DOI: 10.58168/MoInSyTe2024_170-180
УДК 004.9

ПОСТРОЕНИЕ ЗАЩИЩЕННОГО ХРАНИЛИЩА НА C#

А.В. Полуэктов[1], В.И. Силонов[1], И.В. Журавлева[1], А.С. Кравченко[1]

[1]ФГБОУ ВО «Воронежский государственный лесотехнический университет имени Г.Ф. Морозова»

Аннотация. В статье рассматривается вопрос построения защищенного хранилища данных на языке программирования C#. Изложены основные методы защиты данных, включая криптографию, аутентификацию и авторизацию, а также физическую защиту сервера. Приведены рекомендации по проектированию защищенного хранилища, включая архитектурные решения и управление доступом. Рассмотрена реализация на C# с использованием криптографических библиотек, хэширования паролей и шифрования секретной информации. Представлены примеры тестовых сценариев и оценка безопасности.

Ключевые слова: Защищенное хранилище, C#, криптография, аутентификация, авторизация, физическая защита, проектирование, доступ, шифрование, тестирование, безопасность.

BUILDING SECURED STORAGE IN C#

A.V. Poluektov[1], V.I. Silonov[1], I.V. Zhuravleva[1], A.S. Kravchenko[1]

[1]Voronezh State University of Forestry and Technologies named after G.F. Morozov

Abstract. The article discusses the issue of building a secure data storage in the C# programming language. The basic methods of data protection are outlined, including cryptography, authentication and authorization, as well as physical server protection. Provides guidelines for secure storage design, including architectural design and access control. An implementation in C# using cryptographic libraries, password hashing and encryption of secret information is considered. Sample test scenarios and security assessments are presented.

Keywords: Secure storage, C#, cryptography, authentication, authorization, physical security, design, access, encryption, testing, security.

Проблема сохранности данных возникает из-за угроз, которым они подвергаются в цифровом окружении. Их условно можно определить, как потенциаль-

ные угрозы, такие как кибератаки, хакерские атаки, вирусы, программное обеспечение-вредоносное ПО так и физические повреждения, которые могут привести к потере, утечке или повреждению данных. Защищенное хранилище данных позволяет предотвратить и решить проблемы сохранности данных и представляет собой специальное место или инфраструктуру, где данные могут быть сохранены и защищены от несанкционированного доступа и повреждений. Защищенные хранилища обеспечивают различные меры безопасности, такие как шифрование данных, контроль доступа, механизмы аутентификации и мониторинг, чтобы предотвратить несанкционированный доступ или использование данных.

Защищенное хранилище данных имеет следующие характеристики [2], [5]:

- конфиденциальность - обеспечивают конфиденциальность данных, предотвращая несанкционированный доступ или утечку информации;
- целостность - гарантируют целостность данных, предотвращая их повреждение или изменение несанкционированными лицами;
- доступность - обеспечивают доступ к данным только авторизованным пользователям, предотвращая потерю или недоступность информации;
- восстановление данных - в случае потери или повреждения данных защищенное хранилище позволяет восстановить информацию из резервных копий или архивов;
- соответствие законодательству - помогают организациям соблюдать требования законодательного регулирования и юридические нормы в отношении обработки и хранения данных.

Все указанные аспекты делают защищенное хранилище данных необходимым для обеспечения сохранности информации и предотвращения потенциальных угроз ее безопасности.

Для реализации программных методов защиты используются способы защиты [3], [4], [6].

- криптографические алгоритмы - используются для шифрования данных и защиты их от несанкционированного доступа, преобразуют данные в нечитаемую форму, которую могут расшифровать только те, у кого есть соответствующий ключ доступа;
- аутентификация и авторизация - это процессы проверки легитимности пользователей и предоставления им соответствующих прав доступа;
- физическая защита сервера - направлена на защиту аппаратного

обеспечения и физического местоположения сервера. она ограничивает несанкционированный доступ к серверам и предотвращает потенциальные физические угрозы.

При разработке специального программного обеспечения с использованием языка программироания C# можно использовать следующие программные методы.

Пример 1. Защита данных с использованием криптографических библиотек.

