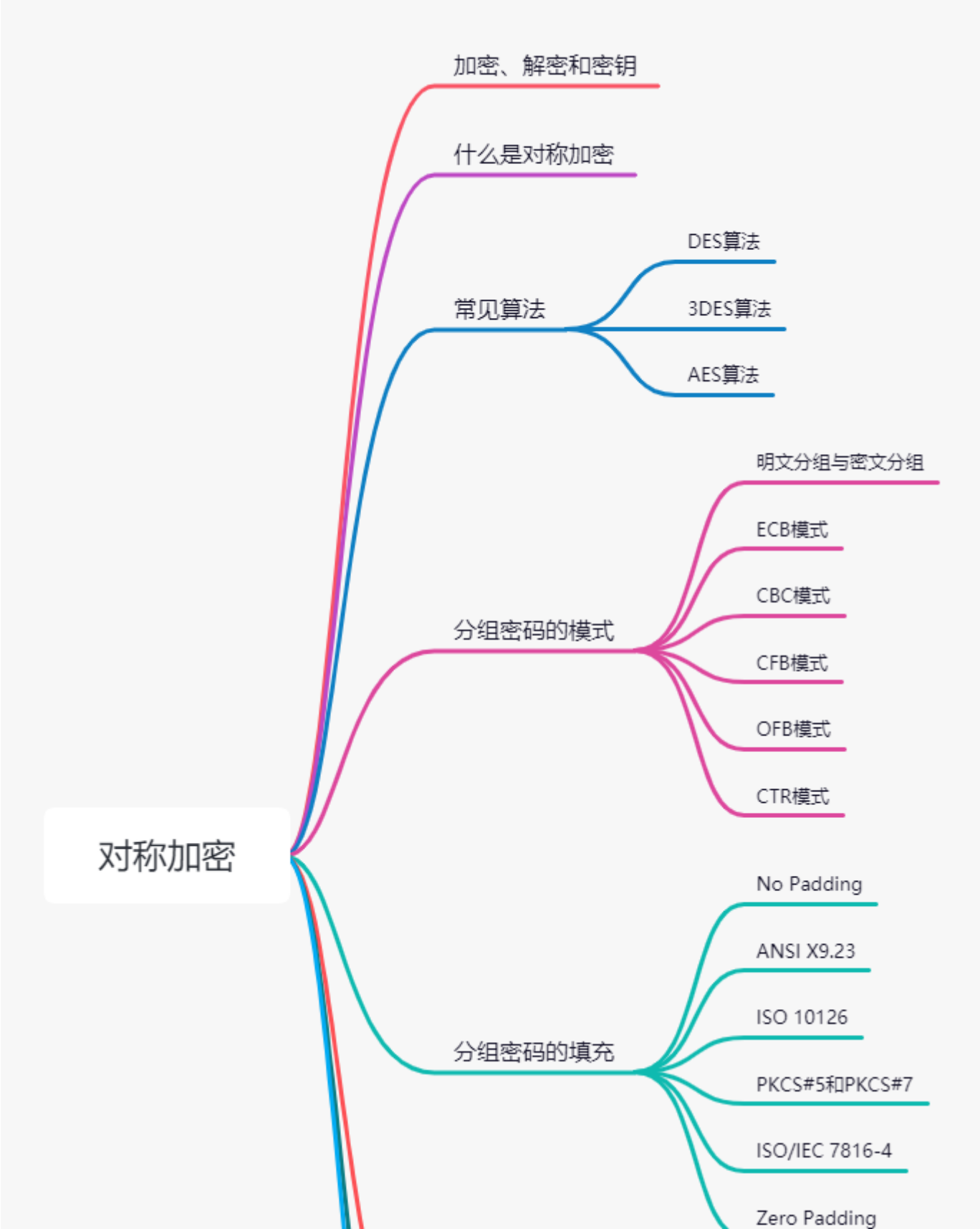
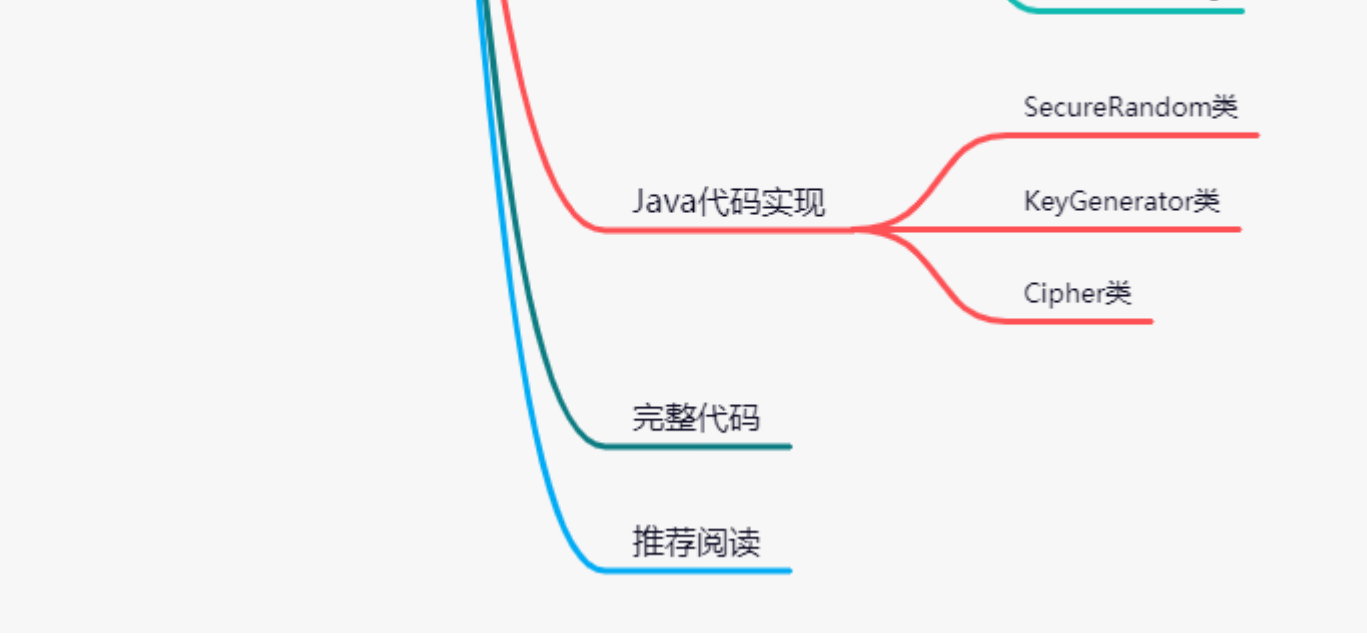
一文搞懂对称加密：加密算法、工作模式、填充方式、代码实现

