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 №2

Тема: «Алгоритм RSA»

|  |  |  |
| --- | --- | --- |
|  | 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 ассиметричных алгоритмов шифрования, на примере использования алгоритма RSA.

**Задание**

Выполнить шифрование строки исходного текста, методом RSA, используя в качестве p и q простые числа с разрядностью не меньшей двенадцати, выполнив условие случайности p и q для каждого нового шифрования (вариант 7, место в списке 15).

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Основные принципы работы RSA

Сначала пара математических определений. Целое число называют простым, если оно делится нацело только на единицу и на само себя, иначе его называют составным.

Два целых числа называют взаимно простым, если их наибольший общий делитель (НОД) равен 1.

Алгоритм работы RSA таков. Сначала надо получить открытый и секретный ключи:

1. Выбираются два простых числа p и q

2. Вычисляется их произведение n(=p\*q)

3. Выбирается произвольное число e (e<n), такое, что НОД(e,(p-1)(q-1))=1, то есть e должно быть взаимно простым с числом (p-1)(q-1).

4. Методом Евклида решается в целых числах уравнение e\*d+(p-1)(q-1)\*y=1. Здесь неизвестными являются переменные d и y – метод Евклида как раз и находит множество пар (d,y), каждая из которых является решением уравнения в целых числах.

5. Два числа (e,n) – публикуются как открытый ключ.

6. Число d хранится в строжайшем секрете – это и есть закрытый ключ, который позволит читать все послания, зашифрованные с помощью пары чисел (e,n).

Как же производится собственно шифрование с помощью этих чисел:

1. Отправитель разбивает свое сообщение на блоки, равные k=[log2(n)] бит, где квадратные скобки обозначают взятие целой части от дробного числа.

2. Подобный блок, как Вы знаете, может быть интерпретирован как число из диапазона (0;2k-1). Для каждого такого числа (назовем его mi) вычисляется выражение ci=((mi)e)mod n. Блоки ci и есть зашифрованное сообщение

То есть для того чтобы прочесть сообщение ci=((mi)e)mod n достаточно возвести его в степень d по модулю m : ((ci)d)mod n = ((mi)e\*d)mod n = mi.

**ХОД РАБОТЫ**

Стоит обратить внимание на зависимость длины шифруемого сообщения от длины ключей – от того, сколько разрядов выделено на числа p и q. Она выражается следующей формулой:

Макс. размер сообщения (байты) = (битовая\_длина\_n / 8) – 1

Таким образом, если на p и q выделить 12 бит (разрядность 12), то n будет иметь разрядность 24 (2^12\*2^12=2^24, n=p\*q) и максимальная длина шифруемого сообщения будет 2 байта (8 бит). Каждая русская буква занимает 2 байта, английская – 1. Цифры 0-9 занимают 1 байт каждая. Таким образом можно будет зашифровать максимум 1 букву русского или две английского алфавитов, либо 2 цифры.

Чтобы увеличить длину возможного сообщения, в программе выделяется на p и q по 128 бит.

Математическая база:

Мы специально выбираем d так, чтобы:

e × d ≡ 1 (mod φ(n))

А по определению сравнения по модулю:

e × d ≡ 1 (mod φ(n)) ⇔ e × d = 1 + k × φ(n) (для некоторого целого k)

Получается:

encrypted = ((message)^e) mod n.

(encrypted)^d mod n = (message^e)^d mod n

(message^e)^d mod n = message^(e × d) mod n

Подставляем e × d:

= message^(1 + k × φ(n)) mod n

Разбиваем степень:

Степень 1 + k × φ(n) можно разбить:

= message^1 × message^(k × φ(n)) mod n

Переписываем:

= message × (message^φ(n))^k mod n

Теорема Эйлера в действии. Теорема Эйлера говорит:

Если message и n взаимно просты, то: message^φ(n) mod n = 1

Подставляем волшебную единицу:

= message × (1)^k mod n

= message × 1 mod n

= message mod n

= message (поскольку message < n)

Главное окно программы представлено на рисунке 1. Перед выполнением шифрования необходимо сгенерировать ключи – выбрать p и q, вычислить n.

