《白帽子讲 Web 安全》
第 11 章 加密算法与随机数
Xor 异或
异或的数学符号为“⊕”，计算机符号为“xor”。其运算法则为：a⊕b = (¬a ∧ b) ∨ (a ∧ ¬b)
如果 a、b 两个值不相同，则异或结果为 1。如果 a、b 两个值相同，异或结果为 0。
异或逻辑运算（半加运算, XOR）
异或运算通常用符号"⊕"表示，其运算规则为：
0⊕0=0 0 同 0 异或，结果为 0
0⊕1=1 0 同 1 异或，结果为 1
1⊕0=1 1 同 0 异或，结果为 1
1⊕1=0 1 同 1 异或，结果为 0
即两个逻辑变量相异，输出才为 1

与运算（&）<逻辑乘>
参加运算的两个数据，按二进制位进行“与”运算。
运算规则：0&0=0; 0&1=0; 1&0=0; 1&1=1;
即：两位同时为“1”，结果才为“1”，否则为 0
例如：3&5 即 0000 0011 & 0000 0101 = 0000 0001 因此，3&5 的值得 1。
例如：9&5 即 0000 1001 (9 的二进制补码)&00000101 (5 的二进制补码) =00000001 (1 的二进制补码)可见 9&5=1。

或运算（|）<逻辑加>
参加运算的两个对象，按二进制位进行“或”运算。
运算规则：0|0=0; 0|1=1; 1|0=1; 1|1=1;
即 ：参加运算的两个对象只要有一个为 1，其值为 1。
例如:3|5 即 0000 0011 | 0000 0101 = 0000 0111 因此，3|5 的值得 7。
例如：9|5 可写算式如下： 00001001|00000101 =00001101 (十进制为 13)可见 9|5=13

异或运算（^）
参加运算的两个数据，按二进制位进行“异或”运算。
运算规则：0^0=0; 0^1=1; 1^0=1; 1^1=0;
即：参加运算的两个对象，如果两个相应位为“异”（值不同），则该位结果为 1，否则为 0。
例如：9^5 可写成算式如下： 00001001^00000101=00001100 (十进制为 12)可见 9^5=12

php authcode 关键部分解读

实现 xor 加密过程
$result .= chr(ord($string[$i]) ^ ($box[($box[$a] + $box[$j]) % 256]));
Ord 返回 "X" 的 ASCII 值：
Chr 从不同的 ASCII 值返回字符

(1) Reused Key Attack
攻击者不需要知道秘钥，即可以还原处明文，假设有秘钥 C，明文 A，明文 B，那么 XOR 加密表示为
E(A) =A xor C
E(B)=A xor C

密文公布于众，因此很容易就可以计算 E(A) xor E(B)
E(A) xor E(B) =(A xor C) xor (B xor C)=A xor B xor C xor C=A xor B
可以得出
E(A) xor E(B) =A xor B
这意味着 4 个数据，只需要知道 3 个，就可以推导出剩下的 1 个。密钥 C 已经已经不需要了。

(2) Bit-flipping Attack
E(A) xor E(B) =A xor B
由此可以得出
A xor E(Z) xor B=E(B)
知道 A 的明文，B 的明文，A 的密文时，可以推导出 B 的密文
解决 Bit-flipping 攻击的方法是验证密文的完整性，常见的方法是增加带有 KEY 的 MAC(消息验证码 Message Authentication Code)，通过 MAC 验证密文是否被篡改

(3) WEP 破解
WEP 是一种常见的无线加密传输协议，破解 WEP 秘钥，就可以连接无线的 Access Point。
WEP 采用 RC4 算法。
WEP 破解可以采用 Aircrack 实现

(4) ECB 模式的缺陷
ECB(电码薄模式)
一个明文分组加密成一个密文分组。因为相同的明文永远被加密成相同的密文分组，所以理论上制作一个包含有明文及其对应的密文的密码本是可能的！
ECB 模式所带来的问题是：如果密码分析者有很多消息的明密文，那它就可以在不知道密钥的情况下编写密码本。在许多实际情况中，有很多消息趋于重复。计算机的产生的消息，如电子邮件，可能有固定的结构。
该模式好的一面就是用同一个密钥加密多个消息时不会危险。

(5) Padding Oracle Attack？？？
https://blog.csdn.net/qq_31481187/article/details/71773789

(6) 密钥管理
将密钥（包括密码）保存在配置文件或者数据库中

(7) 伪随机数问题
伪随机数(pseudo random number)问题。

小结
在加密算法的选择和使用上，有以下最佳实践
(1) 不要使用 ECB 模式
(2) 不要使用流密码（RC4）
(3) 使用 HMAC-SHA1 代替 MD5（甚至可以代替 SHA1）
(4) 不要使用相同的 key 做不同的事情