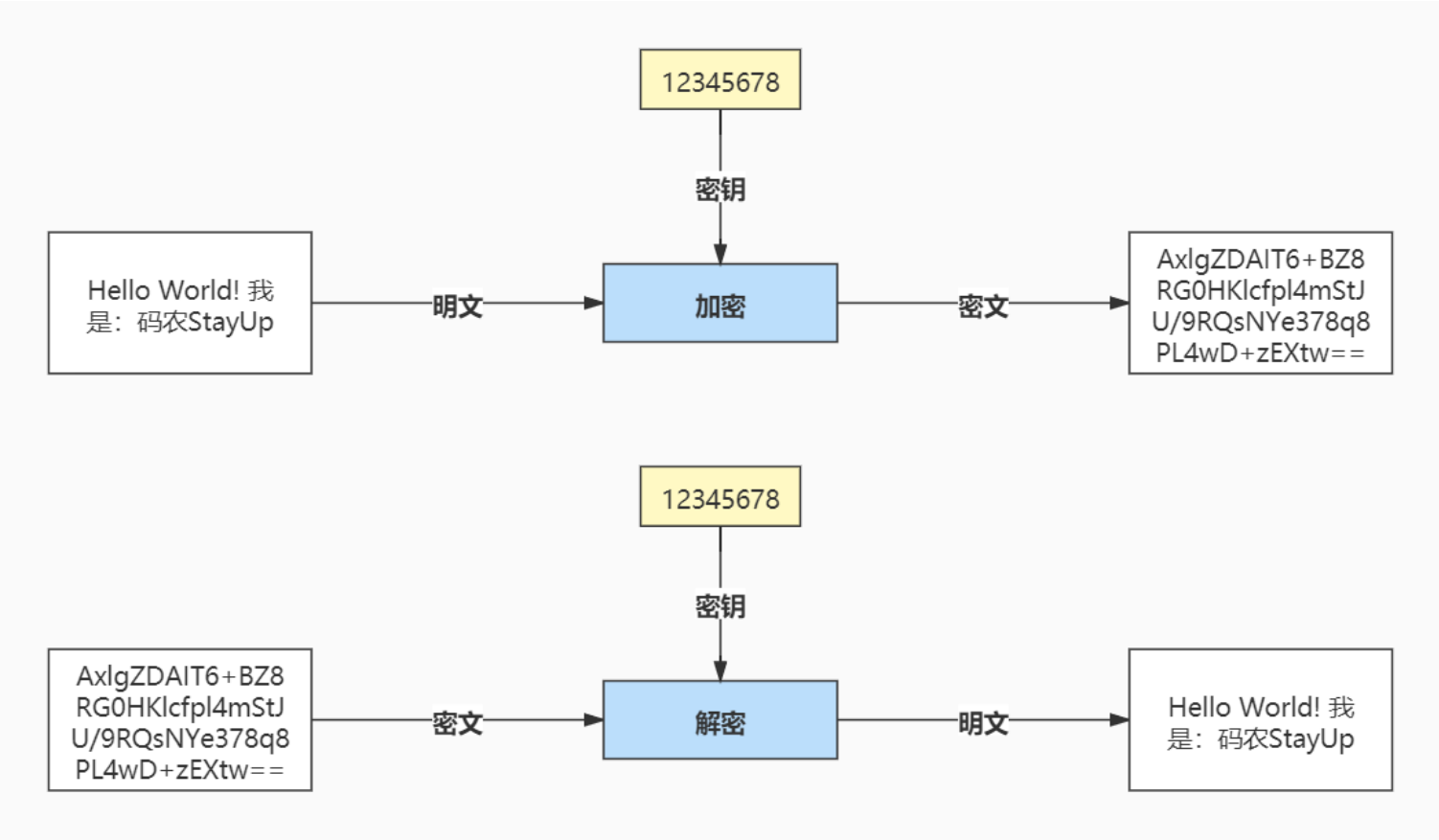
> 微信搜索：码农StayUp
> 主页地址：https://gozhuyinglong.github.io
> 源码分享：https://github.com/gozhuyinglong/blog-demos





加密、解密和密钥

加密（Encrypt）是从明文生成密文的步骤，**解密**（Decrypt）是从密文还原成明文的步骤，而这两个步骤都需要用到**密钥**（Key）。这和我们现实中，用钥匙上锁和开锁是一样的。



什么是对称加密

对称加密（Symmetric Cryptography）是密码学中的一类加密算法，这类算法在加密和解密时，使用相同的密钥。

对称加密又称为**共享密钥加密**，其最大的缺点是，对称加密的安全性依赖于密钥，一旦泄露，就意味着任何人都能解密消息。

对称加密的优点是加密速度快，所以在很多场合被使用。

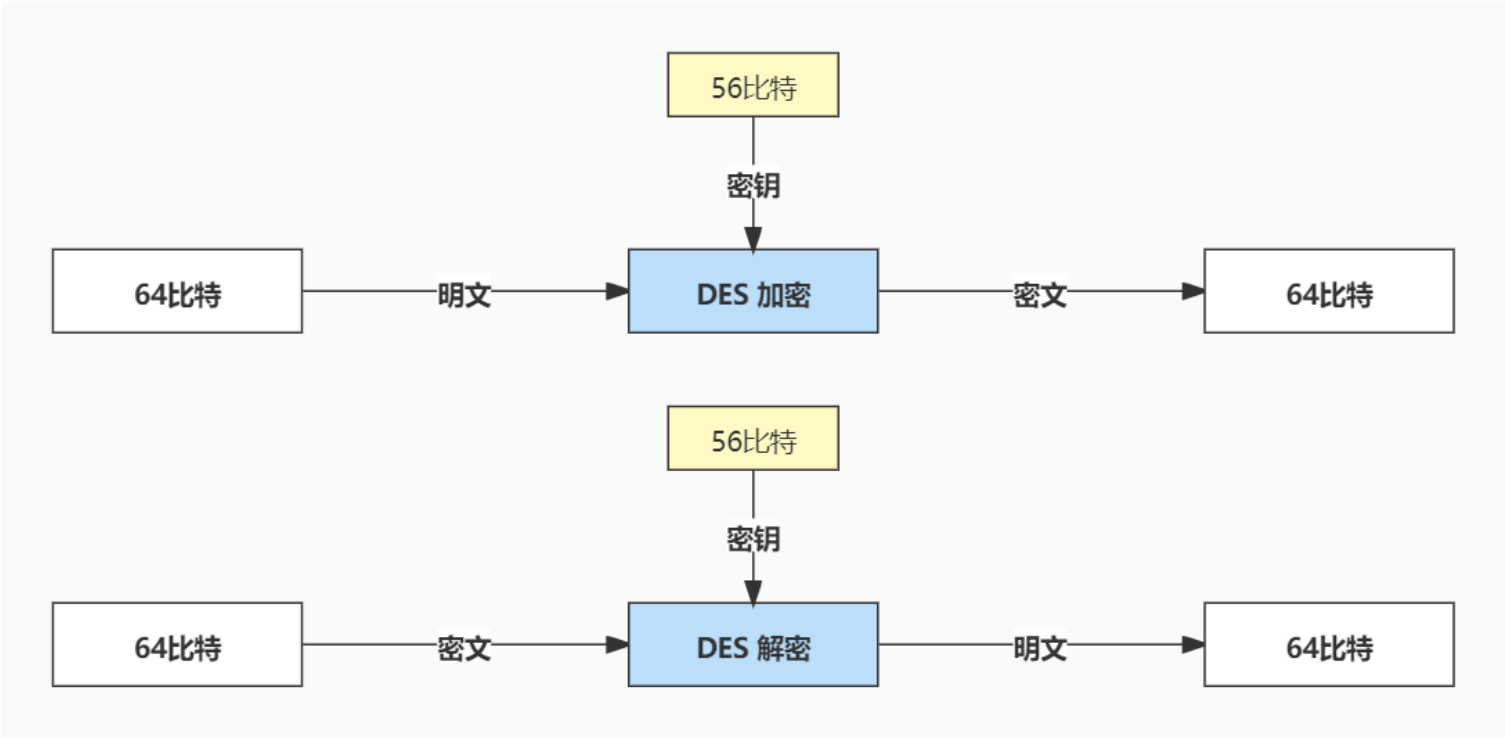
常见算法

本节介绍对称加密的一些常见算法，包括DES、3DES和AES。

DES算法

DES（Data Encryption Standard，中文：数据加密标准），是一种对称加密算法。该算法在1976年被美国联邦政府的国家标准局确定为联邦资料处理标准（FIPS），并于1977年被发布，随后在国际上广泛流传开来。然而，随着计算机的进步，DES 已经能够被暴力破解，所以该算法已经不安全了。

