УДК 004.056.5
DOI: 10.26102/2310-6018/2023.41.2.023

Разработка декоратора StegoStream для ассоциативной защиты байтового потока

Р.Ф. Гибадуллин✉, Д.А. Гашигуллин, И.С. Вершинин

Казанский национальный исследовательский технический университет им. А.Н. Туполева–КАИ, Казань, Российская Федерация
landwatersun@mail.ru✉

Резюме. Потоковая архитектура .NET основана на трех концепциях: опорные хранилища, декораторы и адаптеры. Опорное хранилище представляет собой конечную точку, такую как файл на накопителе, массив в оперативной памяти или сетевое подключение. Опорное хранилище не может использоваться, если программисту не открыт к нему доступ. Стандартным классом .NET, который предназначен для такой цели, является Stream (поток); он предоставляет стандартный набор методов, позволяющих выполнять побайтовое чтение, запись и позиционирование. Потоки делятся на две категории: потоки с опорными хранилищами и потоки с декораторами. Потоки с опорными хранилищами и потоки с декораторами имеют дело исключительно с байтами. Это гибко и эффективно, однако приложения часто работают на более высоких уровнях, таких как текст или XML. Адаптеры преодолевают такой разрыв, помещая поток в оболочку класса со специализированными методами, которые типизированы для конкретного формата. В статье представлен разработанный авторами декоратор StegoStream, основанный на ассоциативном механизме защиты данных. Данный декоратор обладает следующими преимуществами: он обеспечивает взаимодействие с адаптером; освобождает потоки с опорными хранилищами от необходимости самостоятельной реализации таких возможностей, как сокрытие и раскрытие; потоки, декорированные StegoStream, не страдают от изменения интерфейса; StegoStream можно использовать при соединении в цепочки с другими декораторами (например, декоратор сжатия можно соединить с декоратором сокрытия). Практическое использование декоратора StegoStream представлено на примере разработанного мультиклиентного защищенного чата с централизованным сервером.

Ключевые слова: ассоциативная стеганография, криптография, потоковая архитектура, декоратор, информационная безопасность.

Для цитирования: Гибадуллин Р.Ф., Гашигуллин Д.А., Вершинин И.С. Разработка декоратора StegoStream для ассоциативной защиты байтового потока. *Моделирование, оптимизация и информационные технологии.* 2023;11(2). URL: https://moitvivt.ru/ru/journal/pdf?id=1359 DOI: 10.26102/2310-6018/2023.41.2.023

Development of StegoStream decorator for associative protection of byte stream

R.F. Gibadullin✉, D.A. Gashigullin, I.S. Vershinin

Kazan National Research Technical University named after A.N. Tupolev–KAI, Kazan, the Russian Federation
landwatersun@mail.ru✉

Abstract. The .NET streaming architecture is based on three concepts: reference repositories, decorators, and adapters. A reference repository represents an endpoint, such as a file on a storage device, an array in RAM, or a network connection. It cannot be used unless the programmer has access to it. The standard .NET class designed for this purpose is Stream; it provides a standard set of methods that allow byte-by-byte reading, writing, and positioning. Streams fall into two categories: streams with reference

© Гибадуллин Р.Ф., Гашигуллин Д.А., Вершинин И.С., 2023

repository and streams with decorators. Streams with reference repositories and streams with decorators deal exclusively with bytes. While flexible and efficient, applications often operate at higher levels, such as text or XML. Adapters bridge this gap by putting a stream into a class shell with specialized methods that are typified for a specific format. The paper presents the StegoStream decorator developed by the authors, which is based on associative data protection mechanism. This decorator has the following advantages: it provides interaction with the adapter; it releases streams with reference repositories from the necessity of independent implementation of such features as hiding and unhiding; streams decorated with StegoStream do not suffer from interface changes; StegoStream may be used when chaining with other decorators (for example, the compression decorator may be combined with the hiding decorator). Practical use of StegoStream decorator is presented drawing on the example of the developed multi-client secure chat with a centralized server.

Keywords: associative steganography, cryptography, streaming architecture, decorator, information security.

For citation: Gibadullin R.F., Gashigullin D.A., Vershinin I.S. Development of StegoStream decorator for associative protection of byte stream. *Modeling, Optimization and Information Technology*. 2023;11(2). URL: https://moitvivt.ru/ru/journal/pdf?id=1359 DOI: 10.26102/2310-6018/2023.41.2.023 (In Russ.).

Введение

Стеганография играет важную роль в информационной безопасности, поскольку она обеспечивает скрытую передачу информации, предотвращая ее обнаружение и перехват. В основе стеганографии лежит идея маскировки данных внутри других данных или носителей, таких как изображения, аудиофайлы, видеофайлы, текстовые документы или бинарные файлы. Это позволяет передавать конфиденциальные данные незаметно для третьих лиц.

Роль стеганографии в информационной безопасности включает в себя следующие аспекты:

1. Защита конфиденциальности данных: стеганография позволяет скрыть секретные сообщения или информацию в обычных файлах, делая их невидимыми для непосвященных.
2. Обеспечение анонимности: стеганография может быть использована для обеспечения анонимности отправителей и получателей информации. Это особенно актуально в случаях, когда требуется защитить личность участников обмена информацией.
3. Устойчивость к стороннему вмешательству: поскольку стеганографические методы скрывают информацию внутри других данных, они устойчивы к перехвату и атакам. Это означает, что даже если злоумышленники перехватят зашифрованные данные, они не смогут получить доступ к скрытой информации без знания специальных стеганографических ключей и методов.
4. Защита интеллектуальной собственности: стеганография может использоваться для встраивания цифровых водяных знаков в цифровые произведения искусства или другие файлы, обеспечивая таким образом их защиту от неправомерного использования и распространения.
5. Координация скрытых каналов: стеганография может использоваться для создания скрытых каналов связи между различными участниками. Это позволяет координировать действия группы или организации, минимизируя риск обнаружения и перехвата информации.
6. Разведывательная деятельность: военные и разведывательные службы также могут использовать стеганографию для передачи секретных данных и сообщений, обеспечивая таким образом свою оперативную безопасность и устойчивость к перехвату.

В целом, стеганография является одним из инструментов в арсенале информационной безопасности, который может быть использован для защиты данных и коммуникации в различных сценариях. Однако стоит отметить, что стеганография обычно используется в сочетании с другими методами защиты, такими как криптография и сетевые протоколы, обеспечивающие безопасность и приватность данных.

