# 12 | 怎么利用加密端攻击?

2020-12-21 范学雷

实用密码学 进入课程 >

**讲述：范学雷**

时长 10:33 大小 9.67M

你好，我是范学雷。

上一讲，我们讨论了怎么利用解密端攻击 CBC 模式的对称密钥分组算法。那么，加密端的攻击方案是怎么进行的？有什么办法可以规避加密端的攻击？这是我们这一次要解决的问题。

虽然我们之前说，CBC 模式应该退休了，但是现实中，它还有广泛的应用。我们花了很多篇幅来讨论 CBC 模式，除了提醒你 CBC 模式的缺陷外，还希望你了解缺陷修补的办法，这样你就可以修复现有的应用了。

这一次的讨论，比起上一次，稍显烧脑。因为，这是一个相对较新的攻击技术。不过，我可以保证，如果你了解了这种攻击技术，将会有一个巨大的收获。

## 怎么利用加密端攻击？

我们先来看看是怎么利用加密端攻击的？其实，CBC 模式针对加密端的攻击，和针对解密端的攻击方案一样，最常见的攻击方式也是通过异或运算展开的。

## 攻击面应该怎么选？

要想了解针对加密端的攻击方案，我们先来回顾一下 CBC 模式的加密过程。在这个过程中，一个明文分组的加密，需要如下的输入数据：

上一次的密文分组 Ci-1；

这一次的明文分组 Pi；

加密和解密共享的密钥 K。

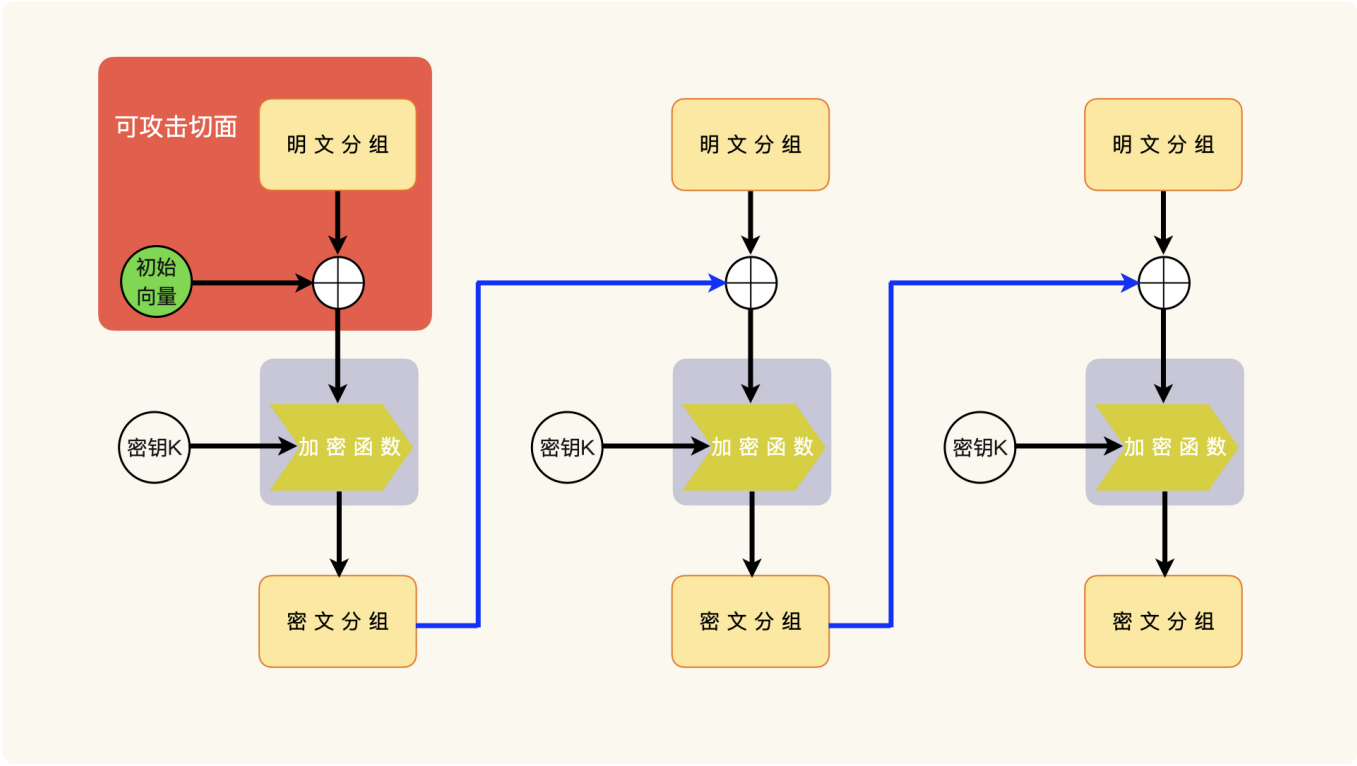
这一次的明文分组 Pi 和上一次的密文分组 Ci-1 进行异或运算，获得中间结果 Ti。