DES是一种**分组密码**（Block Cipher，或者叫**块加密**），即将明文按64比特进行分组加密，每组生成64位比特的密文。它的密钥长度为56比特（从规格上来说，密钥长度是64比特，但由于每隔7比特会设置一个用于错误检查的比特，因此实际长度为56比特）。



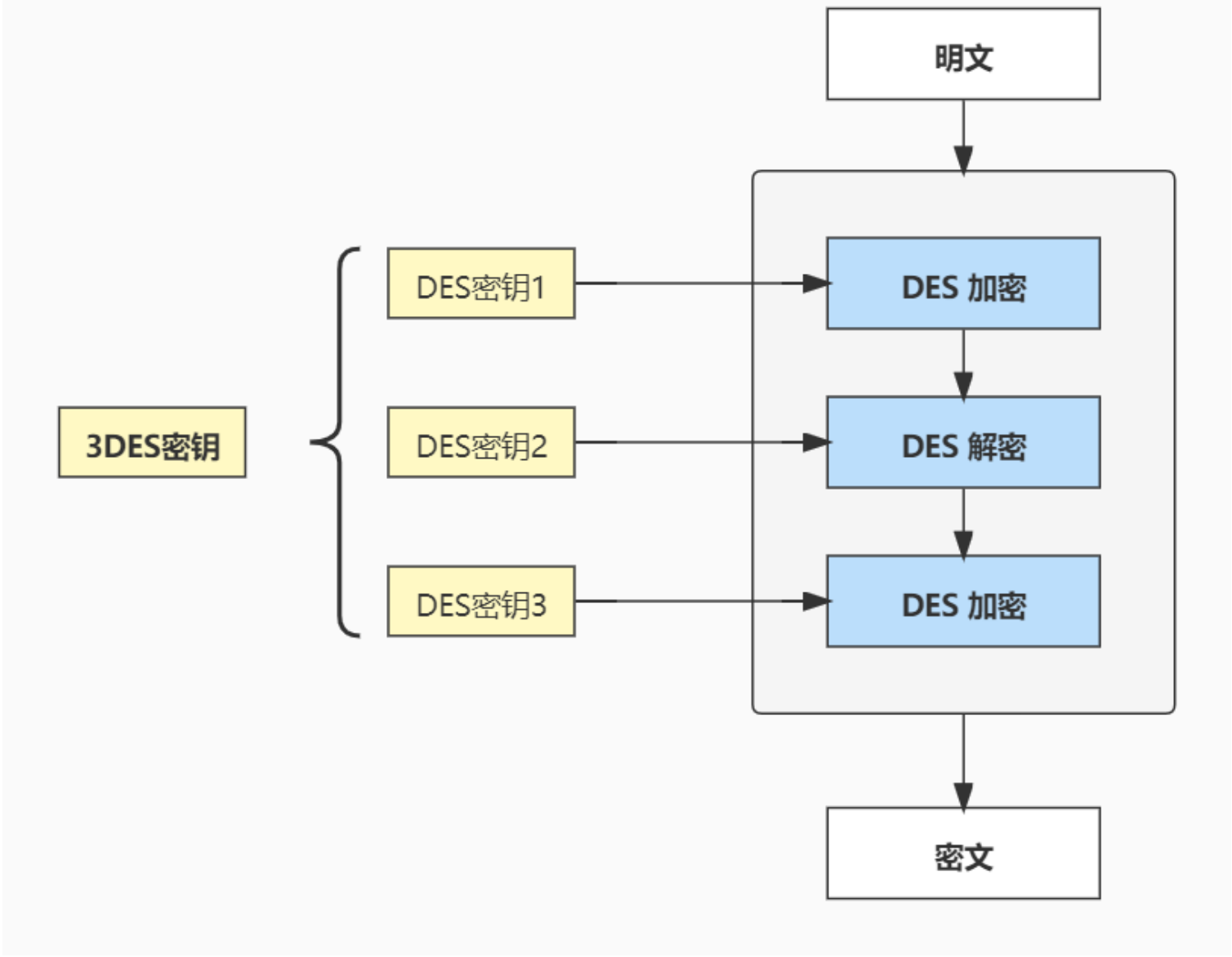
3DES算法

三重数据加密算法（Triple Data Encryption Algorithm，缩写为TDEA），简称**3DES**（Triple-DES），是DES的增强版，相当于对每组数据应用了三次DES算法。

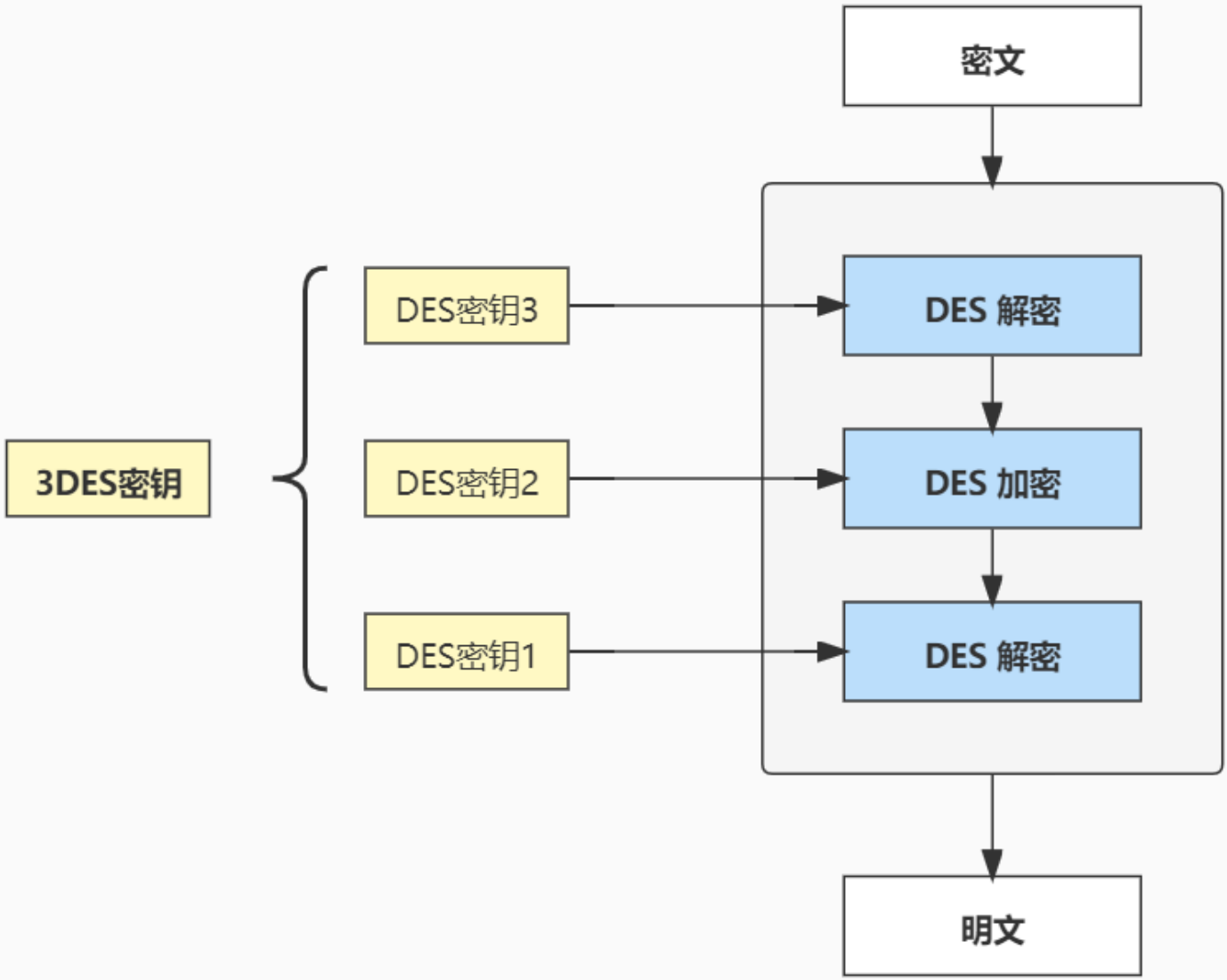
由于DES算法的密钥长度过短，容易被暴力破解，为了解决这一问题，设计出了该算法。它使用简单的方法，通过增加DES密钥长度的方式来避免类似攻击，而不是一种全新的密码算法。

