|  |
| --- |
| **Github（或者Coding）账号：** [**https://github.com/tsrigo/xdu\_crypto\_exps/tree/main/exp2**](https://github.com/tsrigo/xdu_crypto_exps/tree/main/exp2) |
| **个人博客关于密码学实验的链接：** [**实验二**](https://blog.csdn.net/weixin_45574854/article/details/134388441?csdn_share_tail=%7B%22type%22%3A%22blog%22%2C%22rType%22%3A%22article%22%2C%22rId%22%3A%22134388441%22%2C%22source%22%3A%22weixin_45574854%22%7D) |
| **实验题目（中文）：**   1. **MTC3 AES密钥:在欧洲护照的机器可读区域中编码** 2. **每次字节的ECB解密(更难)** 3. **cbc比特翻转攻击** |
| **实验摘要（中文）：**  **本实验涉及三个加密算法的破解和攻击方法，分别是 AES 加密、逐字节的 ECB 解密程序，以及 CBC 位翻转攻击。这次实验提供了对解密 AES 加密、 逐字节 ECB 解密程序和 CBC 位翻转攻击的实际经验。可以从中学到基本访问 控制协议生成密钥、逐步解密未知字符串、利用 CBC 模式的漏洞进行攻击等 关键技能。说明了在实际应用中，必须谨慎选择加密算法和实施安全措施以防范类似攻击。** |
| **题目描述（清楚描述题目中文，写出自己的理解，请勿复制原题目）**  **1. 这道题目的要求是解密一段使用AES加密（模式为CBC，初始化向量IV为零，填充方式为01-00）的消息。这个消息已经被base64编码。**  **解密这个消息需要一个密钥，这个密钥是基于基本访问控制（BAC）协议生成的，称为KENC。这个密钥并不完整，它在身份证件的机器可读区域（MRZ）中。可以从中导出KENC。任务就是根据这些信息，使用参考文章中的方法，找到原始的明文消息。**  **2. 这道题目的要求是创建一个逐字节的ECB解密程序。首先，需要复制oracle函数，让它在加密前将一个特定的Base64编码字符串添加到明文中。然后，通过一系列的步骤，包括发现密码的块大小，检测函数是否使用ECB，以及制作一个恰好缺1个字节的输入块，来解密这个未知的字符串。还需要制作一个字典，包含每个可能的最后一个字节，并将一个字节短的输入的输出与字典中的一个条目匹配，以此来发现未知字符串的每一个字节。这个过程需要对每一个字节重复进行**  **3. 这道题目是关于CBC位翻转攻击的。需要编写两个函数，第一个函数接受任意输入字符串，添加特定的前缀和后缀，并对特定字符进行转义，然后将输入填充到16字节的AES块长度，并使用随机AES密钥进行加密。第二个函数则需要解密字符串，并查找特定的字符组合。如果找到，返回真；否则，返回假。题目的挑战在于，需要在不知道AES密钥的情况下，通过修改密文使得第二个函数返回真。这需要利用CBC模式的一个特性，即在密文块中的1位错误会在下一个密文块中产生相同的1位错误。** |
| **过程（包括背景，原理：必要的公式，图表；步骤，如有必要画出流程图，给出主要实现步骤代码）**  **第一题**  **1 求未知数字**  **2 求key**  **2.1求Kseed**  **通过SHA1哈希函数对护照信息进行哈希，然后取哈希值的前32位作为Kseed。**  **2.2求Ka和Kb**  **对Kseed进行哈希，取前16位作为ka，取后16位作为kb。**  **2.3对Ka和Kb奇偶校验生成key**  **对ka和kb进行奇偶校验，生成k1和k2，然后将k1和k2合并作为最终的密钥key。**  **2.4. 拿key和base64解码后的密文解密**  **运行截图**    **第二题**  **1 确定分组大小**  **当明文的大小是分组的倍数的时候，pkcs7会添加一个dummy block，其大小就是分组大小。**  **2 确定前缀长度**  **当 padding 的长度达到一定值时，我们可以发现前面的 block 将不再发生变化。**  **在发现前的第一次，满足：随机前缀的长度 + padding的长度 = 块长度**  **3 逐字节攻击**  **暴力枚举字节，会得到 256 种加密结果，和真实明文加密结果相同的，就是所求明文。**  **运行截图**    **第三题**   1. **初始化：生成一个随机的AES密钥和初始化向量（IV）。