异或运算获得的中间结果 Ti，Ti = Ci-1 ^ Pi

然后，异或运算获得的中间结果 Ti 和密钥 K，通过加密函数的运算，产生密文分组：

密文分组 Ci

而加密端的异或运算，发生在明文分组和上一次的密文分组（对于第一次运算，上一次的密文分组就是初始化向量）之间。针对加密端的攻击，同样，我们也是在明文分组和上一次的密文分组或者初始化向量上想办法。




## 攻击理论上可行吗？

假设目标明文分组是 Pi，但是攻击者并不知道 Pi 的明文内容，而他攻击的目的就是解密出 Pi 的明文数据。由于密文数据不是保密的，所以攻击者可以知道所有的密文分组，包括参与 Pi 加密运算的上一次密文分组 Ci-1 和这一次的密文分组 Ci。

如果攻击者可以使用加密接口，攻击者就可以构造明文分组，然后调用加密运算，观测加密运算后的密文分组。如果攻击者观测到密文分组和目标明文分组对应的密文分组相同，那么他就可以确定目标明文分组的具体数据了。

那明文分组该怎么构造呢？像解密端的攻击方案一样，我们还是要利用异或运算的归零律和恒等律。假设已经知道了 Pi，当前的加密运算需要使用上一次的密文分组 Cj-1，如果攻击使用的明文可以构造成下面的形式，那么攻击就能够成功：

> Pj = Pi ^ Ci-1 ^ Cj-1

加密运算后的结果，就是目标明文分组对应的密文分组。

> Tj = Pj ^ Cj-1

> = Pi ^ Ci-1 ^ Cj-1 ^ Cj-1

> = Pi ^ Ci-1

> = Ti

> =>

> Ek（Tj）= Ek（Ti）

> =>

> Cj = Ci

> Tj ^ Ci-1 = Ti ^ Ci-1 = Pi ^ Ci-1 ^ Ci-1 = Pi

> Pi = Tj ^ Ci-1 = Pj ^ Cj-1 ^ Ci-1

假设终归是假设，攻击者要是知道明文分组，也就不需要实施攻击了。似乎，这个美好的运算并不能够执行。对于 AES 算法来说，一个明文分组有 16 个字节，也就是 128 位。如果一位一位地尝试，需要 2^128 次运算。这种数量级的运算，现在的计算能力还是无能为力的。

## 攻击实践上可行吗？

那么，既然完整破解有 16 个字节的明文分组不可行，能不能只破解一个字节？

怎么破解一个字节呢？我们还要继续从假设开始。假设攻击者已经知道了明文分组里的 15 个字节，只剩下最后一个字节未知。攻击者能不能破解出这个字节？这样的破解太简单了。

攻击者只需要改动最后一个字节，按照上述的构造明文分组的方式，他最多尝试 255 次，最后一个字节就可以破解了。原则上，只要有一个字节能够被破解，这个算法就算是被破解了。

不过，我们还需要验证这个假设有没有实际的可能性，也就是说，一个明文分组里，攻击者有没有可能会知道 15 个字节，只有 1 个字节是未知的？

遗憾的是，网络里的数据，大部分都是格式化的，比如 HTTP 的数据。下面的数据是我在写这些文字的时候，从 geekbang.org 登录页面记录下来的 HTTP 请求的头部数据。