该算法在每次应用DES时，使用不同的密钥，所以有三把独立密钥。这三把密钥组成一起，是一个长度为168（56 + 56 + 56）比特的密钥，所以3DES算法的密钥总长度为168比特。

3DES的加密过程，并不是进行三次DES加密（加密→加密→加密），而是以**密钥1、密钥2、密钥3**的顺序，进行**加密→解密→加密**的过程。



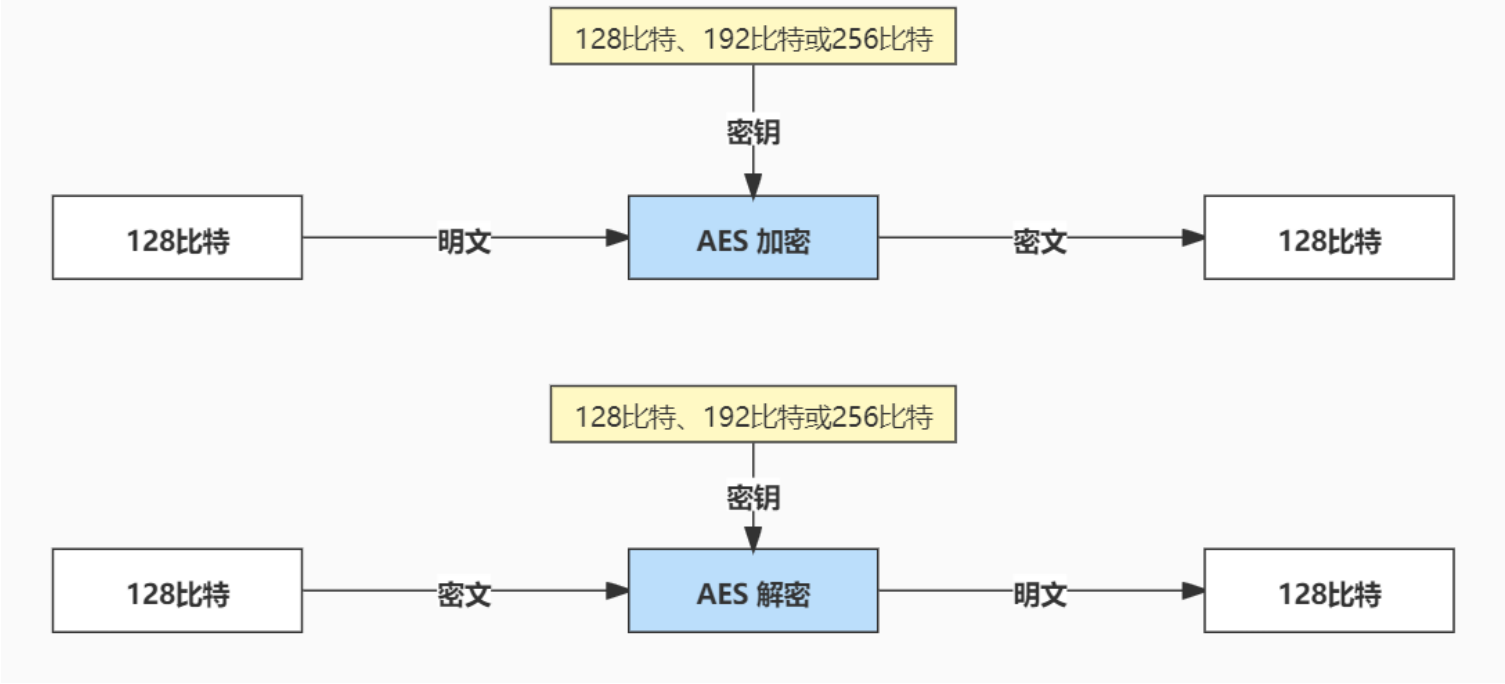
3DES的解密过程和加密正好相反，是以**密钥3、密钥2、密钥1**的顺序，进行**解密→加密→解密**的操作。



AES算法

AES（Advanced Encryption Standard），即高级加密标准，是取代DES算法的一种新的对称加密算法。AES算法是从全世界的企业和密码学家，提交的对称密码算法中竞选出来的，最终 Rijndael 加密算法胜出，所以AES又称为 **Rijndael** 加密算法。

AES也是一种分组密码，它的分组长度为128比特，密钥长度可以为128比特、192比特或256比特。



分组密码的模式

上面介绍的DES、3DES和AES都属于分组密码，它们只能加密固定长度的明文。如果需要加密更长的明文，就需要对分组密码进行迭代，而分组密码的迭代方法称为分组密码的**模式**（Model）。简而一句话：分组密码的模式，就是分组密码的迭代方式。

