



下载APP



## 05 | 如何有效避免长度延展攻击？

2020-12-02 范学雷

实用密码学

[进入课程 >](#)**讲述：范学雷**

时长 11:37 大小 10.65M



你好，我是范学雷。

上一讲，我们列举了常见的单向散列函数，我们还知道了退役的、遗留的和现行的算法，通过对处理能力限制和算法的性能的讨论，我们对如何选择哈希算法有了更明确的认知。

还记得我们留了一个小尾巴吗？我们提到了“长度延展攻击”。“长度延展攻击”是怎么回事？我们为什么要了解它？在单向散列函数的使用上，我们需要注意哪些安全问题？

这就是我们这一次要解决的事情。



### 什么是长度延展攻击？

我们先来看看什么是“长度延展”，这样会有利于你理解“长度延展攻击”。

现在，假设我们有两段数据，S 和 M，以及一个单向散列函数 h。如果我们要把这两段数据合并起来，并且还要计算合并后的散列值，这就叫做单向散列函数的长度延展。

不过，问题来了，是 S 放在前面 ( $h(S|M)$ )，还是 M 放在前面 ( $h(M|S)$ )？既然，我们说，散列值是无法预测的，那么，数据编排的顺序有意义吗？


如果 S 和 M 都是公开的信息，顺序是不重要的。可如果 S 是机密信息，M 是公开信息，这两段数据的排列顺序就至关重要了。**如果机密信息放在了前面，就存在“长度延展攻击”的风险。**

弄清楚了长度延展，长度延展攻击就很好理解了，就是说我们可以利用已知数据的散列值，计算原数据外加一段延展数据后的散列值。也就是说，如果我们知道了  $h(S|M)$ ，我们就可以计算  $h(S|M|N)$ 。其中，数据 N 就是原数据追加的延展数据。

如果 S 和 M 都是公开的信息，能够计算延展数据的散列值也没什么紧要的。但是，如果 S 是机密数据，它的用途一般就和机密有点关系。比如说，因为没有人知道我拥有的机密数据 S，所以，当我给定一段公开信息 M 后，只有我自己才能计算 S 和 M 的散列值。


通过验证 S 和 M 的散列值，我就知道一个给定散列值是我计算、派发出去的，还是别人伪造的。

比如下面的这段数据：

 复制代码

```
1 key_id=44fefaf051fc1c61f5e76f27e620f51d5&perms=read&hash_sig=38d39516d896f879d4
```

其中，key\_id 表示机密数据的编号，perms 表示操作权限，hash\_sig 是使用机密数据 key 对 perms 的签名。签名的计算，就是使用单向散列函数：

 复制代码

```
1 sig = h(key|perms)
```

由于使用了机密数据 key，按照设想，这段数据只能由机密数据的持有者生成，然后分发出去，供授权的人使用。机密数据的持有者接收到这样的数据后，重新计算数据签名，然后对比请求数据里的签名。如果两个签名相同，就表示这是一个自己生成的、合法的授权，就可以授予请求数据所要求的权利。

不过，这个设计就存在“长度延展攻击”的风险。攻击者并不需要知道机密数据，就可以通过一个已知的 URL，构造出一个新的合法的 URL，从而获得不同的授权。

伪造的数据看起来像下面的样子：

 复制代码

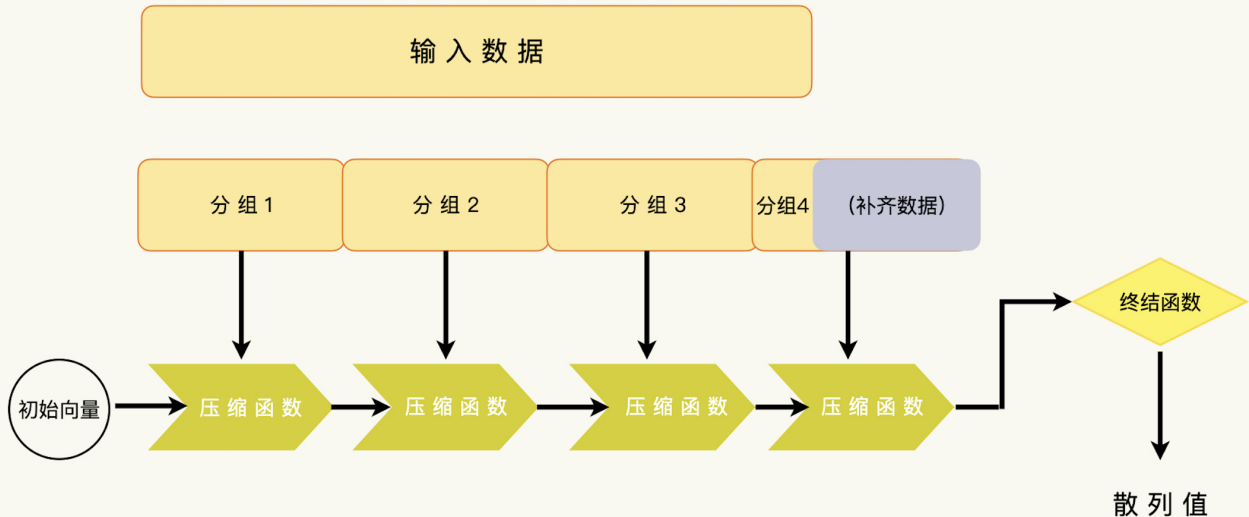
```
1 key_id=44fefaf051fc1c61f5e76f27e620f51d5&perms=read\0x80\0x00...\0x02&delete&ha
```