这里面的数据，除了 Cookie 之外，大部分都是重复的数据。

```
POST /account/sms/login HTTP/1.1
Host: account.geekbang.org
Connection: keep-alive
Content-Length: 371
sec-ch-ua: "\\Not;A\"Brand";v="99", "Google Chrome";v="85", "Chromium";v="85"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTM
Content-Type: application/json
Origin: https://account.geekbang.org
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://account.geekbang.org/signin
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: <deleted>
```

以及 HTTP 的请求内容。请求的内容，除了电话号码（cellphone）和手机验证码之外（code），其他的数据也都是重复的。

```
{"country":86,"cellphone":"12345678901","code":"123456","ucode":"","platform":
```

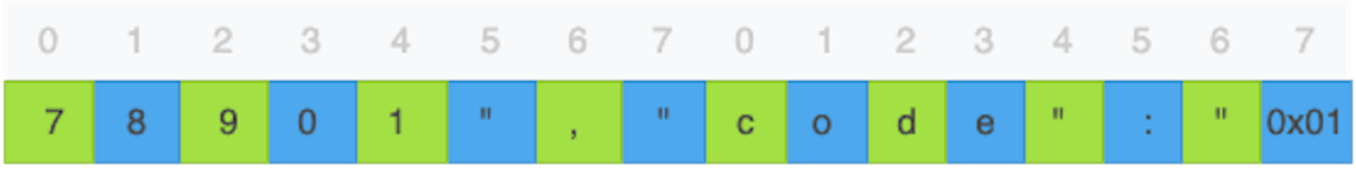
所以，攻击者知道 15 个字节，甚至整个明文分组，这种情况在实际中，是经常出现的。

我们真正关心的是敏感数据，比如手机验证码，是不是能够被破解。那么，首先，我们要了解的是，攻击者能不能破解手机验证码的第一个字节呢？

如果手机验证码的第一个字节刚好是一个明文数据分组的最后一个字节，前面我们已经讨论过了，这个字节就是可以破解的。由于手机验证码只能是数字，破解这个字节最多需要 9 次计算。比起 255 次，9 次计算就更简单了。

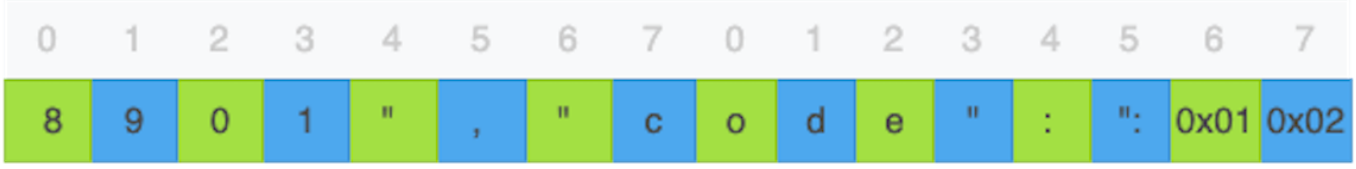

如果手机验证码的第一个字节不是明文数据分组的最后一个字节，该怎么办？思路也很简单粗暴，只要我们想办法把手机验证码的第一个字节挪到明文数据分组的最后一个字节就行。

该怎么操作呢？也很简单，通过更换顺序、删减或者增加手机验证码的前面的字节，就可以了。


既然可以想办法把手机验证码的第一个字节挪到明文数据分组的最后一个字节，当然也可以想办法把手机验证码的第二个字节、或者第三个字节，或者任意一个字节挪到明文数据分组的最后一个字节。