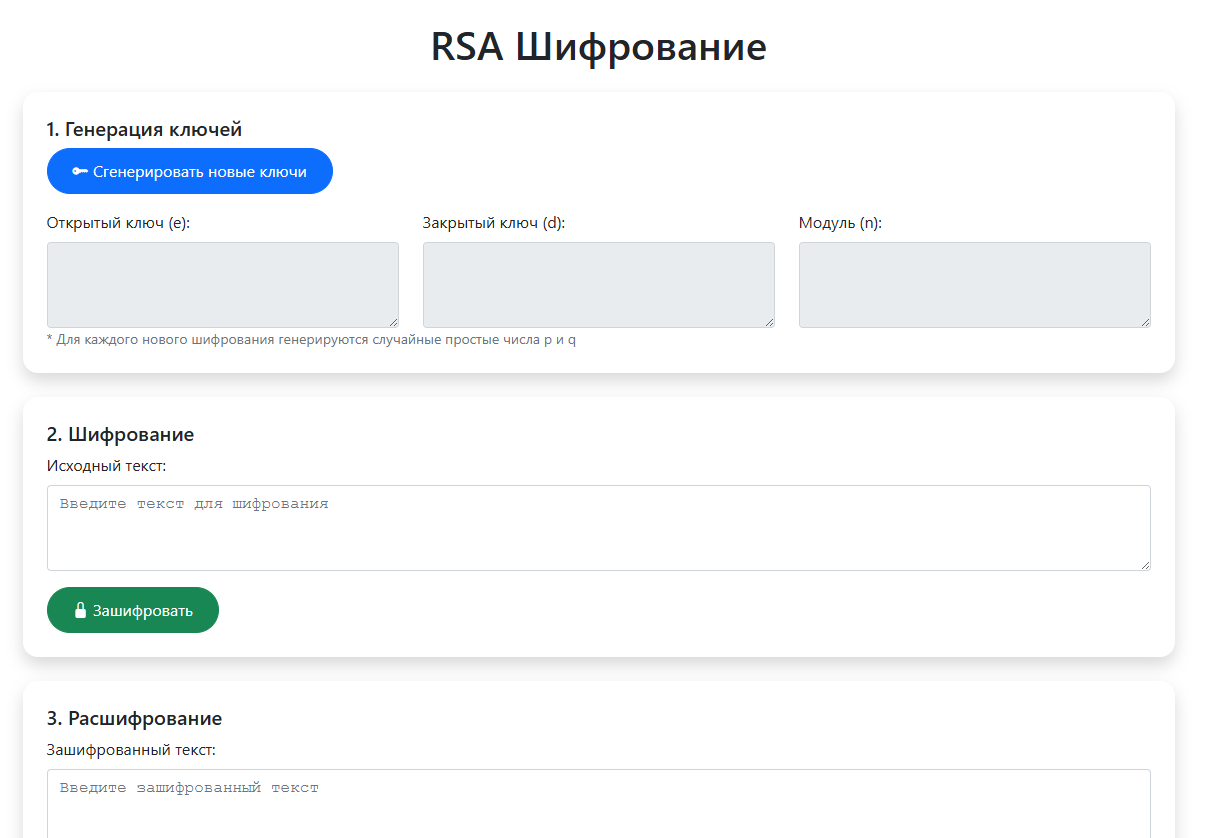


Рисунок 1 – главное окно программы

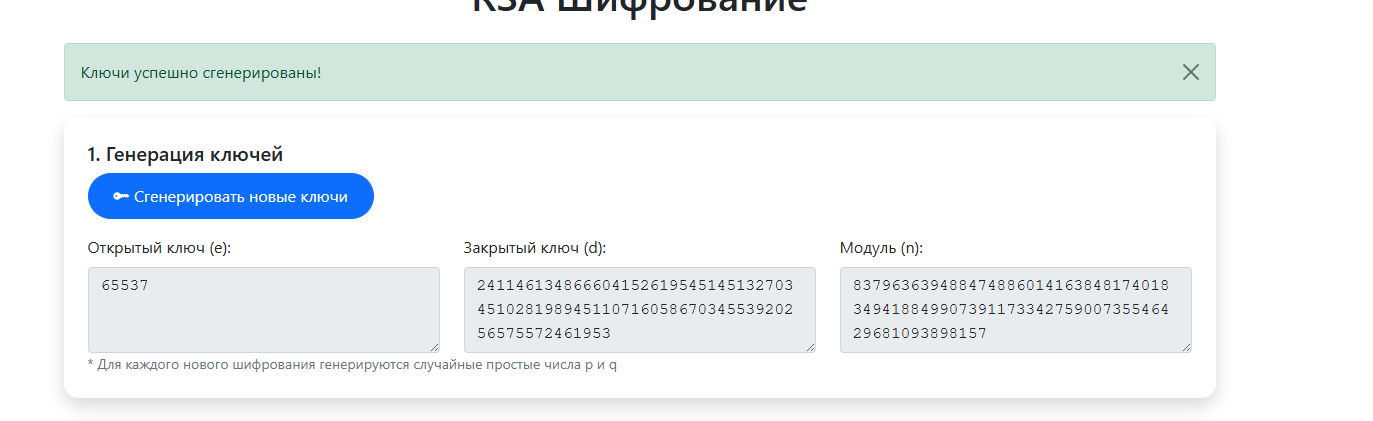


Рисунок 2 – генерация ключей

