Алгоритм шифрования **AES**

**AES (Advanced Encryption Standard; также Rijndael)** — симметричный алгоритм блочного шифрования (размер блока 128 бит, ключ 128/192/256 бит), принятый в качестве стандарта шифрования правительством США по результатам конкурса AES.

Этот алгоритм хорошо проанализирован и сейчас широко используется, как это было с его предшественником DES. Национальный институт стандартов и технологий США (NIST) опубликовал спецификацию AES 26 ноября 2001 года после пятилетнего периода, в ходе которого были созданы и оценены 15 кандидатур. 26 мая 2002 года AES был объявлен стандартом шифрования.

По состоянию на 2009 год AES является одним из самых распространённых алгоритмов симметричного шифрования. Поддержка ускорения AES была введена фирмой Intel в семейство процессоров x86 начиная с Arrandale в 2010 году, а затем на процессорах Sandy Bridge; фирмой AMD — в Bulldozer с 2011 года.

**История AES**

2 января 1997 года NIST объявляет о намерении выбрать преемника для DES, являвшегося американским стандартом с 1977 года. 2 октября 2000 года было объявлено, что победителем конкурса стал алгоритм Rijndael, и началась процедура стандартизации. 28 февраля 2001 года был опубликован проект, а 26 ноября 2001 года AES был принят как FIPS 197. Историческую ретроспективу конкурса можно проследить на веб-сайте NIST.

**Описание алгоритма AES**

**Определения**

|  |  |
| --- | --- |
| **Block** | последовательность бит, из которых состоит input, output, State и Round Key. Также под Block можно понимать последовательность байтов |
| **Cipher Key** | секретный криптографический ключ, который используется Key Expansion процедурой, чтобы произвести набор ключей для раундов (Round Keys); может быть представлен как прямоугольный массив байтов, имеющий четыре строки и *Nk* столбцов |
| **Ciphertext** | выходные данные алгоритма шифрования |
| **Key Expansion** | процедура генерации Round Keys из Cipher Key |
| **Round Key** | Round Keys получаются из Cipher Key использованием процедуры Key Expansion. Они применяются к State при шифровании и расшифровании |
| **State** | промежуточный результат шифрования, который может быть представлен как прямоугольный массив байтов, имеющий 4 строки и *Nb* столбцов |
| **S-box** | нелинейная таблица замен, использующаяся в нескольких трансформациях замены байтов и в процедуре Key Expansion для взаимнооднозначной замены значения байта. (Предварительно рассчитанный S-box вынесен в отдельный файл в программе) |
| **Nb** | число столбцов (32-битных слов), составляющих *State*. Для AES *Nb* = 4 |
| **Nk** | число 32-битных слов, составляющих шифроключ. Для AES *Nk* = 4, 6, или 8 |
| **Nr** | число раундов, которое является функцией *Nk* и *Nb*. Для AES *Nr* = 10, 12, 14 |

|  |  |
| --- | --- |
| **Вспомогательные процедуры** | |
| **AddRoundKey()** | трансформация при шифровании и обратном шифровании, при которой Round Key XOR’ится c State. Длина RoundKey равна размеру State (то есть если *Nb* = 4, то длина RoundKey равна 128 бит или 16 байт) |
| **InvMixColumns()** | трансформация при расшифровании, которая является обратной по отношению к MixColumns() |
| **InvShiftRows()** | трансформация при расшифровании, которая является обратной по отношению к ShiftRows() |
| **InvSubBytes()** | трансформация при расшифровании, которая является обратной по отношению к SubBytes() |
| **MixColumns()** | трансформация при шифровании, которая берёт все столбцы State и смешивает их данные (независимо друг от друга), чтобы получить новые столбцы |
| **RotWord()** | функция, использующаяся в процедуре Key Expansion, которая берёт 4-байтовое слово и производит над ним циклическую перестановку |
| **ShiftRows()** | трансформации при шифровании, которые обрабатывают State, циклически смещая последние три строки State на разные величины |
| **SubBytes()** | трансформации при шифровании, которые обрабатывают State, используя нелинейную таблицу замещения байтов (S-box), применяя её независимо к каждому байту State |
| **SubWord()** | функция, используемая в процедуре Key Expansion, которая берёт на входе четырёхбайтовое слово и, применяя S-box к каждому из четырёх байтов, выдаёт выходное слово |

**Шифрование**

1. **Генерация ключа**:

* Создать секретный ключ с длиной, кратной 128, 192 или 256 битам (AES-128, AES-192 или AES-256).
* Этот ключ используется как для шифрования, так и для дешифрования.

1. **Генерация вектора инициализации (IV)**:

* Сгенерировать случайный вектор инициализации размером 128 бит (16 байт) для режима CBC.
* Вектор инициализации нужен для обеспечения уникальности процесса шифрования каждого сообщения.

