链滴

Android 数据加密和编码总结

作者：woyehua
原文链接：https://ld246.com/article/1544090154024
来源网站：链滴

最近正好在项目中用到数据加密，于是从网上查阅一些资料，了解各种加密方式并写代码验证，就在篇文章中做个总结吧。

我将从这几个方面介绍 Android 中的加密方式以及相关的概念：

1. 异或加密

2. MD5 算法

3. Base64 编码

4. DES 加密

5. AES 加密

6. RSA 加密

从严格意义上来说，MD5 和 Base64 不属于加密，它们分别是信息摘要算法和编码方式，但是网上多人都说 MD5 加密、Base64 加密，我觉得有必要纠正一下。对于其他的几种加密方式，下面我会一一进行举例说明。

1. 异或加密

异或运算（xor）有个特点：数 a 两次异或同一个数 b 仍然为 a，即(a^b)^b=a。利用这个原理可以现数据的加密和解密功能。

举个栗子：a=10100001，b=00000110

```
a=a^b;        // a=10100111
b=b^a;        // b=10100001，此时 b 等于 a
```

异或运算直接对二进制数据进行操作，对每一位（bit）上的数据进行变换。所以输入和输出的数据长相同，不占用额外的空间，可以用于字符和文件的加密，效率比较高。下面我们用代码实践一下：

```
    // 加密的密钥，构造一定长度的字节数组
    private final static byte[] KEY_BYTES = "Vp6flFpGW86g7hi6MhD3Zl2eThJTjPnljXE4".getBytes
);
    private final static int KEY_LENGTH = KEY_BYTES.length;

    /**
     * 异或运算加密
     *
     * @param input 要加密的内容
     * @return 加密后的数据
     */
    public static byte[] xorEncode(byte[] input) {
        int keyIndex = 0;
        int length = input.length;
        for (int i = 0; i < length; i++) {
            input[i] = (byte) (input[i] ^ KEY_BYTES[(keyIndex++ % KEY_LENGTH)]);
        }
        return input;
    }

    /**
     * 异或运算解密
     *
     * @param input 要解密的内容
     * @return 解密后的数据
     */
    public static byte[] xorDecode(byte[] input) {
        int keyIndex = 0;
        int length = input.length;
        for (int i = 0; i < length; i++) {
            input[i] = (byte) (input[i] ^ KEY_BYTES[(keyIndex++ % KEY_LENGTH)]);
        }
        return input;
    }
```

为了方便查看加密后的内容，这里对输出结果做了一下 Base64 编码。

输入：123456abcdef，输出：Z0IFUl1aJzooFCIx

2. MD5 编码

MD5 是将任意长度的数据字符串转化成短小的固定长度的值的单向操作，任意两个字符串不应有相的散列值。因此 MD5 经常用于校验字符串或者文件，因为如果文件的 MD5 不一样，说明文件内容是不一样的，如果发现下载的文件和给定的 MD5 值不一样，就要慎重使用。

MD5 主要用做数据一致性验证、数字签名和安全访问认证，而不是用作加密。比如说用户在某个网注册账户时，输入的密码一般经过 MD5 编码，更安全的做法还会加一层盐（salt），这样密码就具不可逆性。然后把编码后的密码存入数据库，下次登录的时候把密码 MD5 编码，然后和数据库中的对比，这样就提升了用户账户的安全性。

使用 Java 实现简单的 MD5 编码：

```
    /**
```

```
    * 计算字符串的 MD5
    *
    * @param text 原文
    * @return 密文
    */
   public static String md5Encode(String text) {
       try {
           MessageDigest md = MessageDigest.getInstance("MD5");
           byte[] digest = md.digest(text.getBytes());
           StringBuilder sb = new StringBuilder();
           for (byte b : digest) {
               String hexString = Integer.toHexString(b & 0xFF);
               if (hexString.length() == 1) {
                   hexString = "0" + hexString;
               }
               sb.append(hexString);
           }
           return sb.toString();
       } catch (NoSuchAlgorithmException e) {
           logger.error(e);
       }
       return null;
   }
```