在这段伪造的数据中，0x80 到 0x02 之间的数据是数据块补齐数据，而且新添加了删除的权限，并且重新计算、替换了数据签名。

其中，数据签名需要使用机密数据，而攻击者并不知道机密数据，那么攻击者怎样伪造数据签名呢？要解决这个疑问，我们需要先看看单向散列函数的构造。

我们在上一讲简单地提到过，一起来重新回顾一下。一个典型的单向散列函数，应该由四个部分组成：数据分组、链接模式、压缩函数和终结函数。

我们之前着重说了数据分组，我们现在来看看其他的部分：



单向散列函数处理过程

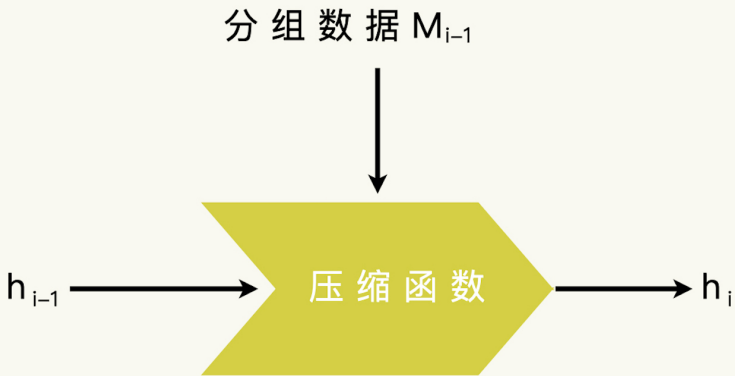
压缩函数是单向函数，负责着算法的单向性要求；

终结函数不是单向函数，负责着整理压缩函数的输出，形成散列值的任务；

链接模式，负责把下一个数据分组和上一个压缩函数的输出结果结合起来，确保算法的雪崩效应能够延续。

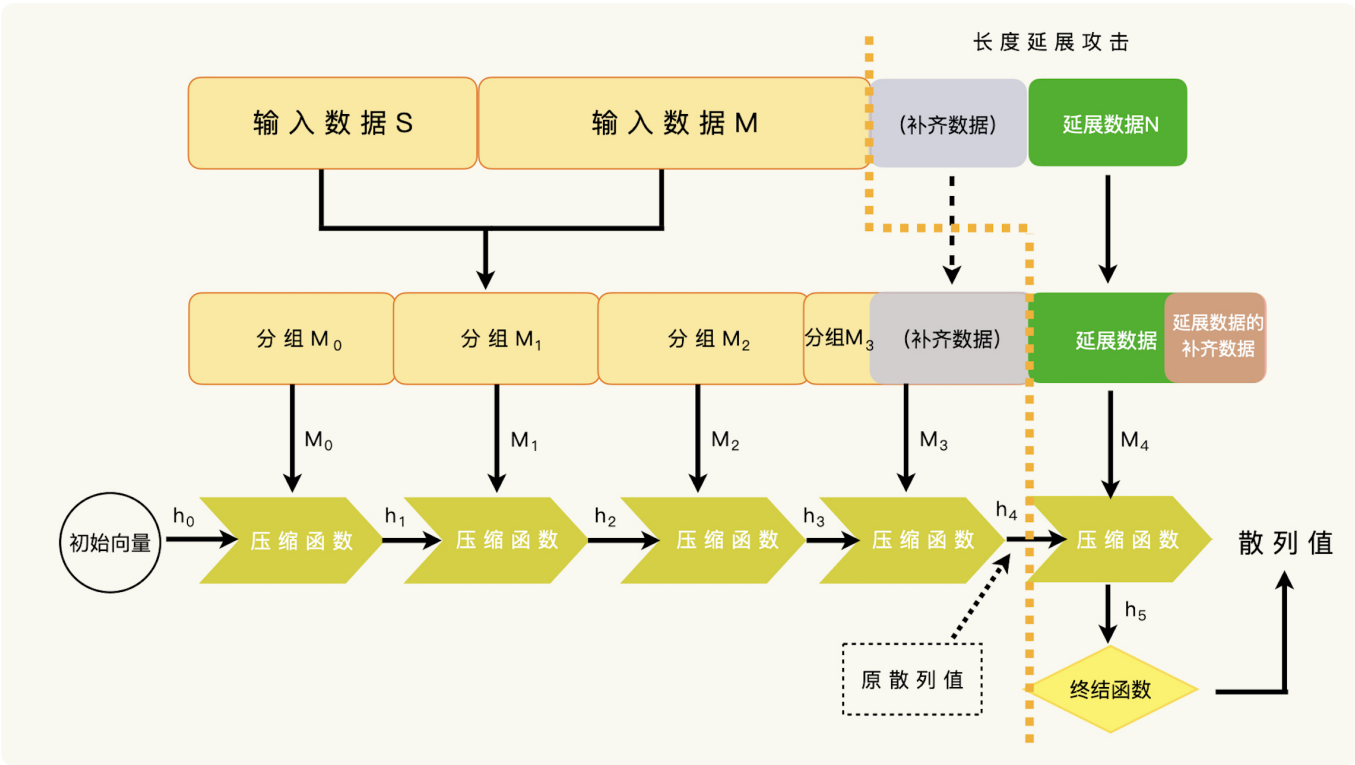
值得一提的是，在 MD5，SHA-1，SHA-256 和 SHA-512 的算法设计中，终结函数就是把压缩函数的输出向量排列成一个字节串。知道了字节串，我们也就知道了压缩函数的输出向量。