Ассоциативная стеганография представляет собой уникальный синтез концепций стеганографии и криптографии, который ориентирован на защиту данных при анализе сцен. Анализ сцен относится к агрегированному описанию изображений с использованием терминологии «объекты-координаты» [1]. Поэтому большинство первых исследовательских работ, например [2, 3], направлено на управление защищенными картографическими базами данных, где каждый k-разрядный код (объекта или координаты) в итоге матричной бинаризации десятичных цифр (γ=10) с последующим маскированием и рандомизацией трансформируется в k-секционный стегоконтейнер. Для формирования данного стегоконтейнера сначала создается так называемый пустой контейнер длиной L=k(9n-12) по числу существенных бит бинарных эталонов десятичных цифр, где n – число столбцов бинарной матрицы-эталона [2]. Он заполняется отрезком псевдослучайной последовательности (ПСП) – гаммой (в основу взят генератор «вихрь Мерсенна» (Mersenne twister), разработанный в 1997 г. японскими учеными Макото Мацумото и Такудзи Нисимура [4]). Затем в него внедряются по позициям случайно сохраненные маскированием биты кода (Рисунок 1). Независимо от n среднее число таких бит 5k << L, что характерно для стеганографии. Поскольку любой байт можно представить трехразрядным десятичным числом (k=3), а цифровой контент – последовательностью байтов, то ассоциативная стеганография применима в различных прикладных областях [5-8].

С учетом современных вызовов информационной безопасности и постоянно увеличивающегося объема передаваемых и хранимых данных, актуальность внедрения стеганографического подхода защиты в существующие программные продукты возрастает. В отличие от традиционных методов стеганографического преобразования, ассоциативный подход обеспечивает практически абсолютную стеганографическую стойкость, а также более высокую помехозащищенность при хранении и передаче информации по незащищенным каналам связи в сравнении с известными криптографическими методами [8].

Таким образом, внедрение ассоциативного стеганографического подхода в программные продукты может значительно повысить их уровень безопасности, предоставляя надежную защиту конфиденциальности данных. Программные продукты, особенно те, которые оперируют конфиденциальной информацией, должны рассматривать возможность использования стеганографии для дополнительной защиты данных от несанкционированного доступа и вмешательства. Внедрение стеганографического подхода защиты может существенно укрепить общую защиту информации и повысить доверие пользователей к программным продуктам.

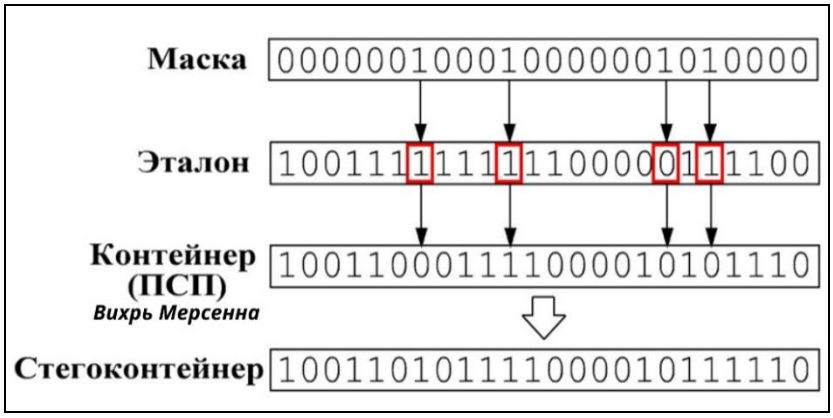

Рисунок 1 – Формирование стегоконтейнера
Figure 1 – Formation of a stegocontainer

В настоящее время платформа .NET Framework продолжает оставаться важным инструментом разработки для многих компаний и разработчиков. Вот несколько причин, почему .NET Framework остается актуальным:

1. Совместимость с существующими приложениями: многие корпоративные и настольные приложения, созданные на протяжении последних двух десятилетий, основаны на .NET Framework.

2. Богатая экосистема: платформа .NET Framework имеет обширную экосистему, включающую множество библиотек, инструментов, компонентов и разработчиков, которые обеспечивают поддержку и разработку новых приложений.

3. Поддержка множества языков программирования: .NET Framework поддерживает множество языков программирования, таких как C#, Visual Basic .NET и F#.

4. Интеграция с другими технологиями Microsoft: .NET Framework хорошо интегрирован с другими технологиями и продуктами Microsoft, такими как SQL Server, SharePoint, Office, и Azure, что облегчает разработку и поддержку корпоративных приложений.

Вышеотмеченное делает разработку декоратора StegoStream на базе .NET Framework актуальной задачей. И несмотря на то, что в последние годы Microsoft активно развивает и продвигает новую платформу .NET Core (позднее переименованную в .NET), предлагаемое авторами решение получено с применением библиотек, которые полностью совместимы с .NET Core, поэтому переход полученной сборки на флагманскую исполняющую среду не представляется затруднительным.

Разработка декоратора StegoStream

Представим потоковую архитектуру .NET, которая основана на трех концепциях: опорные хранилища, декораторы и адаптеры (Рисунок 2) [9].

Опорное хранилище представляет собой конечную точку, которая делает ввод и вывод полезными, скажем, файл или сетевое подключение. Точнее, это один или оба следующих компонента:

1) источник, с которого могут последовательно читаться байты;

2) приемник, куда байты могут последовательно записываться.

Тем не менее, опорное хранилище не может использоваться, если программисту не открыт доступ к нему. Стандартным классом .NET, который предназначен для такой цели, является Stream; он предоставляет стандартный набор методов, позволяющих выполнять чтение, запись и позиционирование. В отличие от массива, где все опорные

данные существуют в памяти одновременно, поток имеет дело с данными последовательно, либо по одному байту за раз, либо в блоках управляемого размера. Следовательно, поток может потреблять мало памяти независимо от размера его опорного хранилища.