Далее необходимо ввести текст для шифрования и зашифровать его.

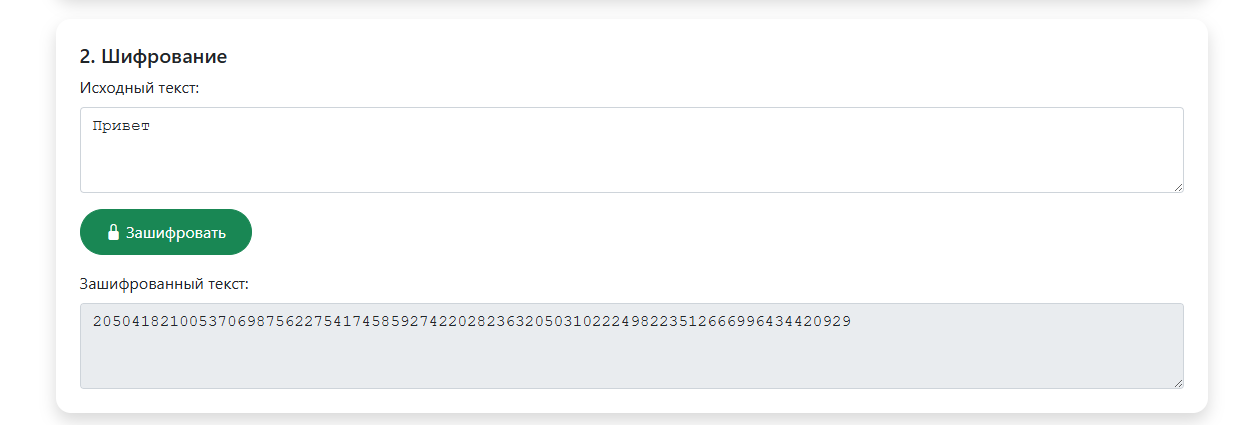


Рисунок 3 – шифрование

Используя те же ключи, можно выполнить расшифровку данных.