分组密码有很多种模式，这里主要介绍以下几种：ECB、CBC、CFB、OFB、CTR。

明文分组与密文分组

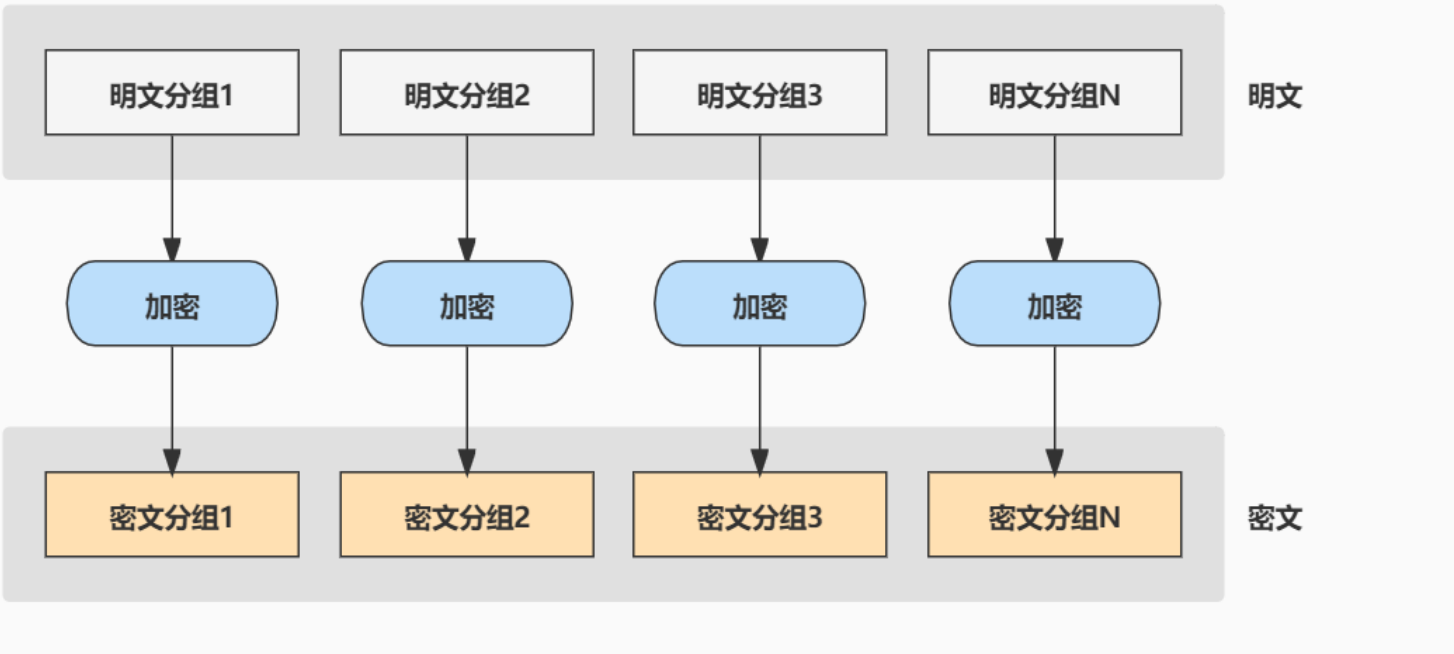
在下面对模式的介绍时，会用到两个术语，这里先介绍一下：

在分组密码中，我们称每组的明文为**明文分组**，每组生成的密文称为**密文分组**。

若将所有的明文分组合并起来就是完整的明文（先忽略填充），将所以的密文分组合并起来就是完整的密文。

ECB模式

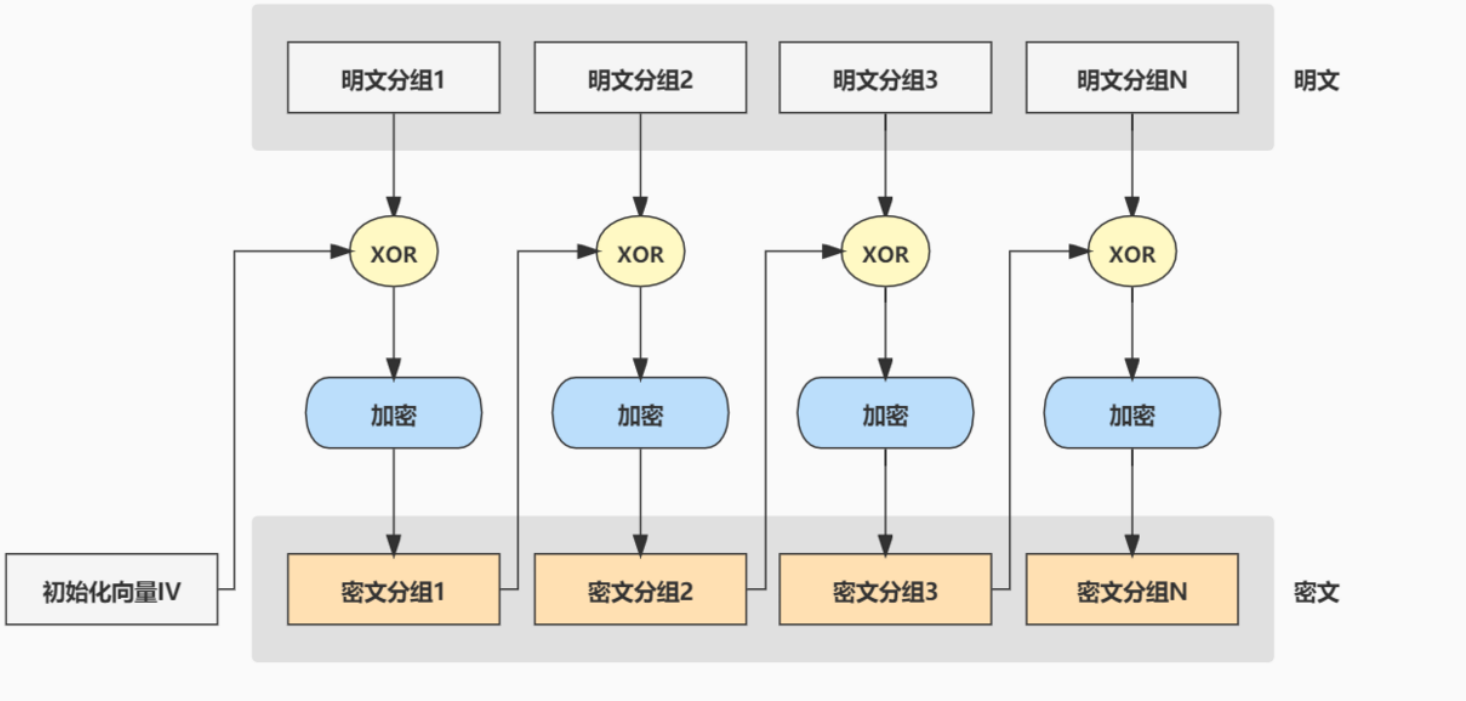
ECB（Electronic CodeBook）模式，即电子密码本模式。该模式是将明文分组，加密后直接成为密文分组，分组之间没有关系。



