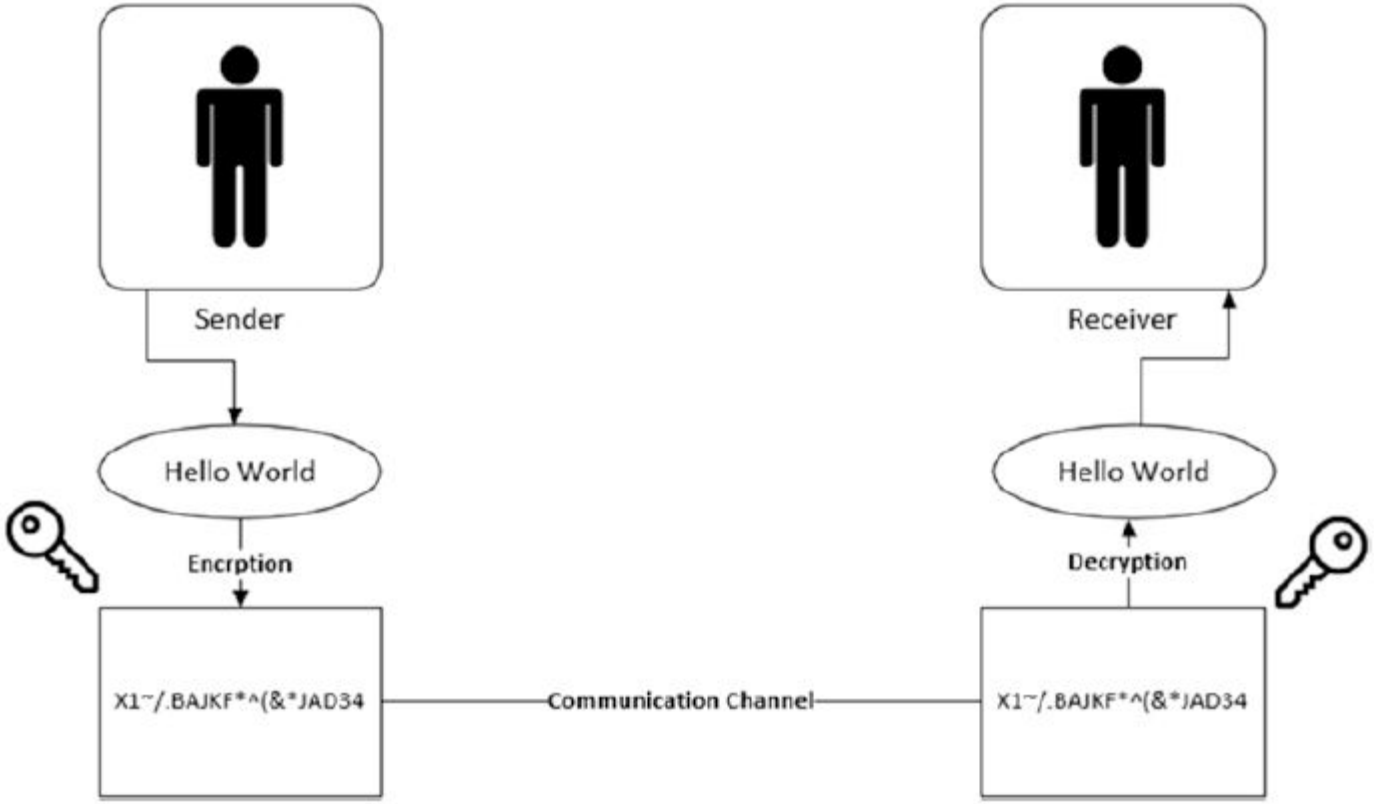
ЛЕКЦИЯ №3 ПО ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Москва, 2020

Процесс шифрования и дешифрования



Криптография C#

Классы криптографии в .NET делятся на три уровня. На первом уровне находится набор абстрактных классов; эти классы определяют задачу шифрования. К ним относятся следующие классы:

AsymmetricAlgorithm

Этот класс представляет асимметричное шифрование, использующее пару ключей "открытый-секретный". Данные, зашифрованные одним ключом, могут быть расшифрованы только другим ключом.