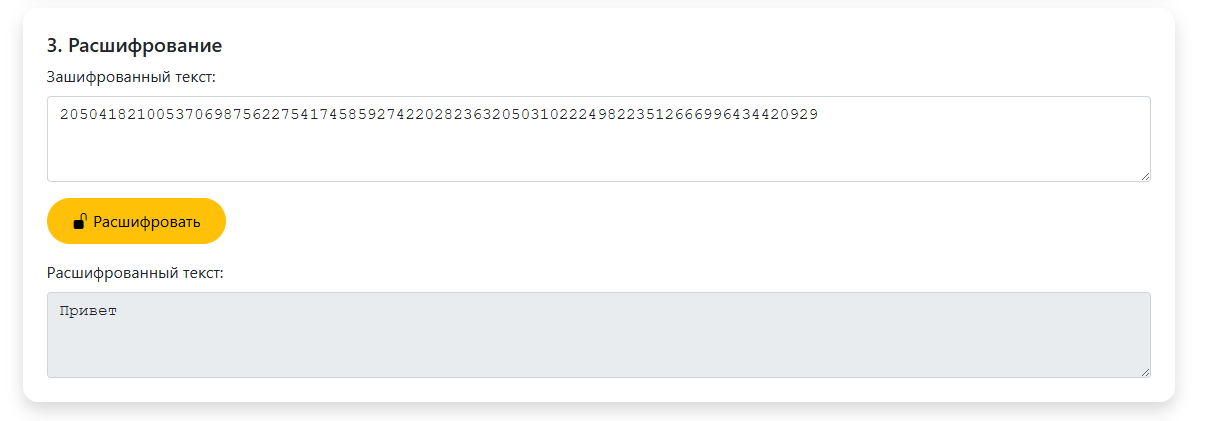


Рисунок 4 – расшифрование

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

Index.xshml

@page

@model RSACipher.Pages.IndexModel

@{

ViewData["Title"] = "RSA Шифрование";

}

<div class="container mt-5">

<h1 class="text-center mb-4">RSA Шифрование</h1>

@if (TempData["Message"] != null)

{

<div class="alert alert-success alert-dismissible fade show" role="alert">

@TempData["Message"]

<button type="button" class="btn-close" data-bs-dismiss="alert"></button>

</div>

}

@if (TempData["Error"] != null)

{

<div class="alert alert-danger alert-dismissible fade show" role="alert">

@TempData["Error"]

<button type="button" class="btn-close" data-bs-dismiss="alert"></button>

</div>

}

<!-- Генерация ключей -->

<div class="card p-4 shadow mb-4">

<h5 class="card-title">1. Генерация ключей</h5>

<form method="post" class="mb-3">

<button type="submit" asp-page-handler="GenerateKeys" class="btn btn-primary">

<i class="bi bi-key-fill"></i> Сгенерировать новые ключи

</button>

</form>

<div class="row">

<div class="col-md-4">

<label class="form-label">Открытый ключ (e):</label>

<textarea class="form-control" rows="3" readonly>@Model.PublicKey</textarea>

</div>

<div class="col-md-4">

<label class="form-label">Закрытый ключ (d):</label>

<textarea class="form-control" rows="3" readonly>@Model.PrivateKey</textarea>

</div>

<div class="col-md-4">

<label class="form-label">Модуль (n):</label>

<textarea class="form-control" rows="3" readonly>@Model.Modulus</textarea>

</div>

</div>

<small class="text-muted">\* Для каждого нового шифрования генерируются случайные простые числа p и q</small>

</div>

<!-- Шифрование -->

<div class="card p-4 shadow mb-4">

<h5 class="card-title">2. Шифрование</h5>

<form method="post">

<div class="mb-3">

<label for="InputText" class="form-label">Исходный текст:</label>

<textarea class="form-control" id="InputText" name="InputText" rows="3"

placeholder="Введите текст для шифрования">@Model.InputText</textarea>

</div>

<button type="submit" asp-page-handler="Encrypt" class="btn btn-success">

<i class="bi bi-lock-fill"></i> Зашифровать

</button>

</form>

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

{

<div class="mt-3">

<label class="form-label">Зашифрованный текст:</label>

<textarea class="form-control" rows="3" readonly>@Model.EncryptedText</textarea>

</div>

}

</div>

<!-- Расшифрование -->

<div class="card p-4 shadow">

<h5 class="card-title">3. Расшифрование</h5>

<form method="post">

<div class="mb-3">

<label class="form-label">Зашифрованный текст:</label>

<textarea class="form-control" name="EncryptedText" rows="3"