输入：123456abcdef，输出：6f3b8ded65bd7a4db11625ac84e579bb

3. Base64 编码

Base64 编码是我们程序开发中经常使用到的编码方法，它用 64 个可打印字符来表示二进制数据。这 4 个字符是：小写字母 a-z、大写字母 A-Z、数字 0-9、符号"+"、"/"（再加上作为垫字的"="，实际是 65 个字符），其他所有符号都转换成这个字符集中的字符。Base64 编码通常用作存储、传输一二进制数据编码方法，所以说它本质上是一种将二进制数据转成文本数据的方案。

在 Android 中使用 Base64 很简单，系统的 API 已经封装好方法，我们直接调用即可。

```
// 编码
 String encode = Base64.encodeToString(" 123456abcdef".getBytes(), Base64.DEFAULT);

// 解码
 byte[] decodeByte = Base64.decode(encode .getBytes(), Base64.DEFAULT);
String decode = new String(decodeByte);
```

输入：123456abcdef，输出：MTIzNDU2YWJjZGVm

无论是编码还是解码都会有一个参数 flags，系统 API 提供了以下几种：

- DEFAULT：表示使用默认的方法来加密。
- NO_PADDING：表示省略加密字符串最后的 "="。
- NO_WRAP：表示省略所有的换行符（设置后 CRLF 就会失去作用）。
- CRLF：表示使用 CR、LF 这一对作为一行的结尾而不是 Unix 风格的 LF。

● URL_SAFE：表示加密时不使用对 URL 和文件名有特殊意义的字符来作为加密字符，就是以 "-" 和 "_" 代替 "+" 和 "/"。

4. DES 加密

DES 是一种对称加密算法，所谓对称加密算法就是：加密和解密使用相同密钥的算法。DES 加密算出自 IBM 的研究，后来被美国政府正式采用，之后开始广泛流传。但近些年使用越来越少，因为 DES 使用 56 位密钥，以现代的计算能力，24 小时内即可被破解。

顺便说一下 3DES（Triple DES），它是 DES 向 AES 过渡的加密算法，使用 3 条 56 位的密钥对数进行三次加密。是 DES 的一个更安全的变形。它以 DES 为基本模块，通过组合分组方法设计出分组密算法。比起最初的 DES，3DES 更为安全。

使用 Java 实现 DES 加密解密，注意密码长度要是 8 的倍数。加密和解密的 Cipher 构造参数一定要同，不然会报错。

```
    /* 加密使用的 key */
    private final static byte[] KEY_BYTES = "Vp6fhlFXKpGW8k6QPRg7Q6Jb7HyAhRi6MIhJ2YtGDZl26eTthJTj5PnIjXH5EI4".getBytes();

   /**
     * DES 加密
     *
     * @param content 待加密内容
     * @param key     加密的密钥
     * @return 加密后的字节数组
     */
    public static byte[] encryptDES(byte[] content, byte[] key) {
        try {
            SecureRandom random = new SecureRandom();
            DESKeySpec desKey = new DESKeySpec(key);
            SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");
            SecretKey secretKey = keyFactory.generateSecret(desKey);
            // DES 是加密方式, EBC 是工作模式, PKCS5Padding 是填充模式
            Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey, random);
            return cipher.doFinal(content);
        } catch (Exception e) {
            logger.error(e);
        }
        return null;
    }

  /**
     * DES 解密
     *
     * @param content 待解密内容
     * @param key     解密的密钥
     * @return 解密的数据
     */
    public static byte[] decryptDES(byte[] content, byte[] key) {
        try {
            SecureRandom random = new SecureRandom();
```

```
            DESKeySpec desKey = new DESKeySpec(key);
            SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");
            SecretKey secretKey = keyFactory.generateSecret(desKey);
            Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
            cipher.init(Cipher.DECRYPT_MODE, secretKey, random);
            return cipher.doFinal(content);
        } catch (Exception e) {
            logger.error(e);
        }
        return null;
    }
```