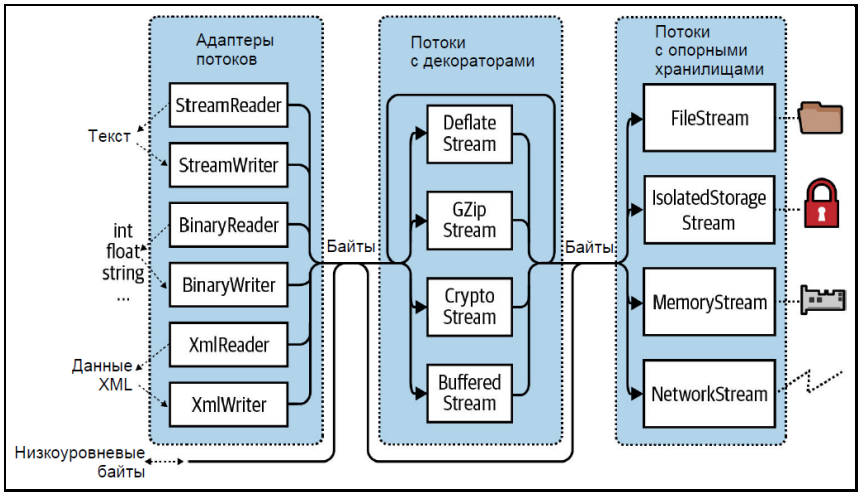

Рисунок 2 – Потоковая архитектура [9]
Figure 2 – Streaming architecture [9]

Потоки делятся на две категории:

1. Потоки с опорными хранилищами. Потоки, которые жестко привязаны к определенному типу опорного хранилища, такие как FileStream или NetworkStream.

2. Потоки с декораторами. Потоки, которые наполняют другие потоки, каким-то образом трансформируя данные, например, DeflateStream или CryptoStream.

Потоки с декораторами обладают перечисленными ниже архитектурными преимуществами:

1. Они освобождают потоки с опорными хранилищами от необходимости самостоятельной реализации таких возможностей, как сжатие и шифрование.

2. Потоки не страдают от изменения интерфейса, когда они декорированы.

3. Декораторы можно подключать во время выполнения.

4. Декораторы можно соединять в цепочки (скажем, декоратор сжатия можно соединить с декоратором шифрования).

Подключение декораторов во время выполнения означает, что имеется возможность динамически добавлять, изменять или удалять функциональность, предоставляемую декораторами, без изменения основного кода программы или базовых потоков. Это позволяет гибко определять, какие дополнительные функции должны применяться к потокам, исходя из текущих условий, пользовательских настроек или других факторов. Например, в зависимости от параметров конфигурации или прав доступа пользователя, можно добавить шифрование, сжатие или другую функциональность к потоку.

```
Stream stream = new FileStream("example.txt", FileMode.OpenOrCreate,
FileAccess.ReadWrite);
bool enableCompression = true; // Значение может быть задано динамически, например,
из конфигурации
bool enableEncryption = false; // Значение может быть задано динамически, например,
из прав доступа пользователя
if (enableCompression)
{
  stream = new GZipStream(stream, CompressionMode.Compress);
```

```
}
if (enableEncryption)
{
  // Применяем декоратор для шифрования (предполагая, что он реализован)
  stream = new EncryptionStream(stream);
}
using (StreamWriter writer = new StreamWriter(stream))
{
  writer.WriteLine("Данные для записи");
}
```

В этом примере динамически добавляются декораторы для сжатия и шифрования на основе условий enableCompression и enableEncryption. В зависимости от этих условий дополнительные функции будут добавлены к потоку во время выполнения программы. Это показывает гибкость и динамичность, которую предоставляют декораторы при работе с потоками.

Потоки с опорными хранилищами и потоки с декораторами имеют дело исключительно с байтами. Это гибко и эффективно, однако приложения часто работают на более высоких уровнях, таких как текст или XML. Адаптеры преодолевают такой разрыв, помещая поток в оболочку класса со специализированными методами, которые типизированы для конкретного формата. Например, средство чтения текста открывает доступ к методу ReadLine, а средство записи XML – к методу WriteAttributes.

Таким образом, потоки с опорными хранилищами предоставляют низкоуровневые данные; потоки с декораторами обеспечивают прозрачные двоичные трансформации вроде шифрования; адаптеры предлагают типизированные методы для работы с типами более высокого уровня, такими как строки и XML. Связи между ними проиллюстрированы на Рисунке 2. Чтобы сформировать цепочку, необходимо просто передать один объект конструктору другого класса.

Для разработки заявленного декоратора создан класс StegoStream, который наследуется от абстрактного класса Stream. Для простоты ограничимся представлением полей, конструктора и методов Read и Write класса StegoStream, более детально с реализацией декоратора можно ознакомиться в ветке StegoStream репозитория [10].

```
public class StegoStream : Stream
{
  private Stream _stream;
  private StegoAlg _stegoAlg;
  private char[,] _key;

  public StegoStream(Stream stream, StegoAlg stegoAlg)
  {
    _stream = stream;
    _stegoAlg = stegoAlg;
  }
  public override int Read(byte[] buffer, int offset, int count)
  {
     for (int i = offset; i < count; i++)
     {
       var hiddenCode = new byte[HiddenCodeLength/8];
       _innerStream.Read(hiddenCode, 0, hiddenCode.Length);
       _helper.TryDisclose(hiddenCode, _key, out buffer[i]);
     }
     return count;
  }
  public override void Write(byte[] buffer, int offset, int count)
  {
    for (int i = offset; i < offset + count; i++)
    {
      var hiddenCode = _helper.Hide(buffer[i], _key);
```

```
      _innerStream.Write(hiddenCode, 0, hiddenCode.Length);
    }
  }
  ...
}
```

Данная реализация обеспечивает возможность использования собственного декоратора потока StegoStream вместе с другими потоками, например:

```
string filePath = "test.txt";
string originalText = "Confidential information ...";

Stegomask s = new Stegomask(36);
s.GetEtalons(out char[,] etalons);
s.GetKey(out char[,] key);
var sa = new StegoAlg(36, etalons, key);

// Запись
using (FileStream fileStream = File.Create(filePath))
using (StegoStream ss = new StegoStream(fileStream, sa))
{
  byte[] data = Encoding.UTF8.GetBytes(originalText);
  ss.Write(data, 0, data.Length);
}

// Чтение
using (FileStream fileStream = File.OpenRead(filePath))
using (StegoStream ss = new StegoStream(fileStream, sa))
{
  byte[] buffer = new byte[originalText.Length];
  int bytesRead = ss.Read(buffer, 0, buffer.Length);
  string content = Encoding.UTF8.GetString(buffer, 0, bytesRead);
  Console.WriteLine("Recovered content: ");
  Console.WriteLine(content);
}
```

Сравнение StegoStream с CryptoStream

Сопоставим скорость работы разработанного декоратора StegoStream с декоратором CryptoStream, который позволяет обеспечить криптографические преобразования байтового потока на основе различных видов шифров.