每当这个字节是明文数据分组的最后一个字节时，最多 9 次就可以破解这个字节了。一个 6 位数字的手机验证码，攻击者破解起来最多需要 6 * 9 = 54 次。




是的，如果加密使用的是 CBC 模式，54 次运算，6 位数的手机验证码就可以破解出来了，

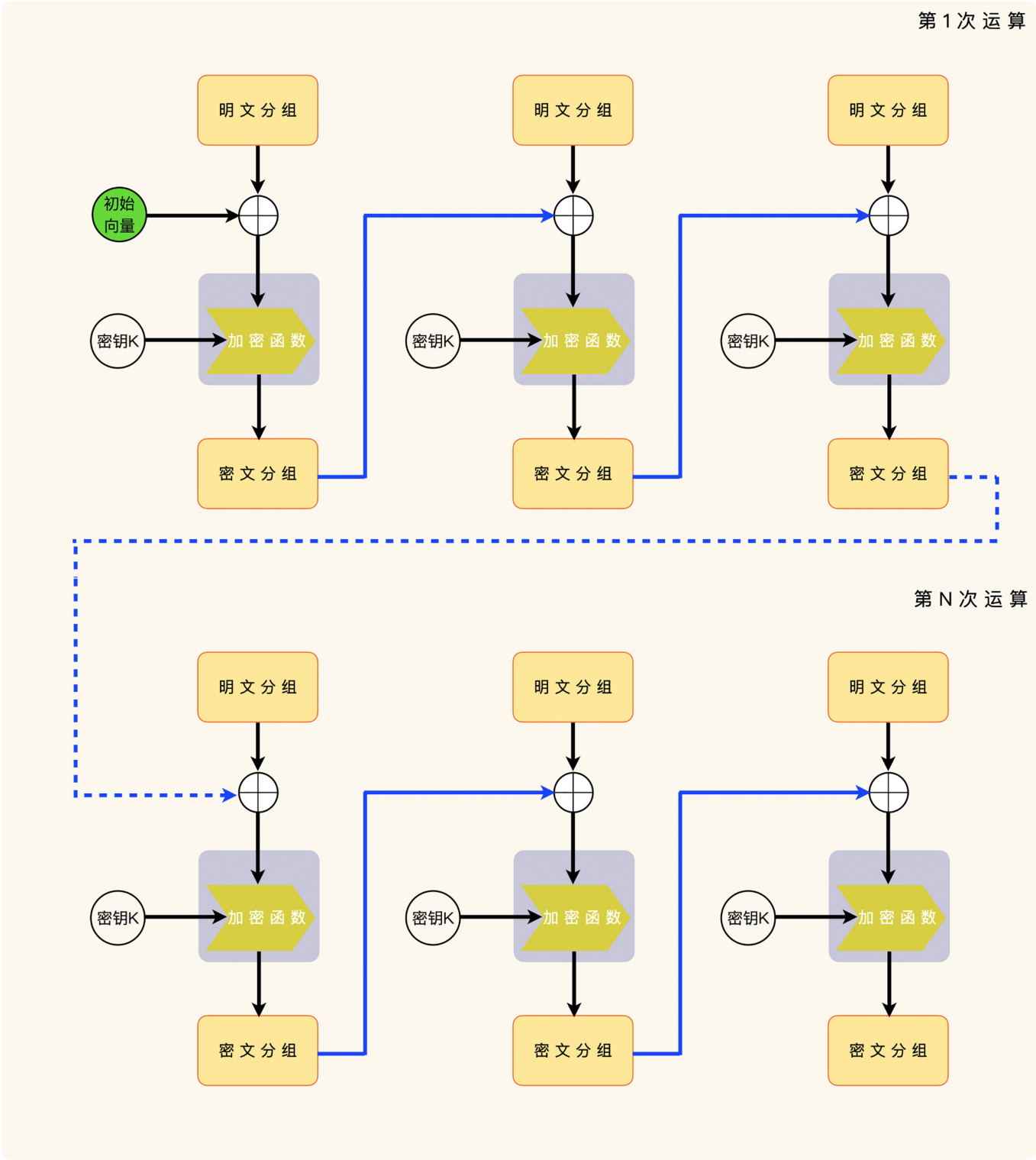
2011 年 9 月，两位天才般的研究人员（Juliano Rizzo，Thai Duong）公开了上述的攻击方案。并且表示，只要给他们几分钟时间，他们就可以利用该漏洞入侵你的支付账户。他们给这个攻击技术取了一个超酷的名字，叫做 BEAST。