placeholder="Введите зашифрованный текст">@Model.EncryptedText</textarea>

</div>

<button type="submit" asp-page-handler="Decrypt" class="btn btn-warning">

<i class="bi bi-unlock-fill"></i> Расшифровать

</button>

</form>

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

{

<div class="mt-3">

<label class="form-label">Расшифрованный текст:</label>

<textarea class="form-control" rows="3" readonly>@Model.DecryptedText</textarea>

</div>

}

</div>

<!-- Информация о алгоритме -->

<div class="card mt-4 p-4 shadow">

<h5 class="card-title">Информация о RSA</h5>

<ul class="list-unstyled">

<li><strong>p и q:</strong> Случайные простые числа (≤128 бит)</li>

<li><strong>n = p × q:</strong> Модуль</li>

<li><strong>φ(n) = (p-1)(q-1):</strong> Функция Эйлера</li>

<li><strong>e:</strong> Открытая экспонента (обычно 65537 - простое число, подходит в большинстве вариантов)</li>

<li><strong>d = e⁻¹ mod φ(n):</strong> Закрытая экспонента</li>

</ul>

</div>

</div>

<style>

.container {

max-width: 1200px;

}

.card {

border: none;

border-radius: 15px;

}

.btn {

border-radius: 25px;

padding: 10px 25px;

}

textarea {

font-family: 'Courier New', monospace;

font-size: 0.9em;

}

</style>

Index.cshtml.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.RazorPages;

using System.Numerics;

using System.Security.Cryptography;

using System.Text;

namespace RSACipher.Pages

{

public class IndexModel : PageModel

{

[BindProperty]

public string InputText { get; set; } = "";

[BindProperty]

public string PublicKey { get; set; } = "";

[BindProperty]

public string PrivateKey { get; set; } = "";

[BindProperty]

public string Modulus { get; set; } = "";

[BindProperty]

public string EncryptedText { get; set; } = "";

[BindProperty]

public string DecryptedText { get; set; } = "";

// Добавляем временное хранение ключей в TempData

private const string PublicKeyTemp = "PublicKey";

private const string PrivateKeyTemp = "PrivateKey";

private const string ModulusTemp = "Modulus";

public void OnGet()

{

// Восстанавливаем ключи из TempData при загрузке страницы

PublicKey = TempData[PublicKeyTemp]?.ToString() ?? "";

PrivateKey = TempData[PrivateKeyTemp]?.ToString() ?? "";

Modulus = TempData[ModulusTemp]?.ToString() ?? "";

// Сохраняем обратно в TempData для следующего запроса

TempData.Keep(PublicKeyTemp);

TempData.Keep(PrivateKeyTemp);

TempData.Keep(ModulusTemp);

}

public IActionResult OnPostGenerateKeys()

{

try

{

// Генерация простых чисел p и q разрядностью не менее 12 бит

BigInteger p = GenerateLargePrime(128);//12 мин

BigInteger q = GenerateLargePrime(128);//12 мин

while (p == q)

{

q = GenerateLargePrime(128);

}

BigInteger n = p \* q;

BigInteger phi = (p - 1) \* (q - 1);

// Выбор открытой экспоненты e

BigInteger e = 65537;

// Проверяем, что e взаимно просто с phi

while (BigInteger.GreatestCommonDivisor(e, phi) != 1)

{

e = GenerateLargePrime(16);

}

// Вычисление секретной экспоненты d

BigInteger d = ModInverse(e, phi);

// Сохраняем ключи в свойствах и TempData

PublicKey = e.ToString();

PrivateKey = d.ToString();

Modulus = n.ToString();

