly IElGamalService \_elGamalService;

private static ElGamalKeys \_currentKeys;

public IndexModel(IElGamalService elGamalService)

{

\_elGamalService = elGamalService;

}

[BindProperty]

public AppModel Model { get; set; } = new AppModel();

public string KeySizes { get; set; } = string.Empty;

public void OnGet()

{

KeySizes = \_elGamalService.GetKeySizes();

}

public IActionResult OnPost(string action)

{

KeySizes = \_elGamalService.GetKeySizes();

try

{

if (action == "encrypt" && !string.IsNullOrEmpty(Model.InputText))

{

// Генерируем новые ключи при каждом шифровании

\_currentKeys = \_elGamalService.GenerateKeys();

// Шифруем текст

Model.EncryptedText = \_elGamalService.Encrypt(Model.InputText, \_currentKeys);

// Сохраняем ключи для показа

Model.PublicKey = $"p={\_currentKeys.P}\ng={\_currentKeys.G}\ny={\_currentKeys.Y}";

Model.PrivateKey = \_currentKeys.X.ToString();

}

else if (action == "decrypt" && !string.IsNullOrEmpty(Model.EncryptedText) && \_currentKeys != null)

{

// Дешифруем текст текущими ключами

Model.DecryptedText = \_elGamalService.Decrypt(Model.EncryptedText, \_currentKeys);

}

}

catch (Exception ex)

{

Model.Error = $"Ошибка: {ex.Message}";

}

return Page();

}

}

}