Для этого у языка программирования C# есть специальные криптографические библиотеки и используются классы из пространства имен System.Security.Cryptography. Ниже приведен пример шифрования и расшифрования строки с использованием алгоритма AES:

```
using System;
using System.IO;
using System.Security.Cryptography;

public static class CryptoExample
{
    private static readonly byte[] Key = new byte[32] { 0x2A, 0x5C, 0x82, 0x47, 0xC9, 0xE3, 0x9F, 0x81, 0xCD, 0xAE, 0x53, 0x7F, 0x3C, 0x24, 0x92, 0x16, 0xE6, 0x01, 0x4B, 0x7D, 0xF2, 0xA5, 0xD6, 0x98, 0xC1, 0x8B, 0x55, 0x63, 0x74, 0x29, 0xBB, 0x64 };
    private static readonly byte[] IV = new byte[16] { 0x42, 0xA7, 0x3E, 0xAF, 0x08, 0xE4, 0xD5, 0xF1, 0xC7, 0x72, 0x9B, 0x6D, 0x58, 0x93, 0x02, 0x34 };

    public static byte[] EncryptString(string plainText)
    {
        using (Aes aesAlg = Aes.Create())
        {
            aesAlg.Key = Key;
            aesAlg.IV = IV;

            ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);
```

```
            using (MemoryStream msEncrypt = new MemoryStream())
            {
                using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
                {
                    using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
                    {
                        swEncrypt.Write(plainText);
                    }
                    return msEncrypt.ToArray();
                }
            }
        }

        public static string DecryptString(byte[] cipherText)
        {
            using (Aes aesAlg = Aes.Create())
            {
                aesAlg.Key = Key;
                aesAlg.IV = IV;

                ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);

                using (MemoryStream msDecrypt = new MemoryStream(cipherText))
                {
                    using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read))
                    {
                        using (StreamReader srDecrypt = new StreamReader(csDecrypt))
                        {
                            return srDecrypt.ReadToEnd();
                        }
                    }
                }
```

```
        }
    }

    public static void Main()
    {
        string plainText = "Hello, world!";
        byte[] cipherText = EncryptString(plainText);
        string decryptedText = DecryptString(cipherText);

        Console.WriteLine("Original: {0}", plainText);
        Console.WriteLine("Encrypted: {0}", Convert.ToBase64String(cipher-
Text));
        Console.WriteLine("Decrypted: {0}", decryptedText);
    }
}
```

Выполнение кода выше приведет к зашифрованию строки "Hello, world!" с использованием алгоритма AES и выводу зашифрованного текста, а затем к расшифровке и выводу исходного текста.

Пример 2. Хэширование паролей и секретных данных.

Хэширование паролей и секретных данных - это распространенная практика для защиты данных от несанкционированного доступа. Ниже приведен пример хэширования пароля с использованием алгоритма SHA256:

```
using System;
using System.Security.Cryptography;

public static class HashingExample
{
    public static string HashPassword(string password)
    {
        using (SHA256 sha256Hash = SHA256.Create())
        {
            byte[] bytes = sha256Hash.ComputeHash(System.Text.Encod-
ing.UTF8.GetBytes(password));
```

```
            return Convert.ToBase64String(bytes);
        }
    }

    public static bool VerifyPassword(string password, string hashedPassword)
    {
        string hashedInput = HashPassword(password);

        return (hashedInput == hashedPassword);
    }

    public static void Main()
    {
        string password = "myPassword123";
        string hashedPassword = HashPassword(password);

        Console.WriteLine("Original password: {0}", password);
        Console.WriteLine("Hashed password: {0}", hashedPassword);

        bool isPasswordValid = VerifyPassword(password, hashedPassword);
        Console.WriteLine("Password is valid: {0}", isPasswordValid);
    }
}
```

Выполнение кода выше приводит к хэшированию пароля "myPassword123" с использованием алгоритма SHA256 и выводу захэшированного значения. Затем происходит проверка валидности пароля, используя функцию VerifyPassword.

Пример 3. Работа с шифрованными контейнерами данных.

