

# 17.1 Обеспечение безопасности при использовании *InternalsVisibleTo*

User: [ru.chatter@gmail.com](mailto:ru.chatter@gmail.com) ([ru.chatter@gmail.com](mailto:ru.chatter@gmail.com))

Created: 10/9/2025 9:28:38

Updated: 10/9/2025 10:18:19

Exported: 10/9/2025 10:21:52

Link: <https://chatgpt.com/c/68e75607-d918-8325-a82e-4a37cde67346>

## Response:

Отличный вопрос — вот зачем это нужно 🙋

Когда вы пишете код на C#, модификатор `internal` делает тип (класс, структура, интерфейс и т.д.) **доступным только внутри текущей сборки (assembly)**.

Это удобно для инкапсуляции: внешний код (другие проекты) не видят ваши внутренние реализации.

Но иногда возникает ситуация:

💡 вы хотите **тестировать внутренние классы** в проекте **модульного тестирования (Unit Test Project)**, который собирается в **другую сборку**.

А тестовый проект не имеет доступа к `internal`-типам.

## ✖ Пример ситуации

У вас есть основной проект:

```
// Assembly: MyApp.dll
namespace MyApp.InternalLogic
{
    internal class Calculator
    {
        internal int Add(int a, int b) => a + b;
    }
}
```

И есть проект тестов:

```
// Assembly: MyApp.Tests.dll
[TestClass]
public class CalculatorTests
{
    [TestMethod]
    public void Add_works()
    {
        var calc = new Calculator(); // ❌ Ошибка: класс internal, недоступен
    }
}
```

Компилятор выдаст ошибку:

```
"MyApp.InternalLogic.Calculator" is inaccessible due to its protection level."
```

## ✓ Решение: `InternalsVisibleTo`

Чтобы разрешить тестовому проекту видеть внутренние классы, в **основном проекте** нужно добавить специальный атрибут в файл атрибутов (обычно `AssemblyInfo.cs` или отдельный `.cs` с атрибутами):

```
using System.Runtime.CompilerServices;
[assembly: InternalsVisibleTo("MyApp.Tests")]
```

Теперь проект `MyApp.Tests` будет иметь **доступ к internal-типам и членам** сборки `MyApp.dll`.

## ⚙ Где это используют

- В модульном тестировании (самый частый случай).
- В больших системах, где несколько сборок логически связаны, и одна из них — «вспомогательная».
- При написании библиотек, где публичный API ограничен, но тесты должны проверять внутренние механизмы.

## ⚠ Замечание безопасности

Атрибут `InternalsVisibleTo` **работает только для конкретной сборки по имени**.

Если сборка подписана сильным именем (strong name), то нужно указывать **публичный ключ** сборки-получателя. Это предотвращает случайное или вредоносное получение доступа.

Если хочешь, я могу показать короткий пример с тестами на MSTest или xUnit, где `InternalsVisibleTo` реально открывает доступ к `internal` классу — чтобы ты увидела, как это выглядит на практике. Хочешь?

2/2

## Prompt:

Для подписанных сборок нужно указать публичный ключ -- зачем?

## Response:

Отличный, очень точный вопрос 💡

Ты совершенно права — в случае **подписанных сборок** (strong-named assemblies) к атрибуту `[assembly: InternalsVisibleTo("...")]` действительно нужно добавить **публичный ключ**.

И это сделано **из соображений безопасности и подлинности**.

Разберёмся шаг за шагом 🙋

## 🔒 Что такое *подписанная сборка* (strong-named assembly)

Подписанная сборка — это сборка, у которой есть **уникальная криптографическая подпись**.

Эта подпись создаётся с использованием **пары ключей**:

- **приватный ключ** — хранится у разработчика и используется для подписи;
- **публичный ключ** — записывается в метаданные сборки и может быть использован кем угодно для проверки подлинности.

💡 Идея:

сильное имя (Strong Name) гарантирует, что сборку нельзя подменить другой с тем же именем.

То есть:

“MyLibrary.dll” с сильным именем нельзя подделать просто скопировав файл и заменив содержимое.

## ⚙️ Почему нужен публичный ключ в `InternalVisibleTo`

Предположим, у тебя есть библиотека:

```
// myLibrary.dll (подписана strong name)
[assembly: InternalVisibleTo("MyLibrary.Tests")]
```

➡ Если бы C# разрешил так, **любая** сборка, названная `myLibrary.Tests.dll`, могла бы получить доступ к `internal`-типам твоей библиотеки.

Это создаёт **дыру в безопасности** — кто угодно может подделать DLL и получить внутренний доступ.

## 🛡️ Решение

Поэтому для **подписанных сборок** нужно указать не только имя, но и **публичный ключ** сборки, которой ты доверяешь:

```
[assembly: InternalVisibleTo(
    "MyLibrary.Tests,
    PublicKey=0024000004800000940000000602000000240000525341310004000001000100a58b..."
)]
```

Теперь CLR проверяет:

1. Что имя совпадает (`MyLibrary.Tests`);
2. Что **подпись сборки** соответствует указанному публичному ключу.