ECB模式是所有模式中最简单的一种，该模式的明文分组与密文分组是一一对应的关系，若明文分组相同，其密文分组也一定相同。因此，ECB模式也是最不安全的模式。

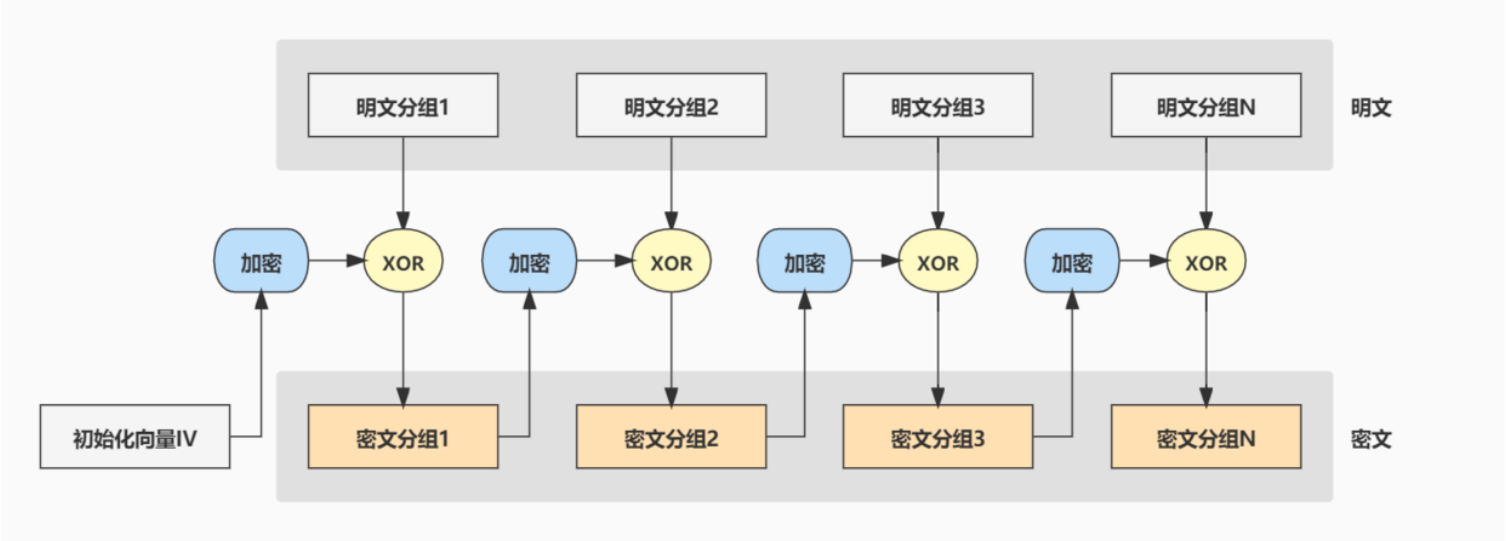
CBC模式

CBC（Cipher Block Chaining）模式，即密码分组链接模式。该模式首先将明文分组与前一个密文分组进行XOR运算，然后再进行加密。只有第一个明文分组特殊，需要提前为其生成一个与分组长度相同的比特序列，进行XOR运算，这个比特序列称为**初始化向量**（Initialization Vector），简称**IV**。



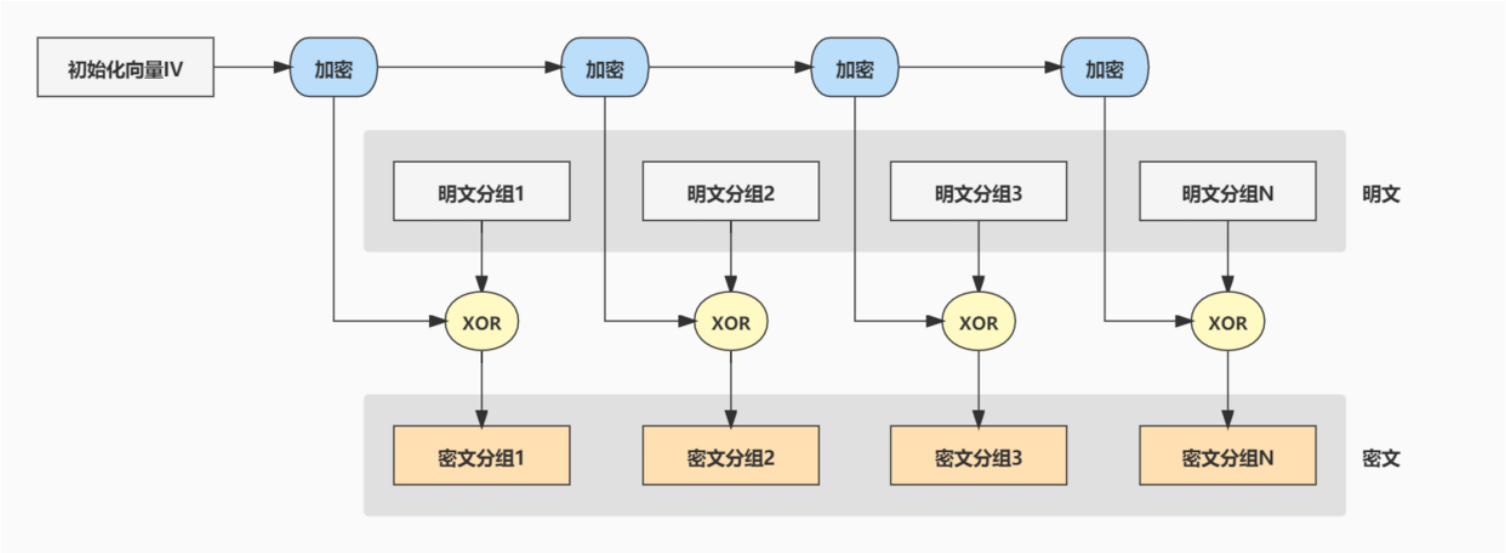
CFB模式

CFB（Cipher FeedBack）模式，即密文反馈模式。该模式首先将前一个密文分组进行加密，再与当前明文分组进行XOR运算，来生成密文分组。同样CFB模式也需要一个IV。