SymmetricAlgorithm

Этот класс представляет симметричное шифрование, использующее разделяемое секретное значение. Данные, зашифрованные с помощью этого ключа, могут быть расшифрованы только посредством того же самого ключа.

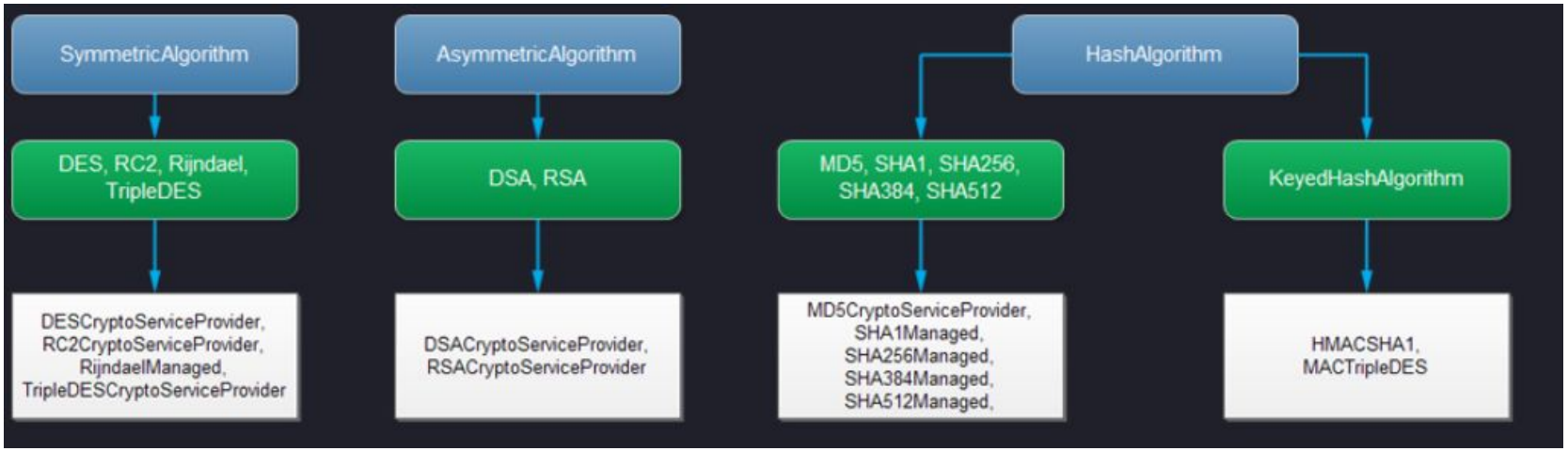
HashAlgorithm

Этот класс представляет генерацию и верификацию хешей. Хеши известны еще и как однонаправленные алгоритмы шифрования, поскольку данные можно только шифровать, но не расшифровывать. Хеши применяются для проверки, не изменялись ли данные.

Криптография C#

Второй уровень содержит классы, представляющие специфические алгоритмы шифрования. Они унаследованы от базовых абстрактных классов и также являются абстрактными. Например, класс DES, реализующий алгоритм DES (Data Encrypting Standard - стандарт шифрования данных), унаследован от SymmetricAlgorithm.

Классы третьего уровня представляют набор реализаций шифрования. Каждый из этих классов унаследован от класса алгоритма второго уровня. Это значит, что такой алгоритм шифрования, как DES, может иметь множество классов реализации.



Криптография C#

Как уже упоминалось, .NET Framework поддерживает три типа шифрования: симметричное, асимметричное и однонаправленное (хеширование). Симметричные алгоритмы всегда используют один и тот же ключ для шифрования и расшифровки. Симметричные алгоритмы работают быстро в обоих направлениях. В таблице ниже перечислены наиболее важные симметричные алгоритмы, поддерживаемые .NET Framework:

Симметричные алгоритмы, поддерживаемые .NET

Абстрактный алгоритм	Реализация по умолчанию	Допустимая длина ключа	Максимальная длина ключа
DES	*DESCryptoServiceProvider*	64	64
TripleDES	*TripleDESCryptoServiceProvider*	128, 192	192
RC2	*RC2CryptServiceProvider*	40 - 128	128
Rijndael	*RijndaelManaged*	128, 192, 256	256

Для большинства случаев хорошим выбором является алгоритм Rijndael. Он обеспечивает высокую производительность и поддерживает ключи большого размера

Криптография C#

Ниже перечислены основные проблемы, связанные с ними:

Передача ключа. При использовании симметричных алгоритмов для обмена данными между двумя приложениями, расположенными на разных машинах, необходим безопасный способ передачи ключа.