2011 年 9 月之前的岁月，CBC 模式一直是加密算法的主流模式，几乎所有的网络加密都选用 CBC 模式。2011 年以后，CBC 模式就开始正式办理退休手续了。新的协议或者应用，不应该继续使用 CBC 模式了；现有的协议或者应用，也需要采取措施防范 BEAST 攻击。

## 有什么防范措施？

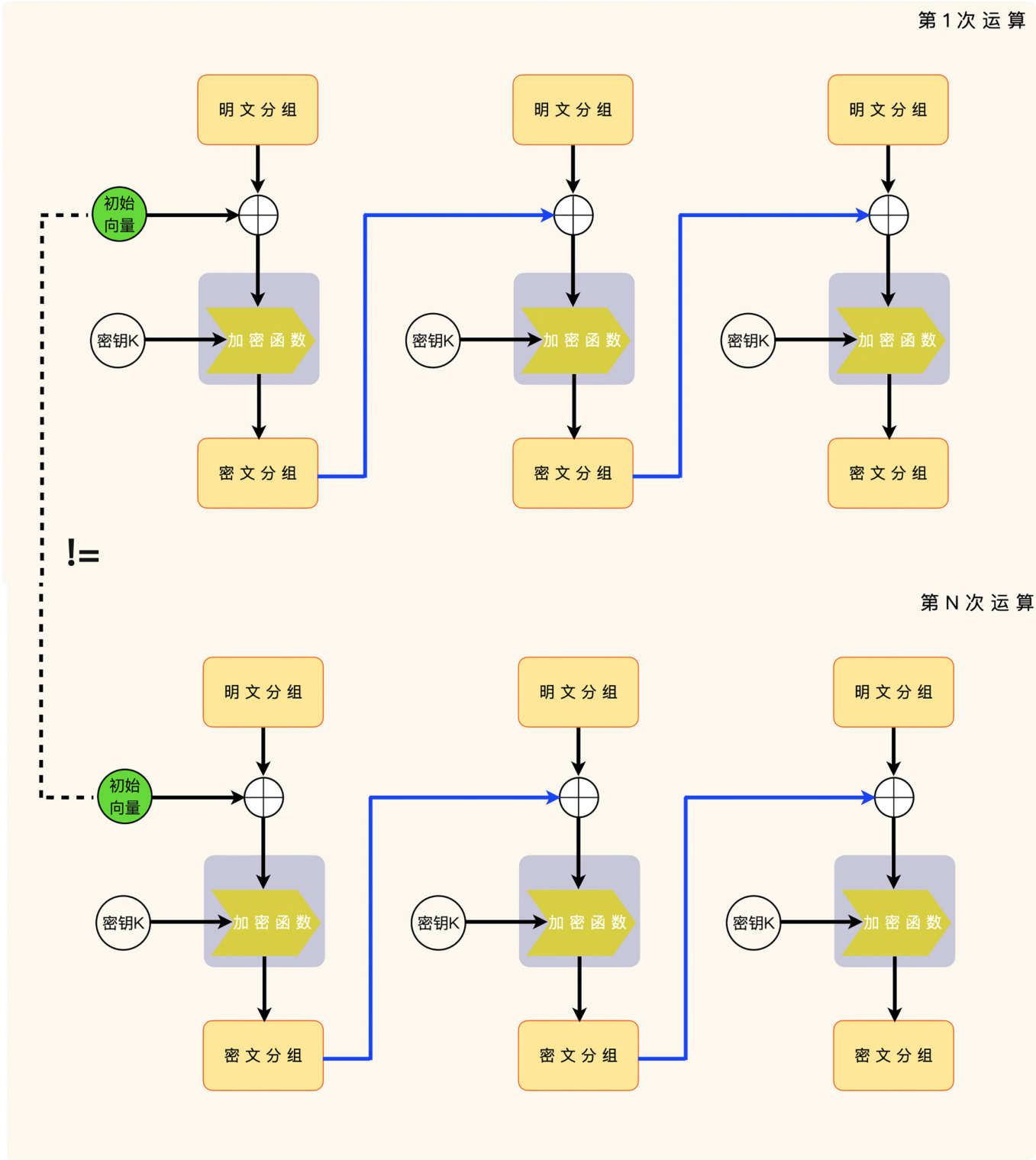
那现有的协议或者应用，要怎样采取措施来防范攻击呢？

其实，防范措施还是要在初始化向量上想办法。BEAST 攻击起作用的关键，就是要使用上一次加密运算的最后一个密文分组。