Только тогда `internal`-типы становятся доступны.

Если кто-то создаст поддельную `myLibrary.Tests.dll` без этого ключа — она не получит доступ.

## ✖ Итого

Сценарий	Нужно указывать PublicKey?	Почему
Сборка без подписи	❌ Нет	Нет проверки подлинности
Сборка с подписью (strong name)	✅ Да	Иначе можно подделать сборку с тем же именем

Если хочешь, я могу показать, как сгенерировать пару ключей (snk) и как подписать сборку — чтобы ты увидела, где этот публичный ключ берётся и как он выглядит на практике. Показать?

## Prompt:

но ведь если ключ открытый, то любой может создать поддельную MyLibrary.Tests.dll

## Response:

Очень умный вопрос 🤔 — и это как раз то, что часто вызывает путаницу, даже у опытных разработчиков. Давай разберём это пошагово.

### 💡 Ключевая идея

Да, **публичный ключ** действительно *открытый* — любой может его увидеть и скопировать. Но важно:

Подписать сборку можно **только** с помощью **приватного ключа**, а не публичного.

### 🔒 Аналогия

Представь:

- Публичный ключ — это как **замочная скважина**: все могут посмотреть, как она выглядит.
- Приватный ключ — это **ключ от замка**: только владелец им владеет и может открыть.

Когда ты создаёшь подписанную сборку:

- Компилятор берёт твой **приватный ключ**.
- Подписывает метаданные и хэш сборки.
- Вставляет **публичный ключ** в её заголовок.

В результате сборка получает **уникальную цифровую подпись**.

## ✖ Проверка при загрузке

Когда CLR (или компилятор) встречает атрибут:

```
[assembly: InternalsVisibleTo(
    "MyLibrary.Tests, PublicKey=002400000480000094000000060200000024000052534131..."
)]
```

он говорит:

«Я открою доступ к internal-типам **только той сборке**,  
которая подписана **парой ключей**, где публичный ключ совпадает с этим.»

Это значит:

- Да, кто угодно может написать сборку с именем `MyLibrary.Tests` и даже вставить туда этот **открытый** ключ в метаданные.
- ❌ Но если у него **нет приватного ключа**, он не сможет **создать правильную цифровую подпись**.
- Поэтому **CLR отвергнет** поддельную сборку при проверке подлинности.

## 🔍 Иными словами

Что делает публичный ключ	Что не делает
Позволяет всем проверить, что сборка подписана «владельцем»	Не позволяет подделать или сгенерировать правильную подпись
Позволяет CLR сравнивать и удостоверяться в подлинности сборки	Не раскрывает приватный ключ и не даёт доступ к подписи

## ⚙️ Итого

- ✅ Публичный ключ можно распространять — он безопасен.
- 🗝️ Только владелец приватного ключа может подписывать настоящие сборки.
- 🔍 CLR сверяет, что подпись действительно соответствует публичному ключу.
- ❌ Подделать `MyLibrary.Tests.dll` невозможно без приватного ключа.

Если хочешь, я покажу **наглядно**, как выглядит процесс подписи сборки ( `sn.exe` , `snk` -файл, и как CLR проверяет подлинность).

Хочешь?

## Prompt:

да

## Response:

Отлично 👍 Тогда разберём **пошагово**, как создаётся и проверяется подпись сборки в .NET — и как это связано с `InternalsVisibleTo`.

### ❧ 1. Генерация пары ключей (private + public)

.NET использует инструмент **Strong Name Tool** (`sn.exe`), который входит в SDK.

Его можно запустить из *Developer Command Prompt for Visual Studio*.

Создадим пару ключей:

```
sn -k myKey.snk
```

Это создаст файл `myKey.snk`, который содержит **и приватный, и публичный ключ**.

### ❧ 2. Подписание сборки этим ключом

Допустим, у нас есть библиотека `myLibrary.csproj`.

Её можно подписать двумя способами:

#### ✅ В Visual Studio

- Открыть свойства проекта → вкладка **Signing** (Подпись)
- Отметить ☒ «Sign the assembly»
- Указать путь к файлу `myKey.snk`

#### ✅ Через атрибут

Можно также явно прописать в `AssemblyInfo.cs`:

```
[assembly: AssemblyKeyFile("myKey.snk")]
```

⚠️ Обычно так не делают в современных проектах, потому что Visual Studio сама управляет ключом.