(5) Salts 与 IV 需要随机产生
(6) 不要自己实现加密算法，尽量使用安全专家已经实现好的库
(7) 不要依赖系统的保密性

当你不知道该如何选择时，有以下建议

(1) 使用 CBC 模式的 AES256 用于加密；
(2) 使用 HMAC-SHA512 用于完整性检查；
(3) 使用带 salt 的 SHA256 或 SHA512 用于 Hashing

From
https://blog.csdn.net/weixin_38756990/article/details/72177367

HMAC-SHA1 是从 SHA1 哈希函数构造的一种键控哈希算法，被用作 HMAC（基于哈希的消息验证代码）。 此 HMAC 进程将密钥与消息数据混合，使用哈希函数对混合结果进行哈希计算，将所得哈希值与该密钥混合，然后再次应用哈希函数。输出的哈希值长度为 160 位。在发送方和接收方共享机密密钥的前提下，HMAC 可用于确定通过不安全信道发送的消息是否已被篡改。 发送方计算原始数据的哈希值，并将原始数据和哈希值放在一个消息中同时传送。 接收方重新计算所接收消息的哈希值，并检查计算所得的 HMAC 是否与传送的 HMAC 匹配。

因为更改消息和重新生成正确的哈希值需要密钥，所以对数据或哈希值的任何更改都会导致不匹配。 因此，如果原始的哈希值与计算得出的哈希值相匹配，则消息通过身份验证。

SHA-1（安全哈希算法，也称为 SHS、安全哈希标准）是由美国政府发布的一种加密哈希算法。 它将从任意长度的字符串生成 28 位长的字符串。

```
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.codec.binary.Base64;

/**
 * HMAC_SHA1 Sign 生成器.
 *
 *  需要 apache.commons.codec 包
 *
 */
public class HMAC_SHA1 {

    private static final String HMAC_SHA1_ALGORITHM = "HmacSHA1";

    /**
     * 使用 HMAC-SHA1 签名方法对 data 进行签名
     *
     * @param data
     *                被签名的字符串
```

```
     * @param key
     *                  密钥
     * @return
                            加密后的字符串
     */
    public static String genHMAC(String data, String key) {
        byte[] result = null;
        try {
            //根据给定的字节数组构造一个密钥,第二参数指定一个密钥算法的名称
            SecretKeySpec signinKey = new SecretKeySpec(key.getBytes(),
HMAC_SHA1_ALGORITHM);
            //生成一个指定 Mac 算法 的 Mac 对象
            Mac mac = Mac.getInstance(HMAC_SHA1_ALGORITHM);
            //用给定密钥初始化 Mac 对象
            mac.init(signinKey);
            //完成 Mac 操作
            byte[] rawHmac = mac.doFinal(data.getBytes());
            result = Base64.encodeBase64(rawHmac);

        } catch (NoSuchAlgorithmException e) {
            System.err.println(e.getMessage());
        } catch (InvalidKeyException e) {
            System.err.println(e.getMessage());
        }
        if (null != result) {
            return new String(result);
        } else {
            return null;
        }
    }
    /**
     * 测试
     * @param args
     */
    public static void main(String[] args) {
        String genHMAC = genHMAC("111", "2222");
        System.out.println(genHMAC.length()); //28
        System.out.println(genHMAC);    // O5fviq3DGCB5NrHcl/JP6+xxF6s=
    }
}
```

From
https://www.cnblogs.com/gugia/p/4641325.html

在 java 项目中使用 AES256 CBC 加密

首先要注意一点，默认的 JDK 是不支持 256 位加密的，需要到 Oracle 官网下载加密增

强文件（Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 8），否则编译会报错：

java.security.InvalidKeyException: Illegal key size
解压后替换 jre/lib/security/目录下的同名文件即可。

```
public class IotServer {

	private static final byte[] key = {..};//key.length 须满足 16 的整数倍
	private static final byte[] iv = {..};//iv.length 须满足 16 的整数倍
	private static final String transform = "AES/CBC/PKCS5Padding";
	private static final String algorithm = "AES";
	private static final SecretKeySpec keySpec = new SecretKeySpec(key, algorithm);

	public static void main(String[] args) {
		Cipher cipher = Cipher.getInstance(transform);
		cipher.init(Cipher.ENCRYPT_MODE, keySpec, new IvParameterSpec(iv));
		byte[] cipherData = cipher.doFinal("待加密的明文".getBytes("UTF-8"));
		System.out.println(Arrays.toString(cipherData));
	}
}
```

key 和 iv 都可以通过更复杂的方式生成，方法很多这里不再列出，更多的使用技巧会在实际应用中发现。