** 2. **加密：定义一个加密函数，该函数接受一个文本输入，将其添加到预定义的字符串中，并使用AES\_CBC模式进行加密。** 3. **解密：定义一个解密函数，该函数接受一个字节串，使用AES\_CBC模式进行解密，并检查解密后的字符串中是否包含";admin=true;"。** 4. **位翻转：定义一个CBC位翻转函数，该函数接受一个参数、一个密钥大小和一个可调用的加密函数。它首先确定需要填充的字节数，然后通过修改密文中的某些位来生成一个新的密文，使得解密后的明文包含";admin=true;"。**    1. **确定填充字节数：首先，函数会确定需要填充的字节数。这是通过比较加密空字符串和加密包含不同数量’A’字符的字符串的前缀长度来实现的。当前缀长度增加时，我们就找到了需要填充的字节数。**    2. **生成输入文本：然后，函数会生成一个输入文本，该文本由填充的’A’字符和一个固定的字符串（在这个例子中是"heytheremama"）组成。**    3. **生成修改后的密文：接着，函数会生成一个新的密文。这是通过将输入文本加密，并修改加密后的密文中的某些位来实现的。具体来说，函数会遍历参数字符串中的每一个字符，对于每一个字符，函数会找到对应的密文中的位，然后通过异或操作将这个位修改为参数字符串中的字符。这样，解密后的明文中就会包含参数字符串。**    4. **返回修改后的密文：最后，函数会返回修改后的密文。这个密文在解密后会包含";admin=true;"。**   **运行截图** |
| **总结（完成心得与其它，主要自己碰到的问题和解决问题的方法）**  **这次实验，为我进一步理解和破解复杂的加密算法提供了宝贵的经验。在未来的实验中，我将继续探索更多的加密算法，以提高我的破解能力。**  **我认识到CBC位翻转攻击是一种有效的攻击方式，可以成功地绕过某些安全检查。这种攻击方式暴露了加密系统的一些弱点，如果系统允许用户直接访问密文，或者没有使用合适的填充方案和认证加密模式，那么系统就可能受到CBC位翻转攻击。**  **因此，我也从这次实验中吸取到了教训，我们应该在实际的安全环境中规避这种攻击。比如使用合适的填充方案和认证加密模式，或者不允许用户直接访问密文。** |
| **参考文献（包括参考的书籍，论文，URL等，很重要）**  **[参考链接](**[**https://ocw.cs.pub.ro/courses/ic/res/tema1**](https://ocw.cs.pub.ro/courses/ic/res/tema1)**)**  **[CBC(密码块链)自愈特性解释](**[**https://stackoverflow.com/questions/26318430/explanation-of-self-healing-property-of-cbc-cipher-block-chaining**](https://stackoverflow.com/questions/26318430/explanation-of-self-healing-property-of-cbc-cipher-block-chaining)**)**  **[CBC Bitflipping Attacks | full-stack overflow (thmsdnnr.com)](<https://thmsdnnr.com/blog/cbc-bitflipping-attacks/>)**  **[Cryptopals 16 - CBC bitflipping attacks - ETOOBUSY](<https://github.polettix.it/ETOOBUSY/2022/08/23/cryptopals-16/>)**  **[Cryptopals - CBC bitflipping attacks - Braincoke | Security Blog](<https://braincoke.fr/write-up/cryptopals/cryptopals-cbc-bitflipping-attacks/>)**  **[CBC Bit-Flipping Attack Conclusion · Automne's Shadow](<https://ce-automne.github.io/2019/05/23/CBC-Bit-Flipping-Attack-Conclusion/>)**  **[Cryptopals Guided Tour - 16 - CBC bitflipping attacks - YouTube](**[**https://www.youtube.com/watch?v=Q6wopwjhyig**](https://www.youtube.com/watch?v=Q6wopwjhyig)**)** |