Атака "грубой силой". При использовании симметричных ключей в течение длительного времени у злоумышленников может быть достаточно времени для расшифровки трафика простым подбором значений битов ключа. Таким образом, с ростом длины ключа в битах надежность ключа возрастает, как объяснялось ранее. Однако обычно это означает, что все равно периодически нужно менять ключи.

Долговременное управление ключами. Если приходится обновлять ключи через регулярные интервалы времени, их также требуется передавать, что привносит дополнительный риск. Кроме того, ключ должен храниться в безопасном месте.

Системы на основе симметричных алгоритмов шифрования недостаточно безопасны, поэтому и возникает необходимость в асимметричных алгоритмах.

Криптография C#

Шифрование, при котором вы отправляете ключ вместе с данными, чтобы пользователь мог расшифровать данные с помощью того же ключа.
Алгоритм симметричного шифрования работает следующим образом: данные для шифрования преобраз. в блоки шифра, и каждый блок имеет определенный размер для хранения зашифрованных данных. Это называется **цепочка блоков шифров**. Когда данные больше, чем размер блока (размер блока), данные делятся на несколько блоков. Размер блока зависит от используемого алгоритма.
Первый блок содержит зашифрованное значение некоторого случайного значения, называемого вектором инициализации (IV) и ключ шифрования, следующий блок содержит зашифрованное значение предыдущего блока с ключом и так далее. Если размер последнего блока меньше данных, находящихся на нем, блок дополняется.

Криптография C#

Асимметричное шифрование

Асимметричные алгоритмы пытаются решить проблемы, присущие симметричным алгоритмам. Они основаны на математических методах, которые требуют разных ключей для шифрования и расшифровки.

Обычно ключ, используемый для шифрования, называется *открытым ключом (public key)*. Его можно выдать любому, кто желает отправлять вам зашифрованную информацию. С другой стороны, *секретный ключ (private key)* - это единственный ключ, который можно использовать для расшифровки. Таким образом, если вы - единственный владелец секретного ключа, то только вы сможете расшифровать информацию. Этот факт облегчает обмен ключами между различными участниками процесса, поскольку нет необходимости в передаче ключа, с помощью которого можно расшифровать важные данные. В таблице ниже перечислены асимметричные алгоритмы, поддерживаемые .NET Framework:

Асимметричные алгоритмы, поддерживаемые .NET

Абстрактный алгоритм	Реализация по умолчанию	Допустимая длина ключа	Максимальная длина ключа
RSA	*RSACryptoServiceProvider*	384-16384 (с увеличением на 8 бит)	1024
DSA	*DSACryptoServiceProvider*	512-1024 (с увеличением на 64 бита)	1024

Из двух алгоритмов - RSA (его название происходит от фамилий изобретателей алгоритма - Rivest, Shamir, Adleman (Рон Райвест, Ада Шамир, Леонард Адлеман)) и DSA (Digital Signature Algorithm - алгоритм цифровой подписи) - только RSA поддерживает прямое шифрование и расшифровку значений. Алгоритм DSA, как следует из его названия, может использоваться только для подписания информации и верификации подписей.

Криптография C#

Большой проблемой асимметричных алгоритмов является то, что они работают намного медленнее (в зависимости от шифруемых данных) симметричных. Если необходимо передавать данные во множестве запросов, это повлияет на производительность всего приложения.

Поэтому технологии, подобные SSL, используют асимметричные алгоритмы в начале, при установлении сеанса связи. На начальных шагах обмена трафик между клиентом и сервером защищается асимметричным шифрованием (клиент выполняет шифрование с помощью открытого ключа, а сервер осуществляет расшифровку с применением секретного ключа). На данном этапе клиент и сервер могут безопасно обменяться симметричным ключом. Это позволяет объединить преимущества симметричного и асимметричного шифрования. Естественно, должен быть найден способ безопасного хранения секретного ключа, чтобы неавторизованный персонал не имел шансов получить доступ к нему.