1. **Подготовка данных к шифрованию**:

* Преобразовать текст или данные, которые нужно зашифровать, в байтовый формат (например, в кодировке UTF-8).
* Если длина данных не кратна размеру блока (16 байт для AES), применить заполнение (padding), чтобы довести их до нужного размера. В AES обычно используется заполнение PKCS5 или PKCS7.

1. **Настройка шифра**:

* Инициализировать объект шифра AES с использованием следующих параметров:

1. **Алгоритм**: AES
2. **Режим шифрования**: CBC (Cipher Block Chaining)
3. **Заполнение**: PKCS5Padding (или PKCS7)

* Задать режим ENCRYPT\_MODE (режим шифрования) и передать в объект шифра сгенерированный ключ и вектор инициализации.

1. **Шифрование данных**:

* Выполнить шифрование данных с помощью метода шифра doFinal.
* Этот метод преобразует исходные данные (plaintext) в зашифрованные данные (ciphertext), которые также будут в байтовом формате.

1. **Кодирование зашифрованных данных** (опционально):

* Преобразовать зашифрованные данные в удобный для хранения или передачи формат, например, с использованием кодирования Base64, чтобы представить их в виде строки.

1. **Сохранение ключа и IV**:

* Хранить ключ и IV в безопасном месте. При дешифровании потребуется использовать те же значения, чтобы успешно восстановить исходные данные.

Как реализуется шифрование внутри функции difinal?

**AES** является стандартом, основанным на алгоритме Rijndael. Для AES длина input (блока входных данных) и State (состояния) постоянна и равна 128 бит, а длина шифроключа **K** составляет 128, 192, или 256 бит. При этом исходный алгоритм Rijndael допускает длину ключа и размер блока от 128 до 256 бит. Для обозначения выбранных длин **input, State и Cipher Key** в 32-битных словах используется нотация Nb = 4 для **input** и **State**, Nk = 4, 6, 8 для **Cipher Key** соответственно для разных длин ключей.

В начале зашифровывания **input** копируется в массив **State** по правилу **state[r,c] = input[r+4c]**state[r,c]=input[r+4c], для **0<=r < 4 и 0 <= c < Nb**0≤r<40≤c<Nb. После этого к **State** применяется процедура **AddRoundKey**(), и затем **State** проходит через процедуру трансформации (раунд) 10, 12, или 14 раз (в зависимости от длины ключа), при этом надо учесть, что последний раунд несколько отличается от предыдущих. В итоге, после завершения последнего раунда трансформации, State копируется в **output** по правилу **output[r+4c]=state[r,c]output[r+4c] = stat[r,c]**, **для 0≤r<4 0 <= r < 4 и 0 <=c Nb.**0≤c<Nb

Для реализации этого алгоритма на java я использую библиотеку Java Cryptography

Architecture (JCA):  
import javax.crypto.Cipher;  
import javax.crypto.KeyGenerator;  
import javax.crypto.SecretKey;  
import javax.crypto.spec.IvParameterSpec;  
import javax.crypto.spec.SecretKeySpec;  
import java.security.SecureRandom;  
import java.util.Base64;  
  
public class AESCipherExample {  
 public static void main(String[] args) throws Exception {  
 // Данные для шифрования  
 String plainText = "Hello, world! (encrypting with AES)";  
  
 // Генерация случайного ключа  
 KeyGenerator keyGen = KeyGenerator.*getInstance*("AES");  
 keyGen.init(128); // 128-битный ключ для AES-128  
 SecretKey secretKey = keyGen.generateKey();  
  
 // Генерация вектора инициализации (IV)  
 byte[] iv = new byte[16];  
 SecureRandom random = new SecureRandom();  
 random.nextBytes(iv);  
 IvParameterSpec ivSpec = new IvParameterSpec(iv);  
  
 // Шифрование данных  
 Cipher cipher = Cipher.*getInstance*("AES/CBC/PKCS5Padding");  
 cipher.init(Cipher.*ENCRYPT\_MODE*, secretKey, ivSpec);  
 byte[] encryptedBytes = cipher.doFinal(plainText.getBytes("UTF-8"));  
 String encryptedText = Base64.*getEncoder*().encodeToString(encryptedBytes);  
 System.*out*.println("Зашифрованный текст: " + encryptedText);  
  
 // Дешифрование данных  
 cipher.init(Cipher.*DECRYPT\_MODE*, secretKey, ivSpec);  
 byte[] decryptedBytes = cipher.doFinal(Base64.*getDecoder*().decode(encryptedText));  
 String decryptedText = new String(decryptedBytes, "UTF-8");  
 System.*out*.println("Расшифрованный текст: " + decryptedText);  
 }  
}

Зашифрованный текст: 41nXFWGKbRV8z/VO4n0dcOXrYpBy88kdtjAAiwwW19v7HvivxOdkA+DYS/rxiYff

Расшифрованный текст: Hello, world! (encrypting with AES)

Как видно из результата, исходный текст совпадает с текстом, после процедуры расшифровки, соответственно, шифрование и дешифрование произведены верно.