压缩函数接收一个数据分组和上一个压缩函数的运算结果。如果知道了上一个压缩函数的运算结果，我们就能够计算下一个分组数据的压缩函数运算结果。**这里，就是出现安全漏洞的地方。**



单向散列函数压缩函数示意图

我们把原来的散列值作为压缩函数的一个输入，我们再按照数据补齐规范，去补齐原来数据到数据分组的整数倍，然后加入新的数据，我们就可以计算原数据和扩展数据的散列值了。



单向散列函数长度延展攻击示意图

新的散列值的计算，不需要知道预先设想的机密数据。但是整个散列值的计算，又的确使用了机密数据。只不过，这个计算过程需要两个部分，第一部分由机密数据的持有者计算，第二部分是攻击者使用第一部分的结果，伪造了一个使用了机密数据的散列值。

但是，如果我们把数据编排顺序换一下，把公开信息 M 放在前面，机密信息 S 放在后面，长度延展攻击就不起作用了。这就是数据编排顺序对数据安全性的影响。

## 怎么有效避免长度延展攻击？

一个单向散列函数，只要使用了类似上述的压缩函数和链接模式，都是“长度延展攻击”的可疑对象。我们上一次提到的 MD2、MD5、SHA-0、SHA-1、SHA-2，都有长度延展攻击的风险。其中，对于下列算法，长度延展攻击是完全有效的：

MD2

MD5

SHA-0

SHA-1

SHA-256

SHA-512

对于下列算法，长度延展攻击虽然不是完全有效，但是算法的安全级别显著降低了：

SHA-224

SHA-384

对于下列算法，长度延展攻击没有效果（包括所有的 SHA-3 算法）：

SHA-512/224

SHA-512/256

SHA-3

上面这么长的列表，你是不是觉得好多，有点烦？其实，我们讨论长度延展攻击，目的不是让你记住上述的列表。

我们要从中学会、理解一个实用的经验：**不要单纯使用单向散列函数来处理既包含机密信息、又包含公开信息的数据**。即使我们把机密信息放在最后处理，这种使用方式也不省心。

**如果我们需要使用机密数据产生数据的签名，我们应该使用设计好的、经过验证的算法，比如我们后面会讨论的消息验证码（Message Authentication Code）和基于单向散列函数的消息验证码（Hash-based Message Authentication Code）。**

另外，如果需要设计算法，我们还要理解另外一个实用的原则：**算法要皮实、耐用，不能有意无意地用错了就有安全漏洞**。你看，SHA-1 和 SHA-2 已经很简单、皮实了，用错了场景还是有严重的问题。相比之下，SHA-3 同样简单，但是更皮实。

这和我们在 [《代码精进之路》](#) 的专栏里反复讨论的 API 要简单、直观、皮实，是一个道理。

既然我们不能单纯地使用单向散列函数处理混合了机密信息和公开信息的数据。那我们能不能单纯地使用机密信息，或者单纯地使用公开信息？回答这个问题，还要看具体的使用场景。