Шифрованные контейнеры данных позволяют защитить целые наборы данных, сохраняя их в зашифрованном виде и обеспечивая доступ только авторизованным пользователям. В C# можно использовать класс ProtectedData из пространства имен System.Security.Cryptography для работы с шифрованными контейнерами данных. Ниже приведен пример сохранения и чтения шифрованных данных:

```
using System;
using System.IO;
using System.Security.Cryptography;

public static class ProtectedDataExample
{
    public static byte[] ProtectData(byte[] data)
    {
        return ProtectedData.Protect(data, null, DataProtectionScope.CurrentUser);
    }

    public static byte[] UnprotectData(byte[] protectedData)
    {
        return ProtectedData.Unprotect(protectedData, null, DataProtectionScope.CurrentUser);
    }

    public static void Main()
    {
        string originalData = "Sensitive information";
        byte[] dataBytes = System.Text.Encoding.UTF8.GetBytes(originalData);

        byte[] protectedData = ProtectData(dataBytes);
        Console.WriteLine("Protected data: {0}", Convert.ToBase64String(protectedData));

        byte[] decryptedData = UnprotectData(protectedData);
        string decryptedText = System.Text.Encoding.UTF8.GetString(decryptedData);
        Console.WriteLine("Decrypted data: {0}", decryptedText);
    }
}
```

Выполнение приведенного кода приведет к защите данных "Sensitive information", используя функцию ProtectData, и выводу зашифрованных данных.

Затем эти данные расшифровываются с помощью функции UnprotectData, и результат выводится на экран.

Обратим внимание, что в каждом из этих примеров показан основной принцип работы соответствующих функций и методов. В реальных системах защиты данных обычно требуется принять дополнительные меры, такие как сохранение ключей безопасности в безопасном хранилище, управление доступом и т.д.

Для оценки защищенности данных, реализованных на языке программирования C# с использованием криптографических библиотек, хэширования паролей и секретных данных, а также работы с шифрованными контейнерами данных, можно провести следующие тестовые сценарии [1], [7], [8]:

1. Тестирование хэширования паролей:
   - создание пользователя с паролем;
   - хэширование пароля и сохранение его в базе данных;
   - аутентификация пользователя с введенным паролем;
   - сравнение хэша введенного пароля с сохраненным хэшем.
2. Тестирование шифрования и дешифрования данных:
   - шифрование некоторых секретных данных с использованием симметричного ключа;
   - сохранение зашифрованных данных в файл или базу данных;
   - дешифрование данных с помощью правильного ключа;
   - проверка правильности дешифрованных данных.
3. Тестирование работоспособности криптографических библиотек:
   - генерация случайного ключа шифрования;
   - шифрование и расшифрование каких-либо данных с использованием библиотеки;
   - проверка правильности расшифрованных данных.
4. Тестирование обработки ошибок и исключений:
   - попытка использования неправильного пароля для расшифровки данных;
   - попытка дешифрования данных без доступа к необходимым ключам;
   - проверка обработки исключений и корректности сообщений об ошибках.

5. Тестирование работы с шифрованными контейнерами данных:

- создание шифрованного контейнера данных и добавление в него некоторых элементов;
- чтение данных из контейнера и проверка корректности расшифрованных данных;
- редактирование и удаление элементов из контейнера;
- проверка сохранения изменений в контейнере и корректности работы со шифрованными данными.

Анализ результатов тестирования включает в себя оценку успешности выполнения каждого тестового сценария, выявление потенциальных уязвимостей защищенности данных и производительности криптоопераций. Также важно учесть возможные ошибки в реализации и принять меры для их устранения.

Построение защищенного хранилища на языке программирования C# является важной задачей для обеспечения безопасности данных. Реализация описанных принципов и методов поможет предотвратить утечку и несанкционированный доступ к конфиденциальной информации. При разработке такого хранилища необходимо учитывать современные требования безопасности и следовать рекомендациям по обеспечению защиты данных.

Список литературы

1. Маракуева, Н. В. Защищенное файловое хранилище / Н. В. Маракуева // Информационное пространство в аспекте гуманитарных и технических наук - 2015: Материалы IV междисциплинарной межвузовской конференции студентов, магистрантов и аспирантов, Барнаул, 25 ноября 2015 года / Ответственный за выпуск А.В. Черенкова. – Барнаул: ООО "Алтай-Циклон", 2015. – С. 36-39. – EDN VXULYP.

2. Маракуева, Н. В. Защищенное облачное хранилище предприятия / Н. В. Маракуева // Проблемы правовой и технической защиты информации: Сборник научных статей, Барнаул, 24 мая 2017 года / ФГБОУ ВПО «Алтайский государственный университет». Том Выпуск V. – Барнаул: Алтайский государственный университет, 2017. – С. 45-50. – EDN ZTOXWB.

3. Садков, А. А. Разработка многопользовательского защищенного хранилища данных / А. А. Садков // Молодой исследователь: вызовы и перспективы: Сборник статей по материалам LXI международной научно-практической конференции. Том 8 (61): Общество с ограниченной ответственностью "Интернаука", 2018. – С. 146-148. – EDN YTJELU.