Для шифрования и расшифровки важной информации потребуется выполнить следующие шаги (они будут подробно объясняться в последующих разделах):

1. Выберите и создайте алгоритм.
2. Сгенерируйте и сохраните секретный ключ.
3. Зашифруйте или расшифруйте информацию через CryptoStream.
4. Закройте соответствующим образом исходный и целевой потоки.

После создания и тестирования своих служебных классов шифрования должна быть создана база данных для хранения секретной информации и написан код для шифрования и расшифровки этой секретной информации в базе данных.

Криптография C#

Большой проблемой асимметричных алгоритмов является то, что они работают намного медленнее (в зависимости от шифруемых данных) симметричных. Если необходимо передавать данные во множестве запросов, это повлияет на производительность всего приложения.

Поэтому технологии, подобные SSL, используют асимметричные алгоритмы в начале, при установлении сеанса связи. На начальных шагах обмена трафик между клиентом и сервером защищается асимметричным шифрованием (клиент выполняет шифрование с помощью открытого ключа, а сервер осуществляет расшифровку с применением секретного ключа). На данном этапе клиент и сервер могут безопасно обменяться симметричным ключом. Это позволяет объединить преимущества симметричного и асимметричного шифрования. Естественно, должен быть найден способ безопасного хранения секретного ключа, чтобы неавторизованный персонал не имел шансов получить доступ к нему.

Для шифрования и расшифровки важной информации потребуется выполнить следующие шаги (они будут подробно объясняться в последующих разделах):

1. Выберите и создайте алгоритм.
2. Сгенерируйте и сохраните секретный ключ.
3. Зашифруйте или расшифруйте информацию через CryptoStream.
4. Закройте соответствующим образом исходный и целевой потоки.

После создания и тестирования своих служебных классов шифрования должна быть создана база данных для хранения секретной информации и написан код для шифрования и расшифровки этой секретной информации в базе данных.

Криптография C#

```
public static class MyEncryptionUtility
{
// Tссс!! Не говорите никому!
private const string MyKey = "m$%&kljasldk$%/65asjdl";

public static byte[] Encrypt(string data)
{
    // Использовать "MyKey" для шифрования данных
    return null;
}
}
```

Подобного рода ключи легко раскрыть, используя инструменты дизассемблирования. Достаточно просто запустить средство ILDASM и проанализировать класс.

Криптография C#

```
//specify the data
string plainData = "Secret Message";
//convert into bytes of array
byte[] plainDataInBytes = Encoding.UTF8.GetBytes(plainData);
//Create a default cryptography object used to perform symmetric encryption
SymmetricAlgorithm symmetricAlgo = SymmetricAlgorithm.Create();

//Create encryptor with key and IV (Optional)
ICryptoTransform encryptor = symmetricAlgo.CreateEncryptor(symmetricAlgo.Key,
symmetricAlgo.IV);
byte[] cipherDataInBytes = encryptor.TransformFinalBlock(plainDataInBytes, 0,
 plainDataInBytes.Length);
//get the bytes of encrypted data into string
string cipherData = Encoding.UTF8.GetString(cipherDataInBytes);
Console.WriteLine("Encrypted Data is: " + cipherData);
```

Данные должны быть в байтах, поскольку System.Security.Cryptography работает с байтами данных для шифрования. SymmetricAlgorithm - это абстрактный класс симметричных алгоритмов (Aes, DES и т. Д.). Можно использовать его метод Create для создания объекта по умолчанию для криптографии. По умолчанию он использует RijndaelManaged алгоритм (управляемая версия алгоритма Рейндаэля). Вы можете дать название любому симметричному алгоритму в
Создайте метод или можете создать экземпляр из них.
После указания алгоритма вы указываете ключ и IV (которые необязательны) и создаете шифратор.
TransformFinalBlock используется для преобразования данных в байтах для шифрования текста

Криптография C#

Симметричные алгоритмы требуют создания ключа и вектора инициализации (IV). Ключ следует хранить в тайне от любого, кто не должен расшифровывать ваши данные. Вектор инициализации может не быть секретным, но должен изменяться для каждого сеанса.