### ❧ 3. Проверка, что сборка действительно подписана

Чтобы проверить подпись, выполняем:

```
sn -vf myLibrary.dll
```

Если всё в порядке, увидим:

```
Assembly 'myLibrary.dll' is valid
```

## ❧ 4. Извлечение публичного ключа

Публичную часть можно достать так:

```
sn -p MyKey.snk MyKey.publickey
```

А затем получить её в текстовом виде (для вставки в `InternalsVisibleTo`):

```
sn -tp MyKey.publickey
```

Результат будет примерно такой:

```
Public key is
0024000004800000940000000602000000240000525341310004000001000100a58b73...

Public key token is b77a5c561934e089
```

## ❧ 5. Использование в атрибуте `InternalsVisibleTo`

Теперь в **подписанной библиотеке** можно указать:

```
using System.Runtime.CompilerServices;

[assembly: InternalsVisibleTo(
    "MyLibrary.Tests,
    PublicKey=0024000004800000940000000602000000240000525341310004000001000100a58b73..."
)]
```

👉 Это значит:

“Доверяю доступ к моим `internal` типам только той сборке `MyLibrary.Tests`, которая подписана приватным ключом, соответствующим этому публичному ключу.”

## ❧ 6. Проверка подлинности при загрузке

Когда CLR загружает `MyLibrary.Tests.dll`, он делает следующее:

1. Смотрит её **публичный ключ**, записанный в метаданных.
2. Проверяет, что цифровая подпись сборки совпадает с этим ключом.
3. Сравнивает этот публичный ключ с тем, что указан в `InternalsVisibleTo`.

Если всё совпадает — доступ разрешён ✅

Если нет — ошибка компиляции или доступа ❌

## 🔒 Итого: зачем всё это

Этап	Цель
<code>sn -k</code>	Создать пару ключей
Подпись сборки	Сделать сборку уникальной и защищённой
<code>sn -tp</code>	Извлечь публичный ключ
<code>InternalsVisibleTo(..., PublicKey=...)</code>	Разрешить доступ доверенной подписанной сборке
Проверка CLR	Гарантировать, что сборку нельзя подделать

Хочешь, я покажу короткий **живой пример** — два проекта (`myLibrary` и `myLibrary.Tests`) с включённой подписью и `InternalsVisibleTo`, чтобы было видно, как тест получает доступ к `internal`-классу?

## Prompt:

А цифровая подпись вшита прямо в сборку?

## Response:

Отличный вопрос, и да — **👉 цифровая подпись действительно вшита прямо в сборку**.  
Давай разберём подробно, где именно она хранится, как формируется и как CLR её проверяет 🙌

## 🔒 1. Что такое цифровая подпись в сборке .NET

Когда ты подписываешь сборку **сильным именем (Strong Name)**, компилятор вставляет в неё:

- **публичный ключ** — в раздел метаданных (всем виден);
- **цифровую подпись** — в конец сборки, рядом с метаданными и манифестом.

Подпись вычисляется по **всем данным сборки** (код, ресурсы, метаданные) и создаётся с помощью **хэширования + шифрования приватным ключом**.

💡 Это аналог подписи электронных документов — гарантирует целостность и подлинность.

## 📦 2. Где это физически хранится

Сборка (DLL или EXE) состоит из нескольких секций:

```
[ PE Header ]
[ CLR Header ]
[ Metadata Tables ]
[ IL Code ]
[ Strong Name Signature ]
```

- ♦ В разделе **Strong Name Signature** хранится:



- сам **публичный ключ**;
- **хэш** содержимого сборки;
- **подпись**, сделанная приватным ключом (RSA).

### 3. Как работает проверка

Когда CLR загружает сборку:

1. Извлекает её **публичный ключ**.
2. Вычисляет **новый хэш** от содержимого сборки.
3. С помощью **публичного ключа** проверяет, что подпись действительно совпадает с хэшем (т.е. сборка не изменялась).
4. Если проверка не прошла → CLR **не загружает сборку**, выбрасывает исключение «Strong name validation failed».

### 4. Что даёт эта подпись

Возможность	Описание
 Целостность	Никто не может изменить даже 1 байт в сборке — подпись сразу станет недействительной
 Уникальная идентификация	Две сборки с одинаковым именем, но разными ключами — считаются разными
 Надёжная ссылка	Можно безопасно указывать зависимости (CLR знает, какую именно сборку загрузить)
 Совместимость с GAC	Только <b>подписанные сборки</b> можно устанавливать в <b>Global Assembly Cache (GAC)</b>

### 5. Как можно увидеть подпись в реальной DLL

Если открыть сборку утилитой **ILDasm**, ты увидишь примерно так:

```
.assembly MyLibrary
{
    .ver 1:0:0:0
    .publickey = (00240000 04800000 94000000 06020000 ...)
}
```

И в конце файла (если посмотреть в HEX-редакторе) есть блок байт — это и есть **цифровая подпись**.

Также можно проверить сборку:

```
sn -vf MyLibrary.dll
```

Если подпись повреждена, будет сообщение вроде:

```
Failed to verify assembly -- Strong name signature could not be verified.
```

## Итого

Что хранится в сборке	Где	Зачем
Публичный ключ	В метаданных ( <code>.publickey</code> )	Для проверки подлинности
Цифровая подпись	В конце файла	Чтобы защитить от подмены
Хэш содержимого	Внутри подписи	Чтобы проверить, что код не изменён

Хочешь, я покажу схему (диаграмму) — как именно компилятор формирует подпись и где она размещается внутри PE-файла (DLL)?