如果这一次的加密运算不使用上一次加密运算的数据，BEAST 攻击就无法运算了。该用什么替换上一次加密运算的最后一个密文分组呢？你可以先思考一下。

和防范补齐预言攻击的办法一样，替换的办法就是每一次加密运算，都使用不同的初始化向量。

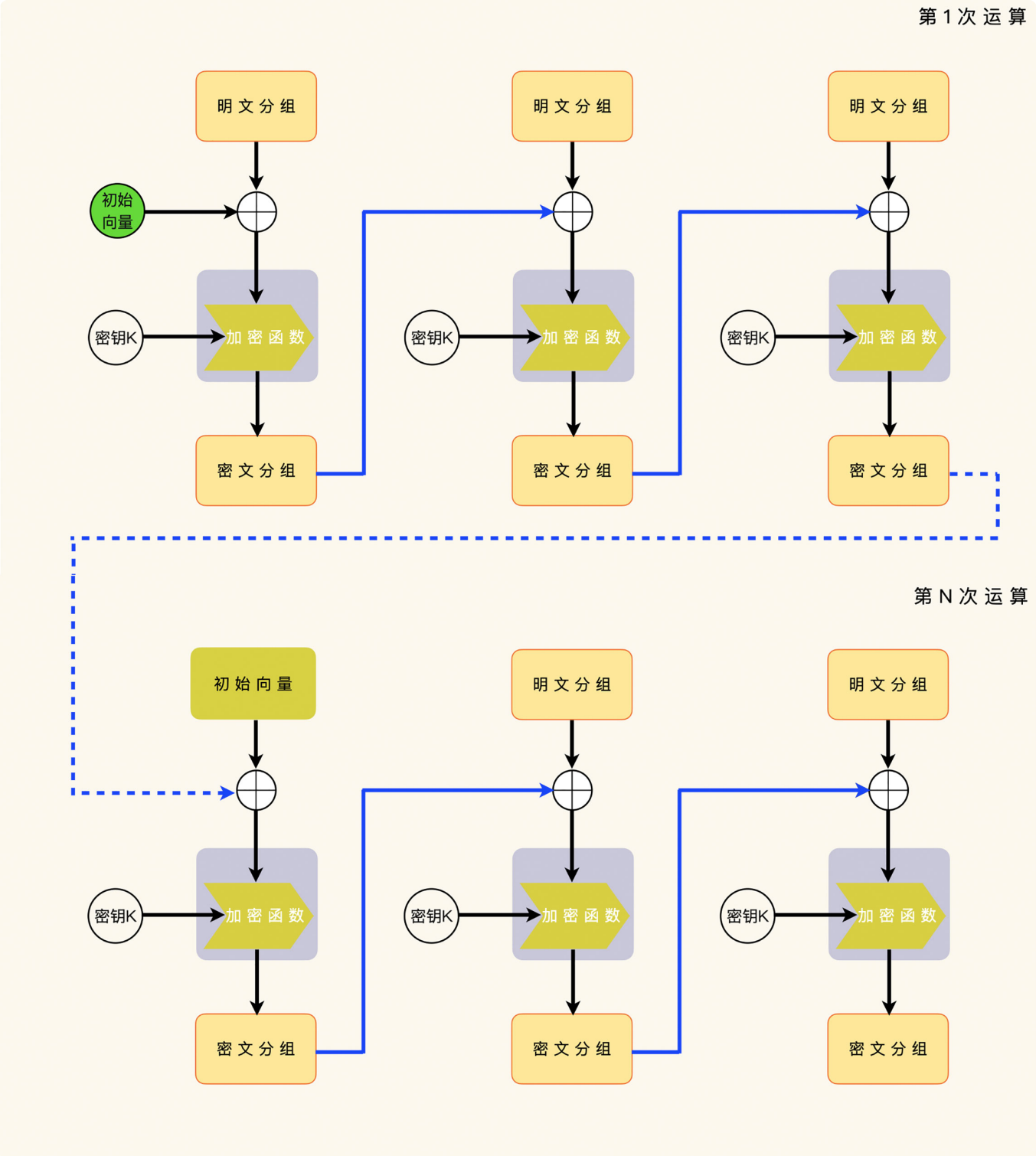



比较麻烦的是，对于每一次解密运算，解密端需要加密端使用的初始化向量，然后才能执行解密运算。可是，初始化向量的同步不是一件容易的事情，特别是在数据包可能被有意无意丢弃的环境下。