输入：123456abcdef，输出：j1kR1+ZraO2Tg78dHueoTg==

5. AES 加密

高级加密标准（英语：Advanced Encryption Standard，缩写：AES），在密码学中又称 Rijndael 密法，是美国联邦政府采用的一种区块加密标准。这个标准用来替代原先的 DES，已经被多方分析且为全世界所使用。简单说就是 DES 的增强版，比 DES 的加密强度更高。

AES 与 DES 一样，一共有四种加密模式：电子密码本模式（ECB）、加密分组链接模式（CBC）、密反馈模式（CFB）和输出反馈模式（OFB）。关于加密模式的介绍，推荐这篇文章：高级加密标准AS的工作模式（ECB、CBC、CFB、OFB）

```
    /* 加密使用的 key */
    private static final String AES_KEY = "KUbHwTqBy6TBQ2gN";
    /* 加密使用的 IV */
    private static final String AES_IV = "pIbF6GR3XEN1PG05";

 /**
     * AES 加密
     *
     * @param content 待解密内容
     * @param key     密钥
     * @return 解密的数据
     */
    public static byte[] encryptAES(byte[] content, byte[] key) {
        try {
            SecretKeySpec secretKeySpec = new SecretKeySpec(key, "AES");
            // AES 是加密方式, CBC 是工作模式, PKCS5Padding 是填充模式
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
            // IV 是初始向量，可以增强密码的强度
            cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec, new IvParameterSpec(AES_IV.getBy
es()));
            return cipher.doFinal(content);
        } catch (Exception e) {
            logger.error(e);
        }
        return null;
    }

    /**
     * AES 解密
```

```
     *
     * @param content 待解密内容
     * @param key     密钥
     * @return 解密的数据
     */
    public static byte[] decryptAES(byte[] content, byte[] key) {
        try {
            SecretKeySpec secretKeySpec = new SecretKeySpec(key, "AES");
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
            cipher.init(Cipher.DECRYPT_MODE, secretKeySpec, new IvParameterSpec(AES_IV.getBy
es()));
            return cipher.doFinal(content);
        } catch (Exception e) {
            logger.error(e);
        }
        return null;
    }
```

输入：123456abcdef，输出：ho9cn9SvmeisfJy6Pv96oQ==

6. RSA 加密

RSA算法是一种非对称加密算法，所谓非对称就是该算法需要一对密钥，若使用其中一个加密，则需用另一个才能解密。目前它是最有影响力和最常用的公钥加密算法，能够抵抗已知的绝大多数密码攻。从提出到现今的三十多年里，经历了各种攻击的考验，逐渐为人们接受，普遍认为是目前最优秀的钥方案之一。

该算法基于一个的数论事实：将两个大质数相乘十分容易，但是想要对其乘积进行因式分解却极其困，因此可以将乘积公开作为加密密钥。由于进行的都是大数计算，RSA 最快的情况也比 DES 慢上好倍，比对应同样安全级别的对称密码算法要慢 1000 倍左右。所以 RSA 一般只用于少量数据加密，如说交换对称加密的密钥。