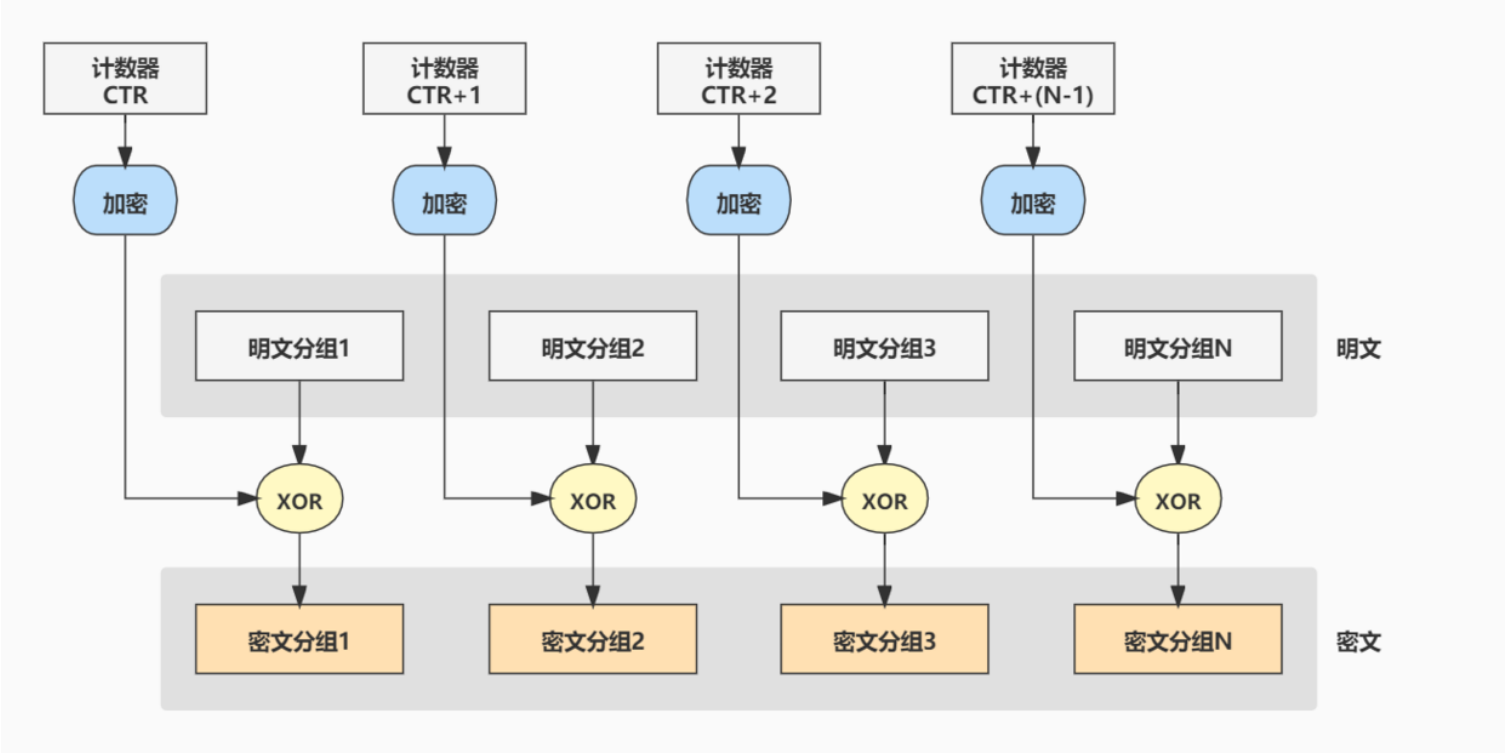
OFB模式

OFB（Output FeedBack）模式，即输出反馈模式。该模式会产生一个密钥流，即将密码算法的前一个输出值，做为当前密码算法的输入值。该输入值再与明文分组进行XOR运行，计算得出密文分组。该模式需要一个IV，进行加密后做为第一个分组的输入。



CTR模式

CTR（CounTeR）模式，即计数器模式。该模式也会产生一个密钥流，它通过递增一个计数器来产生连续的密钥流。对该计数器进行加密，再与明文分组进行XOR运算，计算得出密文分组。



分组密码的填充

在分组密码中，当数据长度不符合分组长度时，需要按一定的方式，将尾部明文分组进行填充，这种将尾部分组数据填满的方法称为**填充**（Padding)。

No Padding

即不填充，要求明文的长度，必须是加密算法分组长度的整数倍。

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD DD DD DD DD |
```

ANSI X9.23

在填充字节序列中，**最后一个字节**填充为**需要填充的字节长度**，其余字节填充**0**。

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 00 00 00 04 |
```

ISO 10126

在填充字节序列中，**最后一个字节**填充为**需要填充的字节长度**，其余字节填充**随机数**。

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 81 A6 23 04 |
```

PKCS#5和PKCS#7

在填充字节序列中，**每个字节**填充为**需要填充的字节长度**。

那如果原始数据正好就是16呢？因为 `PKCS#7` 规定Padding必须存在，因此即使原始数据是16的整数倍，也需要在末尾追加16字节的Padding，即正好追加一个块，这个块每个字节都是0x10

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 04 04 04 04 |
```

ISO/IEC 7816-4

在填充字节序列中，**第一个字节**填充固定值**80**，其余字节填充**0**。若只需填充一个字节，则直接填充**80**。

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 80 00 00 00 |

... | DD DD DD DD DD DD DD DD | DD DD DD DD DD DD DD 80 |
```

Zero Padding

在填充字节序列中，**每个字节**填充为**0**。

```
... | DD DD DD DD DD DD DD DD | DD DD DD DD 00 00 00 00 |
```

Java代码实现

Java在底层已经封装好了对称加密的实现， 我们只需要使用即可。现在介绍几个重要的类：

SecureRandom类

SecureRandom类是一个强安全的**随机数生成器**（Random Number Generator，简称：RNG），加密相关的推荐使用此随机数生成器。

我们可以通过构造方法生成一个实例，或者向构造方法传递一个种子来创建实例。

```
SecureRandom random = new SecureRandom();
```

KeyGenerator类

KeyGenerator类是对称密码的**密钥生成器**，需要指定加密算法，来生成相应的密钥。

Java中支持的算法：

- `AES` (128)
- `DES` (56)
- `DESede` (168)
- `HmacSHA1`
- `HmacSHA256`

下面是一些标准算法的介绍：

Algorithm Name	Description
AES	Advanced Encryption Standard as specified by NIST in FIPS 197. Also known as the Rijndael algorithm by Joan Daemen and Vincent Rijmen, AES is a 128-bit block cipher supporting keys of 128, 192, and 256 bits. To use the AES cipher with only one valid key size, use the format AES_<n>, where <n> can be 128, 192, or 256.
AESWrap	The AES key wrapping algorithm as described in RFC 3394. To use the AESWrap cipher with only one valid key size, use the format AESWrap_<n>, where <n> can be 128, 192, or 256.
ARCFOUR	A stream cipher believed to be fully interoperable with the RC4 cipher developed by Ron Rivest. For more information, see A Stream Cipher Encryption Algorithm "Arcfour", Internet Draft (expired).
Blowfish	The Blowfish block cipher designed by Bruce Schneier.
DES	The Digital Encryption Standard as described in FIPS PUB 46-3.
DESede	Triple DES Encryption (also known as DES-EDE, 3DES, or Triple-DES). Data is encrypted using the DES algorithm three separate times. It is first encrypted using the first subkey, then decrypted with the second subkey, and encrypted with the third subkey.
DESedeWrap	The DESede key wrapping algorithm as described in RFC 3217.
ECIES	Elliptic Curve Integrated Encryption Scheme
PBEWith<digest>And<encryption> PBEWith<prf>And<encryption>	The password-based encryption algorithm found in (PKCS5), using the specified message digest (<digest>) or pseudo-random function (<prf>) and encryption algorithm (<encryption>). Examples: • **PBEWithMD5AndDES**: The password-based encryption algorithm as defined in *RSA Laboratories, "PKCS #5: Password-Based Encryption Standard*, version 1.5, Nov 1993. Note that this algorithm implies *CBC* as the cipher mode and *PKCS5Padding* as the padding scheme and cannot be used with any other cipher modes or padding schemes. • **PBEWithHmacSHA256AndAES_128**: The password-based encryption algorithm as defined in PKCS #5: Password-Based Cryptography Specification, Version 2.1.
RC2	Variable-key-size encryption algorithms developed by Ron Rivest for RSA Data Security, Inc.
RC4	Variable-key-size encryption algorithms developed by Ron Rivest for RSA Data Security, Inc. (See note prior for ARCFOUR.)
RC5	Variable-key-size encryption algorithms developed by Ron Rivest for RSA Data Security, Inc.
RSA	The RSA encryption algorithm as defined in PKCS #1 v2.2

生成密钥代码如下：