Симметричные ключи

Классы симметричного шифрования, предоставляемые платформой .NET Framework, требуют ключ и новый вектор инициализации (IV) для шифрования и расшифровки данных. При создании нового экземпляра одного из управляемых симметричных криптографических классов с помощью конструктора без параметров автоматически создаются новые ключ и вектор инициализации. Все пользователи, которым вы разрешаете расшифровывать свои данные, должны иметь тот же ключ и вектора инициализации и использовать тот же алгоритм. Как правило, новый ключ и вектор инициализации должны создаваться для каждого сеанса, и ни вектор инициализации, ни ключ нельзя сохранять для использования в следующем сеансе.

При передаче симметричного ключа и вектора инициализации на удаленную сторону симметричный ключ обычно шифруется с помощью асимметричного шифрования. Отправка ключа через незащищенную сеть без шифрования небезопасна, так как любой, кто перехватит ключ и вектор инициализации, сможет расшифровать данные.

Криптография C#

```
//Create a default cryptography object used to perform symmetric encryption
SymmetricAlgorithm symmetricAlgo2 = SymmetricAlgorithm.Create();
ICryptoTransform decryptor = symmetricAlgo2.CreateDecryptor(key,
algoIV);
byte[] plainDataInBytes2 = decryptor.TransformFinalBlock(cipherDataInBytes, 0,
cipherDataInBytes.Length);
string plainData2 = Encoding.UTF8.GetString(plainDataInBytes);
Console.WriteLine("Decrypted Data is: " + plainData2);
```

Почему Python

- Автоматизация простых рутинных задач
- Научные приложения
- Обычные desktop-приложения
- Android
- Web (Django и Flask фреймворки)
- Машинное обучение (библиотека Tenser Flow)

Почему Python

- Краткий и элегантный синтаксис
- Множество доступных бесплатных библиотек (open source)
- Легко изучаем
- Невероятно популярен среди серьёзных компаний

Python технически

- Язык программирования общего назначения
- Строго типизированный
- Динамический
- Интерпретируемый
  (предварительно компилируется в байт-код)

- Мультипарадигмальный
- CPython – основная реализация Python

Питон 3

- Python 3 это будущее
- Python 2 и Python 3 частично несовместимы
- Поддержка Python 2 полностью прекратится в 2020 году
- Знаешь Python 3? Быстро перейдёшь и на Python 2!
- Python 3 работает с Unicode по умолчанию
- Большинство библиотек переписаны на Python 3

Notebook-оболочка

- Поддерживает markdown и визуализации
- Легко создавать документированный код
- Удобный режим REPL (read-evaluate-print loop)
- Идеально для изучения ЯП (языка программирования)
- Jupyter Notebook – самый популярный
- Расширение файлов - .ipynb

Len and Count

```
In [6]: x="hello, my name is Elias"

In [7]: len(x)
Out[7]: 23

In [8]: x[len(x)-1]
Out[8]: 's'

In [9]: x.count('l')
Out[9]: 3
```

Casing

In [10]:
```
x.capitalize()
```

Out[10]:
```
'Hello, my name is elias'
```

In [14]:
```
upper_cased=x.upper()
print(upper_cased)
```

```
HELLO, MY NAME IS ELIAS
```

In [18]:
```
lower_cased=upper_cased.lower()
print(lower_cased)
```

```
hello, my name is elias
```

In [22]:
```
print(upper_cased.isupper())
print(lower_cased.islower())
print(x.isupper())
print(x.islower())
```

Casing

In [10]:
```
x.capitalize()
```

Out[10]:
```
'Hello, my name is elias'
```

In [14]:
```
upper_cased=x.upper()
print(upper_cased)
```

```
HELLO, MY NAME IS ELIAS
```

In [18]:
```
lower_cased=upper_cased.lower()
print(lower_cased)
```

```
hello, my name is elias
```

In [22]:
```
print(upper_cased.isupper())
print(lower_cased.islower())
print(x.isupper())
print(x.islower())
```