Ниже представлен код программы, обеспечивающий шифрование и расшифрование 1 МБ данных с использованием блочного шифра AES.

```
SymmetricAlgorithm algorithm = Aes.Create();
byte[] iv = { 15, 122, 132, 5, 93, 198, 44, 31, 9, 39, 241, 49, 250, 188, 80, 7 };
string password = "Пароль для генерации ключа";
byte[] key;
using (SHA256 sha256 = SHA256.Create())
{
  byte[] passwordBytes = Encoding.UTF8.GetBytes(password);
  key = sha256.ComputeHash(passwordBytes);
}
// Генерация 1 МБ случайных данных
byte[] data = new byte[1024 * 1024];
new Random().NextBytes(data);
Stopwatch encryptionTimer = new Stopwatch();
byte[] encryptedData;
using (MemoryStream ms = new MemoryStream())
{
  using (ICryptoTransform encryptor = algorithm.CreateEncryptor(key, iv))
  using (Stream c = new CryptoStream(ms, encryptor, CryptoStreamMode.Write))
  {
    encryptionTimer.Start();
```

```
    c.Write(data, 0, data.Length);
  }
  Console.WriteLine($"Шифрование заняло: {encryptionTimer.ElapsedMilliseconds}
мс.");
  encryptedData = ms.ToArray();
}
Stopwatch decryptionTimer = new Stopwatch();
byte[] decryptedData = new byte[data.Length];
using (MemoryStream ms = new MemoryStream(encryptedData)) // Создаем новый
MemoryStream с зашифрованными данными
{
  using (ICryptoTransform decryptor = algorithm.CreateDecryptor(key, iv))
  using (Stream c = new CryptoStream(ms, decryptor, CryptoStreamMode.Read))
  {
    decryptionTimer.Start();
    c.Read(decryptedData, 0, decryptedData.Length);
  }
}
Console.WriteLine($"Расшифровка заняла: {decryptionTimer.ElapsedMilliseconds}
мс.");
```

Попытка оптимизировать программу вызовом конструктора MemoryStream единожды с последующей установкой ms.Position в нуль перед повторным использованием объекта ms в ходе расшифрования данных может привести к утечке ресурсов и непреднамеренному доступу к данным. В программе после использования CryptoStream для шифрования и вызова Dispose(), внутренний поток (MemoryStream) также закрывается, что делает его недоступным для последующего чтения или записи.

Следует отметить, что несмотря на то, что в программе размер входных данных кратен 128 битам (то есть 16 байтам), размер выходных данных после шифрования будет больше из-за использования дополнения. Дело в том, что в режиме сцепления блоков шифрования (CBC), который является режимом по умолчанию для класса SymmetricAlgorithm, использован алгоритм дополнения, добавляющий дополнительный блок в конец данных. Это делается для того, чтобы различать случай, когда данные имеют размер, кратный размеру блока, и когда данные кончаются дополнением. Рассмотрим, например, один из алгоритмов дополнения – PKCS #7. (Public Key Cryptography Standards № 7 – это стандарт, опубликованный RSA Laboratories, который описывает общую структуру для сообщений криптографии с использованием открытого ключа. Однако, в контексте дополнения данных при блочном шифровании, PKCS #7 часто упоминается для описания метода дополнения, который применяется при шифровании и расшифровании данных.) В этом алгоритме дополнение состоит из последовательности байт, каждый из которых равен количеству добавленных байт. Таким образом, если входные данные имеют размер, кратный размеру блока, алгоритм добавляет дополнительный блок, в котором каждый байт имеет значение, равное размеру блока (например, 16 для AES). В других случаях добавляется дополнение, равное количеству недостающих байт до кратности размеру блока, и каждый байт дополнения имеет значение, равное количеству добавленных байт. При расшифровании такое дополнение будет корректно удалено из расшифрованных данных для получения корректных исходных данных.

Чтобы задействовать шифрование / расшифрование байтового потока алгоритмом 3DES начальный фрагмент вышеуказанного кода следует заменить на:

```
SymmetricAlgorithm algorithm = TripleDES.Create();
byte[] iv = { 15, 122, 132, 5, 93, 198, 44, 31, 9, 39, 241, 49, 250, 188, 80, 7 };
string password = "Пароль для генерации ключа";
byte[] key;
using (SHA256 sha256 = SHA256.Create())
{
```

```
  byte[] passwordBytes = Encoding.UTF8.GetBytes(password);
  key = sha256.ComputeHash(passwordBytes);
}
// Отсечение ключа до 192 бит (24 байта), так как это максимальный размер ключа для
TripleDES
key = new ArraySegment<byte>(key, 0, 24).ToArray();
```

Размер блока в 3DES (Triple Data Encryption Standard) составляет 64 бита, или 8 байт. Это стандартный размер блока для алгоритма DES, и 3DES использует тот же размер блока, выполняя три последовательных операции шифрования DES с разными ключами. Как и в случае с AES режимом шифрования по умолчанию для TripleDES является CBC. В этом режиме исходный текст делится на блоки равного размера, и каждый блок сначала обрабатывается с использованием операции XOR с предыдущим блоком шифротекста, а затем шифруется. Первый блок операции XOR обрабатывается с использованием вектора инициализации (IV), который обеспечивает уникальность каждого шифрования.

Рассмотрим также и российский алгоритм шифрования ГОСТ 28147-89, с 1989 года являющийся стандартным алгоритмом шифрования в СССР, а затем и в Российской Федерации. В 2015 году он был заменен новым стандартом шифрования – «Кузнечик» (ГОСТ Р 34.12-2015). И хотя ГОСТ 28147-89 считается устаревшим стандартом, он до сих пор используется в некоторых российских криптографических системах, например, в системах электронной цифровой подписи (ЭЦП) на основе ГОСТ Р 34.10, в системах защиты информации в банковской сфере, в системах связи военных и государственных учреждений. Чтобы реализовать программу на языке программирования C# для шифрования / расшифрования данных на основе ГОСТ 28147-89 следует выполнить следующие действия (реализация ни одного из шифров ГОСТ в .NET Framework не включена):

1. Установить пакет GostCryptography [11]: Библиотека GostCryptography представляет собой набор реализаций криптографических алгоритмов, основанных на российских стандартах ГОСТ. Эта библиотека позволяет использовать различные алгоритмы шифрования, генерации ключей, создания цифровых подписей и хеширования с помощью .NET Framework и .NET Core.