使用 RSA 加密主要有这么几步：生成密钥对、公开公钥、公钥加密私钥解密、私钥加密公钥解密。

```
    public static final String AES = "AES";
    public static final String ECB_PKCS1_PADDING = "RSA/ECB/PKCS1Padding";

    /**
     * 随机生成 RSA 密钥对
     *
     * @param keyLength 密钥长度, 范围: 512～2048, 一般 1024
     * @return 密钥对
     */
    public static KeyPair generateRSAKeyPair(int keyLength) {
        try {
            KeyPairGenerator kpg = KeyPairGenerator.getInstance(RSA);
            kpg.initialize(keyLength);
            return kpg.genKeyPair();
        } catch (NoSuchAlgorithmException e) {
            logger.error(e);
        }
        return null;
    }
```

```
/**
 * 公钥加密
 *
 * @param data      原文
 * @param publicKey 公钥
 * @return 加密后的数据
 */
public static byte[] encryptByPublicKey(byte[] data, byte[] publicKey) {
    X509EncodedKeySpec keySpec = new X509EncodedKeySpec(publicKey);
    try {
        KeyFactory kf = KeyFactory.getInstance(RSA);
        PublicKey keyPublic = kf.generatePublic(keySpec);
        Cipher cipher = Cipher.getInstance(ECB_PKCS1_PADDING);
        cipher.init(Cipher.ENCRYPT_MODE, keyPublic);
        return cipher.doFinal(data);
    } catch (Exception e) {
        logger.error(e);
    }
    return null;
}

/**
 * 私钥加密
 *
 * @param data       待加密数据
 * @param privateKey 密钥
 * @return 加密后的数据
 */
public static byte[] encryptByPrivateKey(byte[] data, byte[] privateKey) {
    PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(privateKey);
    try {
        KeyFactory kf = KeyFactory.getInstance(RSA);
        PrivateKey keyPrivate = kf.generatePrivate(keySpec);
        Cipher cipher = Cipher.getInstance(ECB_PKCS1_PADDING);
        cipher.init(Cipher.ENCRYPT_MODE, keyPrivate);
        return cipher.doFinal(data);
    } catch (Exception e) {
        logger.error(e);
    }
    return null;
}

/**
 * 公钥解密
 *
 * @param data      待解密数据
 * @param publicKey 密钥
 * @return 解密后的数据
 */
public static byte[] decryptByPublicKey(byte[] data, byte[] publicKey) {
    X509EncodedKeySpec keySpec = new X509EncodedKeySpec(publicKey);
    try {
        KeyFactory kf = KeyFactory.getInstance(RSA);
```

```
            PublicKey keyPublic = kf.generatePublic(keySpec);
            Cipher cipher = Cipher.getInstance(ECB_PKCS1_PADDING);
            cipher.init(Cipher.DECRYPT_MODE, keyPublic);
            return cipher.doFinal(data);
        } catch (Exception e) {
            logger.error(e);
        }
        return null;
    }

    /**
     * 私钥解密
     *
     * @param data  待解密的数据
     * @param privateKey 私钥
     * @return 解密后的数据
     */
    public static byte[] decryptByPrivateKey(byte[] data, byte[] privateKey) {
        PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(privateKey);
        try {
            KeyFactory kf = KeyFactory.getInstance(RSA);
            PrivateKey keyPrivate = kf.generatePrivate(keySpec);
            Cipher cipher = Cipher.getInstance(ECB_PKCS1_PADDING);
            cipher.init(Cipher.DECRYPT_MODE, keyPrivate);
            return cipher.doFinal(data);
        } catch (Exception e) {
            logger.error(e);
        }
        return null;
    }
```

密钥对生成后是一串灰常长特别长的数据，大家可以对其 Base64 后看看公钥和随机性如何。使用测数据123456abcdef加密过、Base64 编码后的结果如下。加密这么短的字符串竟然生成这么长的密，难怪 RSA 算法这么慢！

密文 Base64 后：X6yx1XfkVk4DZpnzcCSZr2oK+WMP7Azm4fBcGNEwWPrRdtf9isfMeKgQsI6kOF5Vb5b5IYAIqHZRE5QcDbIM/3bTWVTVg/t7enGCUSxVaIlvJ/A37syWTUXlh59DZzBMgzG4rbzGCc8CGyO03XFq8gCncr4NMZXQwkKI8Alds=

一般来说，客户端和服务端的通信过程是这样的：服务端生成 RSA 加密的密钥对，把公钥给客户端私钥偷偷保留。客户端在首次使用公钥加密数据，然后发送给服务端。服务端接收并处理后，会把对加密的密钥下发给客户端。客户端接收到对称加密的密钥，以后的通信就会使用对称加密的方式。当也可以由客户端生成对称加密的密钥，然后用公钥加密发给服务端。这样在双方交换密钥时保证了安，之后的对称加密保证了效率。

好了，关于 Android 加密和编码的方式就先介绍这么多，欢迎大家留言交流~