```
/**
 * 通过密码和算法获取 Key 对象
 *
 * @param key       密钥
 * @param algorithm 算法，例如：AES (128)、DES (56)、DESede (168)、HmacSHA1、HmacSHA256
 * @return 密钥 Key
 * @throws Exception
 */
private static Key getKey(byte[] key, String algorithm) throws Exception {
    // 通过算法获取 KeyGenerator 对象
    KeyGenerator keyGenerator = KeyGenerator.getInstance(algorithm);
    // 使用密钥做为随机数，初始化 KeyGenerator 对象
    keyGenerator.init(new SecureRandom(key));
    // 生成 Key
    return keyGenerator.generateKey();
}
```

Cipher类

Cipher类提供了**加密和解密**的功能。该类需要指定一个转换（Transformation）来创建一个实例，转换的命名方式：**算法名称/工作模式/填充方式。**

下面是Java支持的转换：

- `AES/CBC/NoPadding` (128)
- `AES/CBC/PKCS5Padding` (128)
- `AES/ECB/NoPadding` (128)
- `AES/ECB/PKCS5Padding` (128)
- `DES/CBC/NoPadding` (56)
- `DES/CBC/PKCS5Padding` (56)
- `DES/ECB/NoPadding` (56)
- `DES/ECB/PKCS5Padding` (56)
- `DESede/CBC/NoPadding` (168)
- `DESede/CBC/PKCS5Padding` (168)
- `DESede/ECB/NoPadding` (168)
- `DESede/ECB/PKCS5Padding` (168)
- `RSA/ECB/PKCS1Padding` (1024, 2048)
- `RSA/ECB/OAEPWithSHA-1AndMGF1Padding` (1024, 2048)
- `RSA/ECB/OAEPWithSHA-256AndMGF1Padding` (1024, 2048)

下面是一些标准的模式：

Algorithm Name	Description
NONE	No mode.
CBC	Cipher Block Chaining Mode, as defined in FIPS PUB 81.
CCM	Counter/CBC Mode, as defined in NIST Special Publication SP 800-38C: Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality.
CFB, CFBx	Cipher Feedback Mode, as defined in FIPS PUB 81. Using modes such as CFB and OFB, block ciphers can encrypt data in units smaller than the cipher's actual block size. When requesting such a mode, you may optionally specify the number of bits to be processed at a time by appending this number to the mode name as shown in the *"DES/CFB8/NoPadding"* and *"DES/OFB32/PKCS5Padding"* transformations. If no such number is specified, a provider-specific default is used. (For example, the SunJCE provider uses a default of 64 bits for DES.) Thus, block ciphers can be turned into byte-oriented stream ciphers by using an 8-bit mode such as CFB8 or OFB8.
CTR	A simplification of OFB, Counter mode updates the input block as a counter.
CTS	Cipher Text Stealing, as described in Bruce Schneier's book *Applied Cryptography-Second Edition*, John Wiley and Sons, 1996.
ECB	Electronic Codebook Mode, as defined in FIPS PUB 81 (generally this mode should not be used for multiple blocks of data).
GCM	Galois/Counter Mode, as defined in NIST Special Publication SP 800-38D Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.
OFB, OFBx	Output Feedback Mode, as defined in FIPS PUB 81. Using modes such as CFB and OFB, block ciphers can encrypt data in units smaller than the cipher's actual block size. When requesting such a mode, you may optionally specify the number of bits to be processed at a time by appending this number to the mode name as shown in the "*DES/CFB8/NoPadding*" and "*DES/OFB32/PKCS5Padding*" transformations. If no such number is specified, a provider-specific default is used. (For example, the SunJCE provider uses a default of 64 bits for DES.) Thus, block ciphers can be turned into byte-oriented stream ciphers by using an 8-bit mode such as CFB8 or OFB8.
PCBC	Propagating Cipher Block Chaining, as defined by Kerberos V4.

下面是一些标准的填充：

Algorithm Name	Description
NoPadding	No padding.
ISO10126Padding	This padding for block ciphers is described in 5.2 Block Encryption Algorithms in the W3C's "XML Encryption Syntax and Processing" document.
OAEPPadding, OAEPWith<digest>And<mgf>Padding	Optimal Asymmetric Encryption Padding scheme defined in PKCS #1, where <digest> should be replaced by the message digest and <mgf> by the mask generation function. Examples: **OAEPWithMD5AndMGF1Padding** and **OAEPWithSHA-512AndMGF1Padding**. If `OAEPPadding` is used, `Cipher` objects are initialized with a `javax.crypto.spec.OAEPParameterSpec` object to supply values needed for OAEPPadding.
PKCS1Padding	The padding scheme described in PKCS #1 v2.2, used with the RSA algorithm.
PKCS5Padding	The padding scheme described in PKCS #5: Password-Based Cryptography Specification, version 2.1.

SSL3Padding	The padding scheme defined in the SSL Protocol Version 3.0, November 18, 1996, section 5.2.3.2 (CBC block cipher):
	
	`block-ciphered struct {`
	`  opaque content[SSLCompressed.length];`
	`  opaque MAC[CipherSpec.hash_size];`
	`  uint8 padding[`
	`  GenericBlockCipher.padding_length];`
	`  uint8 padding_length;`
	`} GenericBlockCipher;`
	
	The size of an instance of a GenericBlockCipher must be a multiple of the block cipher's block length.
	
	The padding length, which is always present, contributes to the padding, which implies that if:
	
	`sizeof(content) + sizeof(MAC) % block_length = 0,`
	
	padding has to be (block_length - 1) bytes long, because of the existence of `padding_length`.
	
	This makes the padding scheme similar (but not quite) to PKCS5Padding, where the padding length is encoded in the padding (and ranges from 1 to block_length). With the SSL scheme, the sizeof(padding) is encoded in the always present `padding_length` and therefore ranges from 0 to block_length-1.

加密代码如下：

```
private static final String DES_ALGORITHM = "DES";
private static final String DES_TRANSFORMATION = "DES/ECB/PKCS5Padding";

/**
 * DES 加密
 *
 * @param data 原始数据
 * @param key  密钥
 * @return 密文
 */
private static byte[] encryptDES(byte[] data, byte[] key) throws Exception {
    // 获取 DES Key
    Key secretKey = getKey(key, DES_ALGORITHM);

    // 通过标准转换获取 Cipher 对象，由该对象完成实际的加密操作
    Cipher cipher = Cipher.getInstance(DES_TRANSFORMATION);
    // 通过加密模式、密钥，初始化 Cipher 对象
    cipher.init(Cipher.ENCRYPT_MODE, secretKey);
    // 生成密文
    return cipher.doFinal(data);
}
```

解密代码如下：

```
private static final String DES_ALGORITHM = "DES";
private static final String DES_TRANSFORMATION = "DES/ECB/PKCS5Padding";
```

```
/**
 * DES 解密
 *
 * @param data 密文
 * @param key  密钥
 * @return 原始数据
 */
private static byte[] decryptDES(byte[] data, byte[] key) throws Exception {
    // 获取 DES Key
    Key secretKey = getKey(key, DES_ALGORITHM);
    // 通过标准转换获取 Cipher 对象，由该对象完成实际的加密操作
    Cipher cipher = Cipher.getInstance(DES_TRANSFORMATION);
    // 通过解密模式、密钥，初始化 Cipher 对象
    cipher.init(Cipher.DECRYPT_MODE, secretKey);
    // 生成原始数据
    return cipher.doFinal(data);
}
```

完整代码

完整代码请访问我的Github，若对你有帮助，欢迎给个⭐，感谢~~🌹🌹🌹

https://github.com/gozhuyinglong/blog-demos/blob/main/java-source-analysis/src/main/java/io/github/gozhuyinglong/utils/SymmetricKeyUtil.java