2. Установить криптопровайдер ViPNet CSP 4 [12]: ViPNet CSP 4 – российский криптопровайдер, сертифицированный ФСБ России как средство криптографической защиты информации (СКЗИ) и электронной подписи. Установка данного криптопровайдера требуется для обеспечения полноценной работы библиотеки GostCryptography, так как он содержит необходимые компоненты для реализации стандартных алгоритмов ГОСТ и обеспечения совместимости с российскими криптографическими стандартами.

```
var algorithm = new Gost_28147_89_SymmetricAlgorithm();
algorithm.Mode = CipherMode.CBC;
algorithm.Padding = PaddingMode.PKCS7;
// Генерация 1 МБ случайных данных
byte[] data = new byte[1024 * 1024];
new Random().NextBytes(data);
var dataStream = new MemoryStream(data);
byte[] encryptedData;
Stopwatch timer = new Stopwatch();
using (MemoryStream encryptedDataStream = new MemoryStream())
{
  using (ICryptoTransform encryptor = algorithm .CreateEncryptor())
  using (Stream cryptoStream = new CryptoStream(encryptedDataStream, encryptor,
CryptoStreamMode.Write))
  {
    timer.Start();
```

```
    dataStream.CopyTo(cryptoStream);
    Console.WriteLine($"Шифрование заняло: {timer.ElapsedMilliseconds} мс.");
  }
  encryptedData = encryptedDataStream.ToArray();
}
timer.Reset();
using (MemoryStream encryptedDataStream = new MemoryStream(encryptedData))
{
  using (MemoryStream decryptedDataStream = new MemoryStream())
  using (ICryptoTransform decryptor = algorithm.CreateDecryptor())
  using (Stream cryptoStream = new CryptoStream(encryptedDataStream, decryptor,
CryptoStreamMode.Read))
  {
    timer.Start();
    cryptoStream.CopyTo(decryptedDataStream);
    Console.WriteLine($"Расшифровка заняла: {timer.ElapsedMilliseconds} мс.");
  }
}
```

Согласно сведениям, представленных в работе [13], шифр ГОСТ Р 34.12-2015 уступает в производительности по сравнению с шифром ГОСТ 28147-89 в 1,3 раза.

В завершение представим код программы, обеспечивающий сокрытие и раскрытие 1 МБ данных с использованием декоратора StegoStream.

```
Stegomask s = new Stegomask(36);
s.GetEtalons(out char[,] etalons); // Генерация эталонов.
s.GetKey(out char[,] key); // Генерация ключа.
var sa = new StegoAlg(36, etalons, key); // Создание объекта, отвечающего за
стеганографические преобразования: формирование контейнера, сокрытие и раскрытие
данных.
// Генерация 1 МБ случайных данных
byte[] data = new byte[1024 * 1024];
new Random().NextBytes(data);
Stopwatch timer = new Stopwatch();
byte[] stegoData;
// Запись
using (MemoryStream ms = new MemoryStream())
{
  var ss = new StegoStream(ms, sa);
  timer.Start();
  ss.Write(data, 0, data.Length);
  Console.WriteLine($"Сокрытие заняло: {timer.ElapsedMilliseconds} мс.");
  stegoData = ms.ToArray();
}
timer.Reset();
// Чтение
using (MemoryStream ms = new MemoryStream(stegoData))
{
  var ss = new StegoStream(ms, sa);
  timer.Start();
  ss.Read(data, 0, data.Length);
  Console.WriteLine($"Раскрытие заняло: {timer.ElapsedMilliseconds} мс.");
}
```

В программе для ассоциативного механизма защиты заданы следующие параметры:

– n = 36;

– генератор ПСП – Mersenne twister (инкапсулирован в объекте типа StegoAlg).

Выбор столь малого размера n может вызвать вопросы в отношении качеств помехоустойчивости, стойкости к атаке перебором ключей и стегостойкости, однако по результатам анализа возможностей снижения стегоразмеров в статье [8] установлено, что с уменьшением n необходимый для удовлетворения критерия полноты объем

перебора нарастает, но стойкость метода к «лобовой» криптоатаке при выборе n = 40 и 30 сохраняется. Такой выбор позволяет снизить объем стегосообщений на 33-50 % по сравнению со случаем n = 60 при сохранении высоких уровней стегостойкости и помехоустойчивости.

Так как любому байту можно сопоставить трехразрядный десятичный код, который при n = 36 с позиций ассоциативной стеганографии трансформируется в 117 байт, то 1 МБ информации существенно возрастет (такова «плата» за высокий уровень стегостойкости и помехоустойчивости). Поэтому перед сокрытием данных рекомендуется применить алгоритм сжатия.

В пространстве имен System.IO.Compression представлены два ключевых потока сжатия: DeflateStream и GZipStream. Эти потоки используют широко известный алгоритм сжатия, аналогичный тому, что применяется при формировании архивов в формате ZIP. Основное различие между этими классами заключается в том, что GZipStream добавляет дополнительные метаданные в начале и конце сжатых данных, включая код контрольной суммы CRC, который позволяет обнаруживать ошибки.

Кроме того, GZipStream соответствует стандартам, что обеспечивает совместимость с другим программным обеспечением. В составе .NET Framework также представлен BrotliStream, реализующий алгоритм сжатия Brotli. В сравнении с DeflateStream и GZipStream, BrotliStream работает медленнее на этапе сжатия данных, его производительность снижается в 10 раз и более. Однако этот класс обеспечивает более высокий коэффициент сжатия. Стоит отметить, что замедление производительности наблюдается только при сжатии, в то время как процесс распаковки выполняется эффективно.

В целом, DeflateStream, GZipStream и BrotliStream предлагают различные варианты сжатия данных, каждый из которых имеет свои преимущества и недостатки. Выбор определенного класса зависит от специфических требований проекта, таких как требуемая производительность, степень сжатия и совместимость с другими инструментами и программным обеспечением.

Степень сжатия текста при использовании вышеуказанных алгоритмов сжатия сильно зависит от размера, содержимого и структуры исходного текста. Поэтому сложно точно указать степень сжатия в числовом соотношении. Однако известны некоторые общие оценки для этих алгоритмов:

1. DeflateStream: Deflate – это алгоритм сжатия без потерь, который может сжимать данные примерно на 50-70 % от исходного размера. Результаты сжатия могут варьироваться в зависимости от типа данных и характеристик исходного текста.

2. GZipStream: GZip использует тот же алгоритм сжатия, что и Deflate, но добавляет некоторые метаданные и контрольную сумму для обеспечения целостности данных. Степень сжатия GZipStream схожа с DeflateStream, и в среднем составляет около 50-70 % от исходного размера.

3. BrotliStream: Brotli – это новый алгоритм сжатия без потерь, разработанный Google, который обеспечивает лучшую степень сжатия по сравнению с Deflate и GZip. В среднем Brotli может сжимать данные на 60-80 % от исходного размера. Однако фактическая степень сжатия зависит от исходных данных.