TempData[PublicKeyTemp] = PublicKey;

TempData[PrivateKeyTemp] = PrivateKey;

TempData[ModulusTemp] = Modulus;

TempData["Message"] = "Ключи успешно сгенерированы!";

}

catch (Exception ex)

{

TempData["Error"] = $"Ошибка при генерации ключей: {ex.Message}";

}

return Page();

}

public IActionResult OnPostEncrypt()

{

try

{

// Восстанавливаем ключи из TempData

PublicKey = TempData[PublicKeyTemp]?.ToString() ?? "";

PrivateKey = TempData[PrivateKeyTemp]?.ToString() ?? "";

Modulus = TempData[ModulusTemp]?.ToString() ?? "";

if (string.IsNullOrEmpty(InputText) || string.IsNullOrEmpty(PublicKey) || string.IsNullOrEmpty(Modulus))

{

TempData["Error"] = "Введите текст и сгенерируйте ключи!";

// Сохраняем ключи обратно

TempData.Keep(PublicKeyTemp);

TempData.Keep(PrivateKeyTemp);

TempData.Keep(ModulusTemp);

return Page();

}

BigInteger e = BigInteger.Parse(PublicKey);

BigInteger n = BigInteger.Parse(Modulus);

//Или нужно всегда новые генерировать?

//if (string.IsNullOrEmpty(InputText))

//{

// TempData["Error"] = "Введите текст!";

// return Page();

//}

//BigInteger p = GenerateLargePrime(128);

//BigInteger q = GenerateLargePrime(128);

//while (p == q)

//{

// q = GenerateLargePrime(128);

//}

//BigInteger n = p \* q;

//BigInteger phi = (p - 1) \* (q - 1);

//BigInteger e = 65537;

//if (BigInteger.GreatestCommonDivisor(e, phi) != 1)

//{

// TempData["Error"] = "Не удалось подобрать p и q. Попробуйте еще раз.";

// return Page();

//}

//BigInteger d = ModInverse(e, phi);

//// Сохраняем ключи в свойствах и TempData

//PublicKey = e.ToString();

//PrivateKey = d.ToString();

//Modulus = n.ToString();

//TempData[PublicKeyTemp] = PublicKey;

//TempData[PrivateKeyTemp] = PrivateKey;

//TempData[ModulusTemp] = Modulus;

//TempData["Message"] = "Ключи успешно сгенерированы!";

// Преобразуем текст в байты

byte[] bytes = Encoding.UTF8.GetBytes(InputText);

// Преобразуем байты в большое число

BigInteger message = new BigInteger(bytes, isUnsigned: true);

// Проверяем, что сообщение меньше модуля

if (message >= n)

{

TempData["Error"] = "Сообщение слишком большое для выбранного ключа!";

TempData.Keep(PublicKeyTemp);

TempData.Keep(PrivateKeyTemp);

TempData.Keep(ModulusTemp);

return Page();

}

// Шифруем: c = m^e mod n

BigInteger encrypted = BigInteger.ModPow(message, e, n);

EncryptedText = encrypted.ToString();

TempData["Message"] = "Текст успешно зашифрован!";

// Сохраняем ключи обратно

TempData.Keep(PublicKeyTemp);

TempData.Keep(PrivateKeyTemp);

TempData.Keep(ModulusTemp);

}

catch (Exception ex)

{

TempData["Error"] = $"Ошибка при шифровании: {ex.Message}";

TempData.Keep(PublicKeyTemp);

TempData.Keep(PrivateKeyTemp);

TempData.Keep(ModulusTemp);

}

return Page();

}

public IActionResult OnPostDecrypt()

{

try

{

// Восстанавливаем ключи из TempData

PublicKey = TempData[PublicKeyTemp]?.ToString() ?? "";

PrivateKey = TempData[PrivateKeyTemp]?.ToString() ?? "";