4. Кочин, В. П. Разработка образовательного защищенного облачного хранилища данных, интегрированного в инфраструктуру образовательного учреждения / В. П. Кочин, А. В. Жерело // Вестник компьютерных и информационных технологий. – 2022. – Т. 19, № 6(216). – С. 21-28. – DOI 10.14489/vkit.2022.06.pp.021-028. – EDN TNVYJK.

5. Бакуменко, В. В. Использование общедоступных облачных хранилищ данных в качестве криптоконтейнеров для защищенного хранения информации в распределенных компьютерных системах / В. В. Бакуменко, С. С. Куликов // Управление информационными рисками и обеспечение безопасности инфокоммуникационных систем. – 2020. – Т. 18, № 1. – С. 72-85. – EDN MBSZEP.

6. Зольников, К.В. Математическая модель оценки показателей надежности сложных программно-технических комплексов / К.В. Зольников, Д.М. Уткин, Ю.А. Чевычелов // Моделирование систем и процессов. – 2018. – Т. 11, № 1. – С. 21-26.

7. Зольников, В.К. Моделирование и анализ производительности алгоритмов балансировки нагрузки облачных вычислений / В.К. Зольников, О.В. Оксюта, Н.Ф. Даюб // Моделирование систем и процессов. – 2020. – Т. 13, № 1. – С. 32-39.

8. Зольников К.В. Система управления распределением работ при проектировании сложных технических систем / Новикова Т.П., Зольников К.В., Кулай А.Ю., Струков И.И. // В сборнике: Информационные технологии в управлении и моделировании мехатронных систем. материалы 1-й научно-практической международной конференции. 2017. С. 199-204

References

1. Marakueva, N.V. Secure file storage / N.V. Marakueva // Information space in the aspect of humanities and technical sciences - 2015: Materials of the IV interdisciplinary interuniversity conference of students, undergraduates and graduate students, Barnaul, November 25, 2015 / Responsible for issue A.V. Cherenkova. – Barnaul: Altai-Cyclone LLC, 2015. – P. 36-39. – EDN VXULYP.

2. Marakueva, N.V. Secure enterprise cloud storage / N.V. Marakueva // Problems of legal and technical information protection: Collection of scientific articles, Barnaul, May 24, 2017 / Altai State University. Volume Issue V. - Barnaul: Altai State University, 2017. - P. 45-50. – EDN ZTOXWB.

3. Sadkov, A. A. Development of a multi-user secure data warehouse / A. A. Sadkov // Young researcher: challenges and prospects: Collection of articles

based on the materials of the LXI international scientific and practical conference. Vol. 8 (61): Limited Liability Company "Internauka", 2018. – pp. 146-148. – EDN YTJELU.

4. Kochin, V. P. Development of an educational secure cloud data storage integrated into the infrastructure of an educational institution / V. P. Kochin, A. V. Zherelo // Bulletin of computer and information technologies. – 2022. – T. 19, No. 6(216). – pp. 21-28. – DOI 10.14489/vkit.2022.06.pp.021-028. – EDN TNVYJK.

5. Bakumenko, V.V. Using public cloud data storage as cryptocontainers for secure storage of information in distributed computer systems / V.V. Bakumenko, S.S. Kulikov // Information risk management and ensuring the security of infocommunication systems. – 2020. – T. 18, No. 1. – P. 72-85. – EDN MBSZEP.

6. Zolnikov, K.V. Mathematical model for assessing reliability indicators of complex software and hardware systems / K.V. Zolnikov, D.M. Utkin, Yu.A. Chevychelov // Modeling of systems and processes. – 2018. – T. 11, No. 1. – P. 21-26.

7. Zolnikov, V.K. Modeling and performance analysis of cloud computing load balancing algorithms / V.K. Zolnikov, O.V. Oksyuta, N.F. Dayub // Modeling of systems and processes. – 2020. – T. 13, No. 1. – P. 32-39.

8. Zolnikov K.V. Control system for the distribution of work in the design of complex technical systems / Novikova T.P., Zolnikov K.V., Kulai A.Yu., Strukov I.I. // In the collection: Information technologies in the control and modeling of mechatronic systems. materials of the 1st scientific and practical international conference. 2017. pp. 199-204.