Классы DeflateStream, GZipStream и BrotliStream представляют собой декораторы потоков. Они выполняют сжатие или распаковку данных потока, указанного при создании экземпляров этих классов. В качестве декораторов они добавляют функциональность сжатия и распаковки к базовому потоку, не изменяя его структуры. Применение декораторов в данном контексте позволяет добиться гибкости и модульности, так как сжатие или распаковка может быть применена к любому потоку без изменения его основной логики. Благодаря этому программист может легко

настроить обработку данных в зависимости от требуемых параметров сжатия и распаковки, выбирая один из перечисленных выше классов.

В Таблице 1 представлены средние времена выполнения процедур шифрования (расшифрования), сокрытия / раскрытия 1 МБ. Выполнение процедур проводилось на процессоре Intel Core i5 9300H с базовой частотой 2,4 ГГц под управлением ОС Windows 11 Pro.

Таблица 1 – Времена выполнения процедур шифрования (расшифрования), сокрытия / раскрытия в миллисекундах
Table 1 – Execution time of encryption (decryption) procedures, concealment / disclosure in milliseconds

AES	3DES	ГОСТ 28147-89	ГОСТ Р 34.12-2015	Stego (n=36)
2	33	20	26	950/16

Скорость работы шифра AES высока по сравнению с другими алгоритмами шифрования, потому что современные процессоры, включая Intel Core i5 9300H, имеют аппаратную поддержку AES. Это означает, что процессоры Intel содержат инструкции, оптимизированные для быстрого выполнения операций шифрования и дешифрования AES. Набор инструкций, используемых для этой цели, известен как AES-NI (Advanced Encryption Standard New Instructions). В целом скорость работы шифров существенно превышает скорость сокрытия ассоциативного механизма защиты, но уступает скорости раскрытия (кроме AES), что важно для консервативных баз данных, где обновление эпизодично.

Разработка защищенного чата

В качестве основы разработки мультиклиентного чата с централизованным сервером взят проект, предложенный web-разработчиком Andrew Pociu. Приложение обеспечивает обмен сообщениями по компьютерной сети в режиме реального времени. В основе приложения – два модуля: клиент и сервер. Клиенты могут указать IP-адрес сервера и порт, через который будут осуществлять обмен сообщениями после подключения к серверу, задавать свое имя для размещения его в титульной части отправленных сообщений и секретный ключ. Все подключенные к серверу клиенты могут одновременно отправлять сообщения на сервер и видеть сообщения друг друга, посредством широковещательной рассылки, которую имитирует сервер. Серверный модуль содержит информацию обо всех подключенных клиентах, ждет сообщения от каждого и отправляет входящие сообщения для всех, имитируя широковещательную рассылку. На Рисунке 3 представлен графический интерфейс клиентского и серверного модулей разработанного чата.

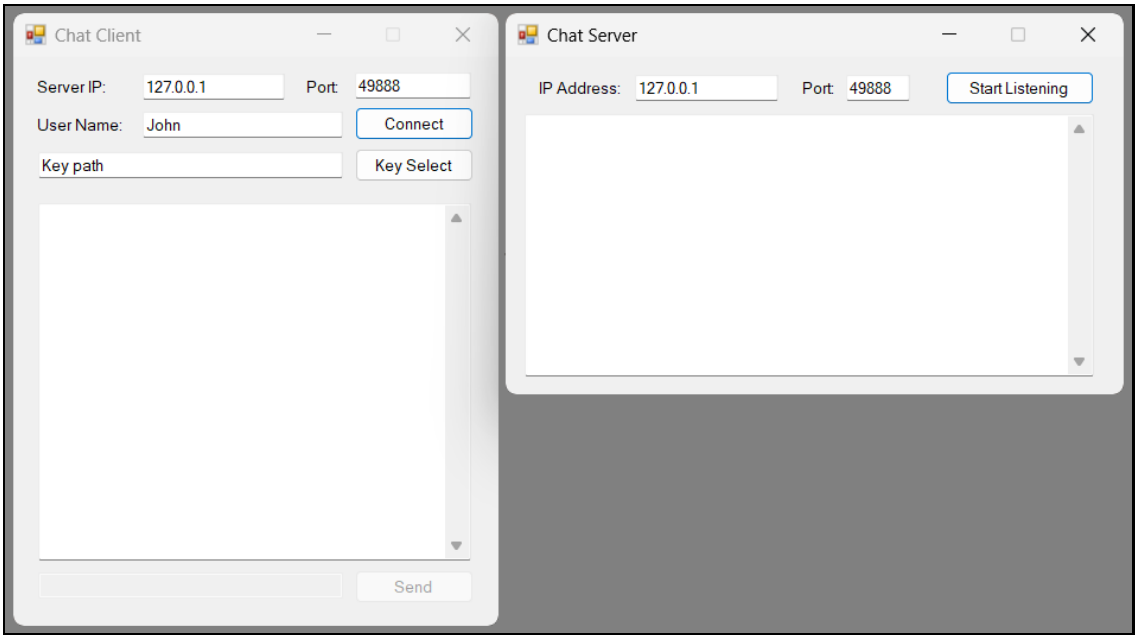

Рисунок 3 – Графический интерфейс мультиклиентного чата
Figure 3 – Multiclient chat graphical interface

Детали разработки мультиклиентного чата подробно представлены в репозитории авторов [14], поэтому ограничимся лишь описанием сути защищенного взаимодействия пользователей на примере взаимодействия клиента и сервера.

Фрагмент программного кода на стороне сервера:

```
TcpListener server = new TcpListener(IPAddress.Any, 49888);
server.Start();
Console.WriteLine("Ожидание подключения клиента...");
TcpClient client = server.AcceptTcpClient();
Console.WriteLine("Клиент подключен.");
using (NetworkStream networkStream = client.GetStream())
using (StegoStream stegoStream = new StegoStream(networkStream, sa)
using (StreamWriter writer = new StreamWriter(stegoStream))
{
  string message = "Сокрытое сообщение от сервера";
  writer.WriteLine(message);
}
```

В данном коде объект TcpListener используется для прослушивания подключений клиентов через TCP протокол на сервере. TcpListener является классом из пространства имен System.Net.Sockets и предоставляет простой способ прослушивания подключений клиентов и установления соединений для обмена данными. При использовании TcpListener сервер выполняет следующие действия:

1. Создание объекта TcpListener, указывая IP-адрес и порт, на котором сервер будет прослушивать подключения клиентов. В данном случае используется IPAddress.Any, что означает прослушивание на всех доступных сетевых интерфейсах, и порт 49888.