Modulus = TempData[ModulusTemp]?.ToString() ?? "";

if (string.IsNullOrEmpty(EncryptedText) || string.IsNullOrEmpty(PrivateKey) || string.IsNullOrEmpty(Modulus))

{

TempData["Error"] = "Введите зашифрованный текст и убедитесь, что ключи сгенерированы!";

TempData.Keep(PublicKeyTemp);

TempData.Keep(PrivateKeyTemp);

TempData.Keep(ModulusTemp);

return Page();

}

BigInteger d = BigInteger.Parse(PrivateKey);

BigInteger n = BigInteger.Parse(Modulus);

BigInteger encrypted = BigInteger.Parse(EncryptedText);

// Дешифруем: m = c^d mod n

BigInteger decrypted = BigInteger.ModPow(encrypted, d, n);

// Преобразуем большое число обратно в байты

byte[] decryptedBytes = decrypted.ToByteArray(isUnsigned: true);

// Если последний байт равен 0 (из-за особенностей BigInteger), удаляем его

if (decryptedBytes.Length > 0 && decryptedBytes[decryptedBytes.Length - 1] == 0)

{

decryptedBytes = decryptedBytes.Take(decryptedBytes.Length - 1).ToArray();

}

DecryptedText = Encoding.UTF8.GetString(decryptedBytes);

TempData["Message"] = "Текст успешно расшифрован!";

// Сохраняем ключи обратно

TempData.Keep(PublicKeyTemp);

TempData.Keep(PrivateKeyTemp);

TempData.Keep(ModulusTemp);

}

catch (Exception ex)

{

TempData["Error"] = $"Ошибка при расшифровании: {ex.Message}";

TempData.Keep(PublicKeyTemp);

TempData.Keep(PrivateKeyTemp);

TempData.Keep(ModulusTemp);

}

return Page();

}

// Генерация большого простого числа

private BigInteger GenerateLargePrime(int bitLength)

{

using (RandomNumberGenerator rng = RandomNumberGenerator.Create())

{

while (true)

{

byte[] bytes = new byte[bitLength / 8 + 1];

rng.GetBytes(bytes);

bytes[bytes.Length - 1] &= 0x7F; // Убедимся, что число положительное

BigInteger candidate = new BigInteger(bytes);

// Убедимся, что число нечетное и достаточно большое

if (candidate.IsEven)

candidate++;

if (candidate < BigInteger.Pow(2, bitLength - 1))

candidate += BigInteger.Pow(2, bitLength - 1);

if (IsProbablePrime(candidate))

return candidate;

}

}

}

// Тест Миллера-Рабина на простоту

private bool IsProbablePrime(BigInteger n, int certainty = 100)

{

if (n == 2 || n == 3)

return true;

if (n < 2 || n.IsEven)

return false;

BigInteger d = n - 1;

int s = 0;

while (d.IsEven)

{

d /= 2;

s++;

}

using (RandomNumberGenerator rng = RandomNumberGenerator.Create())

{

byte[] bytes = new byte[n.ToByteArray().LongLength];

for (int i = 0; i < certainty; i++)

{

BigInteger a;

do

{

rng.GetBytes(bytes);

a = new BigInteger(bytes);

}

while (a < 2 || a >= n - 1);

BigInteger x = BigInteger.ModPow(a, d, n);

if (x == 1 || x == n - 1)

continue;

for (int r = 1; r < s; r++)

{

x = BigInteger.ModPow(x, 2, n);

if (x == 1)

return false;

if (x == n - 1)

break;

}

if (x != n - 1)

return false;

}

}

return true;

}

// Вычисление модульного обратного с помощью расширенного алгоритма Евклида

private BigInteger ModInverse(BigInteger a, BigInteger m)

{

BigInteger m0 = m;

BigInteger y = 0, x = 1;

if (m == 1)

return 0;

while (a > 1)

{

BigInteger q = a / m;

BigInteger t = m;

m = a % m;

a = t;

t = y;

y = x - q \* y;

x = t;

}

if (x < 0)

x += m0;

return x;

}

}

}