## 有没有改进的防范措施？

一个改进的办法，就是继续使用上一次加密运算的最后一个密文分组，同时把每一次运算的初始化向量当做第一个明文分组来处理。

这个办法的好处，就是解密端不需要知道加密端选择的初始化向量，就可以执行解密运算。这个办法的坏处，就是解密端需要丢掉初始化向量这一段数据，不能把它当做应用数据来处理。




如果新的应用不应该再使用 CBC 模式，那么现在推荐使用的加密模式是什么呢？在讨论这个新模式之前，我们还需要补充一块知识，那就是消息验证。下一次，我们讨论消息验证的机制；然后我们就有足够的知识去了解这个新的加密模式了。

## Take Away（今日收获）

今天，我们讨论了怎么在 CBC 模式的加密端，通过挪移，重构明文分组，来展开攻击；以及针对攻击的防范措施。

除了知道加密端攻击手段和防范措施之外，我们还要了解这种挪移，重构明文分组的攻击办法。这种攻击办法自 BEAST 攻击发布以来，已经被成功运用于很多密码学算法的破解。如果你需要设计应用协议，要小心应对明文数据的挪移，以及重构会不会构成威胁的问题。

今天，我们应该：

知道 CBC 模式存在 BEAST 攻击；

知道使用初始化向量来阻断 BEAST 攻击。

## 思考题

今天的思考题，稍微有点和以前不同。这一次，我们去挑战一下怎么给标准协议打补丁。

标准协议一旦成为标准，也就意味着它的行为模式有了规范，不能随意更改。我们假设，一个标准协议已经使用了 CBC 模式加密它的通信数据，而且只有第一次加密运算使用了初始化向量。

后续的加密运算，都不再使用初始化向量。也就是说，这个协议存在安全漏洞，是 BEAST 攻击的目标。毫无疑问，由于是现成的标准，我们不能够修改这个协议，让每一次的加密运算都使用一个不同的初始化向量。

那么，还有没有办法修复这个协议的漏洞，使得修复后的协议实现，还能够和旧的协议实现通信，而没有明显的兼容性和互操作性问题？

这个问题稍微有点难度。在 BEAST 攻击发现之前，业界已经知道 CBC 加密端攻击存在理论上的可能性。BEAST 攻击的出现，简化了攻击方案，使得 CBC 加密端的攻击成为现实。

业界尝试了很多年，都没有找到理想的办法；BEAST 攻击发现之后，也经过很长一段时间，才找到了解决的办法。最后的解决方案，简简单单又出乎意料。

你来试一试，能不能找到这个简单的方案，或者有没有更好的解决方案？

欢迎在留言区留言，记录、讨论你的发现。

好的，今天就这样，我们下次再聊。

提建议



上一篇 11 | 怎么利用解密端攻击？

## 精选留言 (3)

写留言

**可怜大灰狼**
2020-12-21

一开始的公式推导费劲看了两遍才看懂，其中Pj是自己构建的本次明文分组，Cj-1是自己构建的上次密文分组。
关于思考题，我想破解者拥有完整明文加密后的完整密文，我们是不是可以在加密前先对明文做拆分，然后对拆分的每段填充点字节。

展开

1

**25ma**
2020-12-22

感觉已经跟不上了，需要去把基础知识补一补，老师有推荐的资料吗？

**Litt1eQ**
2020-12-21

感觉文章的难度上来了 希望老师可以补充一些相关的文献 之前我是没了解过存在BEAST这种攻击方式的 通过本文的学习使我认识到了密码攻击者会利用尽可能多的方式来破坏密码的安全性