2. Запуск прослушивания подключений клиентов с помощью метода Start().

3. Ожидание подключения клиента с использованием метода AcceptTcpClient(), который блокирует выполнение программы, пока клиент не подключится. Когда клиент подключается, метод возвращает объект TcpClient, который представляет соединение с клиентом.

4. После успешного подключения клиента сервер использует объект TcpClient для взаимодействия с клиентом и обмена данными, в данном случае для передачи сокрытого сообщения.

Следует также отметить, что в действительности сокрытие данных на стороне сервера не происходит, так как он играет роль арбитра между двумя и более клиентами, имитируя широковещательную рассылку.

Фрагмент программного кода на стороне клиента:

```
TcpClient client = new TcpClient("127.0.0.1", 49888);
using (NetworkStream networkStream = client.GetStream())
using (StegoStream stegoStream = new StegoStream(networkStream, sa))
using (StreamReader reader = new StreamReader(stegoStream))
{
  string message = reader.ReadLine();
  Console.WriteLine("Получено от сервера: " + message);
}
```

В клиентском коде объект TcpClient используется для установления TCP-соединения с сервером и обмена данными с ним. TcpClient – это класс из пространства имен System.Net.Sockets, который предоставляет простой и удобный способ работы с TCP-подключениями. При использовании TcpClient в клиентском коде клиент выполняет следующие действия:

1. Создание объекта TcpClient.

2. Подключение к серверу, указывая его IP-адрес и порт с помощью метода Connect(). В данном случае используется IP-адрес 127.0.0.1 (т. е., локальный адрес сервера) и порт 49888.

3. После успешного подключения к серверу клиент использует объект TcpClient для взаимодействия с сервером и обмена данными, в данном случае для получения зашифрованного сообщения от сервера.

4. Получение NetworkStream из объекта TcpClient с помощью метода GetStream() и использование этого потока для обмена данными с сервером. В контексте использования StegoStream, NetworkStream используется в качестве базового потока для работы с StegoStream.

Заключение

Ассоциативный подход защиты данных представляет собой перспективное решение для широкого круга приложений, включая:

1. Видеонаблюдение: в системах видеонаблюдения, где большой объем данных передается и хранится в зашифрованном виде, быстрая расшифровка является важным фактором. В то же время обновление данных происходит реже, поэтому скорость сокрытия данных является менее важной.

2. Медицинские системы: в медицинских учреждениях, где обрабатываются и передаются конфиденциальные данные пациентов, быстрое раскрытие информации может быть критичным для обеспечения своевременного и эффективного лечения. Однако, внесение изменений в данные происходит относительно редко.

3. Системы контроля доступа: в организациях, где требуется быстрый доступ к защищенным помещениям или ресурсам, механизмы защиты данных должны обеспечивать быстрое раскрытие, чтобы удовлетворить потребности пользователей, но могут иметь низкую скорость сокрытия, поскольку информация обновляется редко.

4. Консервативные базы данных: в ситуациях, где данные обновляются редко, но требуется быстрый доступ для чтения, такие как архивы или статические ресурсы,

механизмы защиты данных с высокой скоростью раскрытия и низкой скоростью сокрытия могут быть оптимальными.

5. Военные и космические системы: в военных и космических системах, где связь может быть подвержена сбоям и помехам, высокая помехоустойчивость является критическим фактором. В то же время быстрый доступ к данным может быть необходим для принятия оперативных решений, а низкая скорость сокрытия данных может быть приемлемой.

Результаты работы показывают, что предложенный ассоциативный подход может быть успешно интегрирован в программные компоненты организаций, сталкивающихся с вышеописанными сценариями использования. Полученные в ходе исследования результаты способствуют улучшению надежности и безопасности обработки и передачи данных в различных сферах деятельности.

Предметом дальнейших исследований является применение ассоциативного стеганографического механизма защиты данных в области защиты интеллектуальной собственности. Ассоциативный подход предоставляет ряд перспективных возможностей:

1. Защита авторских прав: ассоциативная стеганография может использоваться для внедрения уникальных цифровых водяных знаков в произведения интеллектуальной собственности, такие как изображения, аудио и видеофайлы. Это позволяет идентифицировать оригинальные произведения и отслеживать их распространение.

2. Мониторинг нарушений: ассоциативный механизм стеганографии может облегчить отслеживание и мониторинг нелегального использования или распространения материалов, защищенных авторским правом, путем внедрения скрытых метаданных или идентификационной информации.

3. Защита конфиденциальных документов: владельцы интеллектуальной собственности могут использовать ассоциативный стеганографический подход для скрытия и защиты конфиденциальной информации, такой как патенты, техническая документация и коммерческая тайна, обеспечивая их сохранность от несанкционированного доступа.

4. Обеспечение анонимности: в случаях, когда авторы произведений интеллектуальной собственности хотят сохранить свою анонимность или избежать преследования, ассоциативная стеганография может быть использована для сокрытия авторства и обеспечения анонимного распространения произведений.

В целом, использование ассоциативного стеганографического механизма защиты данных в защите интеллектуальной собственности открывает новые горизонты для обеспечения конфиденциальности, сохранности и контроля над распространением произведений интеллектуальной собственности.

СПИСОК ИСТОЧНИКОВ

1. Duda R.O., Hart P.E., Stork D.G. Pattern classification and scene analysis. *New York: Wiley*. 1973;3:731–739.
2. Raikhlin V.A., Vershinin I.S., Gibadullin R.F., Pystogov S.V. Reliable recognition of masked binary matrices. Connection to information security in map systems. *Lobachevskii Journal of Mathematics,* 2013;34(4):319–325. DOI: 10.1134/S1995080213040112.
3. Raikhlin V.A., Gibadullin R.F., Vershinin I.S., Pystogov S.V. Reliable recognition of masked cartographic scenes during transmission over the network. *In 2016 International Siberian Conference on Control and Communications (SIBCON), IEEE*. 2016;1–5. DOI: 10.1109/SIBCON.2016.7491657.

4. Tian X., Benkrid K. Mersenne twister random number generation on FPGA, CPU and GPU. *2009 NASA/ESA Conference on Adaptive Hardware and Systems, San Francisco, CA, USA*. 2009;460–464. DOI: 10.1109/AHS.2009.11.
5. Raikhlin V.A., Vershinin I.S., Gibadullin R.F. The elements of associative steganography theory. *Moscow University Computational Mathematics and Cybernetics*. 2019;43(1):40–46. DOI: 10.3103/S0278641919010072.
6. Vershinin I.S., Gibadullin R.F., Pystogov S.V., Raikhlin V.A. Associative steganography. Durability of associative protection of information. *Lobachevskii Journal of Mathematics*. 2020;41(3):440–450. DOI: 10.1134/S1995080220030191.
7. Vershinin I.S., Gibadullin R.F., Pystogov S.V., Raikhlin V.A. Associative steganography of text messages. *Moscow University Computational Mathematics and Cybernetics*. 2021;45(1):1–11. DOI: 10.3103/S0278641921010076.
8. Raikhlin V.A., Gibadullin R.F., Vershinin I.S. Is it possible to reduce the sizes of stegomessages in associative steganography? *Lobachevskii Journal of Mathematics*. 2022;43(2):455–462.
9. Albahari J. *C# 10 in a Nutshell*. O'Reilly Media, Inc.; 2022. 1061 p.
10. Stego. Доступно по: https://bitbucket.org/landwatersun/stego/ (дата обращения: 15.04.2023).
11. GostCryptography. Доступно по: https://github.com/AlexMAS/GostCryptography/ (дата обращения: 08.04.2023).
12. ИнфоТеКС. Доступно по: https://infotecs.ru/ (дата обращения: 15.04.2023).
13. Фомичев В.М., Бобровский Д.А., Коренева А.М. Экспериментальная оценка производительности одного класса криптоалгоритмов на основе обобщения сетей Фейстеля. *Прикладная дискретная математика. Приложение.* 2020;13:59-62. DOI: 10.17223/2226308X/13/18.
14. Multi-Client Chat Server. Доступно по: https://bitbucket.org/landwatersun/multi-client-chat-server/ (дата обращения: 15.04.2023).

REFERENCES

1. Duda R.O., Hart P.E., Stork D.G. Pattern classification and scene analysis. *New York: Wiley*. 1973;3:731–739.
2. Raikhlin V.A., Vershinin I.S., Gibadullin R.F., Pystogov S.V. Reliable recognition of masked binary matrices. Connection to information security in map systems. *Lobachevskii Journal of Mathematics*, 2013;34(4):319–325. DOI: 10.1134/S1995080213040112.
3. Raikhlin V.A., Gibadullin R.F., Vershinin I.S., Pystogov S.V. Reliable recognition of masked cartographic scenes during transmission over the network. *In 2016 International Siberian Conference on Control and Communications (SIBCON), IEEE*. 2016;1–5. DOI: 10.1109/SIBCON.2016.7491657.
4. Tian X., Benkrid K. Mersenne Twister Random Number Generation on FPGA, CPU and GPU. *2009 NASA/ESA Conference on Adaptive Hardware and Systems, San Francisco, CA, USA*. 2009;460–464. DOI: 10.1109/AHS.2009.11.
5. Raikhlin V.A., Vershinin I.S., Gibadullin R.F. The Elements of Associative Steganography Theory. *Moscow University Computational Mathematics and Cybernetics*. 2019;43(1):40–46. DOI: 10.3103/S0278641919010072.
6. Vershinin I.S., Gibadullin R.F., Pystogov S.V., Raikhlin V.A. Associative steganography. Durability of associative protection of information. *Lobachevskii Journal of Mathematics*. 2020;41(3):440–450. DOI: 10.1134/S1995080220030191.

7. Vershinin I.S., Gibadullin R.F., Pystogov S.V., Raikhlin V.A. Associative steganography of text messages. *Moscow University Computational Mathematics and Cybernetics*. 2021;45(1):1–11. DOI: 10.3103/S0278641921010076.
8. Raikhlin V.A., Gibadullin R.F., Vershinin I.S. Is it possible to reduce the sizes of stegomessages in associative steganography? *Lobachevskii Journal of Mathematics*. 2022;43(2):455–462.
9. Albahari J. *C# 10 in a Nutshell*. O'Reilly Media, Inc.; 2022. 1061 p.
10. Stego. URL: https://bitbucket.org/landwatersun/stego/ (accessed on 15.04.2023).
11. GostCryptography. URL: https://github.com/AlexMAS/GostCryptography/ (accessed on 08.04.2023).
12. InfoTeKS. URL: https://infotecs.ru/ (accessed on 15.04.2023).
13. Fomichev V.M., Bobrovsky D.A., Koreneva A.M. Experimental evaluation of performance of one class of cryptoalgorithms based on generalization of Feistel networks. *Pricladnaia diskretnaia matematika. Prilozhenie = Applied discrete mathematics. Appendix.* 2020;13:59–62. DOI: 10.17223/2226308X/13/18. (In Russ.)
14. Multi-Client Chat Server. URL: https://bitbucket.org/landwatersun/multi-client-chat-server/ (accessed on 15.04.2023).

ИНФОРМАЦИЯ ОБ АВТОРАХ / INFORMATION ABOUT THE AUTHORS

Гибадуллин Руслан Фаршатович, кандидат технических наук, доцент, доцент кафедры компьютерных систем, Казанский национальный исследовательский технический университет им. А.Н. Туполева–КАИ, Казань, Российская Федерация.
e-mail: landwatersun@mail.ru
ORCID: 0000-0001-9359-911X

Ruslan Farshatovich Gibadullin, Candidate of Technical Sciences, Associate Professor, Associate Professor at Computer Systems Department, Kazan National Research Technical University named after A.N. Tupolev–KAI, Kazan, the Russian Federation.

Гашигуллин Данил Айратович, студент кафедры компьютерных систем, Казанский национальный исследовательский технический университет им. А.Н. Туполева–КАИ, Казань, Российская Федерация.
e-mail: gashigullin44@gmail.com

Danil Ayratovich Gashigullin, Undergraduate Student, Computer Systems Department, Kazan National Research Technical University named after A.N. Tupolev–KAI, Kazan, the Russian Federation.

Вершинин Игорь Сергеевич, кандидат технических наук, доцент, заведующий кафедрой компьютерных систем, Казанский национальный исследовательский технический университет им. А.Н. Туполева–КАИ, Казань, Российская Федерация.
e-mail: vershinin_igor@rambler.ru

Igor Sergeevich Vershinin, Candidate of Technical Sciences, Associate Professor, Head of Computer Systems Department, Kazan National Research Technical University named after A.N. Tupolev–KAI, Kazan, the Russian Federation.

Статья поступила в редакцию 07.05.2023; одобрена после рецензирования 23.05.2023; принята к публикации 19.06.2023.

The article was submitted 07.05.2023; approved after reviewing 23.05.2023; accepted for publication 19.06.2023.