## 有哪些典型的适用场景？

我们已经知道了，单向散列函数是密码学的核心。下面是一些典型的使用单向散列函数的场景：

校验数据完整性；

数字签名，和非对称密钥及其算法结合使用；


消息验证码，和对称密钥及其算法结合使用；

生成伪随机数；

生成对称密钥。



还记得我们在之前，讨论过了怎么使用单向散列函数校验数据完整性。

 复制代码

```
1  输入：
2      1、数据D
3      2、原始数据的散列值H
4      3、计算散列值使用的散列函数
5  输出：
6      数据D是不是完整的？
7
8
9  运算：
10     1、使用散列函数计算数据D的散列值H'；
11     2、对比数据的散列值H和计算获得的散列值，如果两个散列值相同，则数据D是完整的；否则，数据D
```

如果我们单纯地使用单向散列函数校验数据完整性，是要对比数据的散列值的。既然是对比，也就意味着有两个散列值。这时候，我们需要考虑的主要问题就是：给定的散列值有没有被更改？

散列值的计算是公开的，给定一段数据，谁都可以计算它的散列值。如果数据可以被修改，而且给定的散列值也是修改后的数据的散列值，这个数据完整性校验是没有意义的。

所以，单纯使用单向散列函数去校验数据的完整性，我们需要确保给定的散列值是不能被修改的，这就是这个使用场景的限制。

其余的单向散列函数的使用场景，我们后面还会接着讨论。

## Take Away（今日收获）

今天，我们讨论了单向散列函数的长度延展攻击，以及使用单向散列函数需要注意的事项，还列举了典型的单向散列函数使用场景。

通过今天的讨论，我们要：

知道单向散列函数存在长度延展攻击；

了解避免长度延展攻击的办法；

尽量不要单纯使用单向散列函数来处理包含机密信息的数据。



另外，今天也是单向散列函数这一模块的最后一讲了。我们也来小结一下这一模块要注意的知识点，拉个清单。

在这一模块里，我们要掌握下面的基本概念和最佳实践：

1. **知道单向散列函数的三个特点：正向计算容易，逆向计算困难，散列值长度固定。**
2. **如果散列值不能被恶意修改，单向散列函数可以用来解决数据完整性问题。**
3. **知道有退役的算法、遗留的算法和现行的算法；并且，不要使用退役的算法，尽快升级遗留的算法。**
4. **了解密码学算法常用的三个推荐系统，美国的 NIST、德国的 BSI 和欧洲的 ECRYPT-CSA，要养成定期查看推荐指标的习惯，跟得上密码学的进展。**
5. **知道安全强度，以及现在要使用 128 位的安全强度的密码学算法，长期系统要考虑使用 256 位的密码学算法。**
6. **知道要尽量选用现行的、流行的算法。对于单向散列函数来说，它们是 SHA-256，SHA-384 和 SHA-512。**
7. **尽量不要单纯使用单向散列函数来处理包含机密信息的数据；如果不得已，要尽量避免长度延展攻击。**

## 思考题

好的，又到了留思考题的时间了。

今天的思考题是一个拓展题，你要自己去发现单向散列函数的更多适用场景。

我们一直强调，使用单向散列函数校验数据完整性，需要保证原始的散列值不能被更改。你能不能找到一些场景，可以让我们不用担心原始的散列值被更改，单纯使用单向散列函数就可以校验数据完整性？

除了我们上面列出来的一些场景，你能不能找出更多的单向散列函数使用场景？比如说，利用散列值长度固定的特点，利用碰撞困难的特点？

欢迎在留言区留言，记录、讨论你发现的新使用场景。

好的，今天就这样，我们下次再聊。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 04 | 选择哈希算法应该考虑哪些因素？

下一篇 06 | 对称密钥：如何保护私密数据？

## 精选留言 (7)

写留言



Lorin

2020-12-02

老师，我有两个问题：

- 1、长度延展攻击中，补齐数据是怎么获取到的啊？
- 2、长度延展攻击后，原散列值和扩展数据的散列值肯定不一样，这样的话，接收者都已经知道收到的数据与原散列值不一致了，这个时候长度延展攻击有什么意义呢？

展开 ∨

作者回复: 1、补齐规则是公开的，固定的数据长度，它的补齐数据是固定的。只要知道数据长度，得到补齐数据也就没有什么难度了。

2、接受者会重新计算散列值，然后对比攻击者附带的散列值，而不是保存一份散列值在本地。重新计算散列值和附带的散列值是不是相同，才是要做的对比。



1



Litt1eQ

2020-12-02

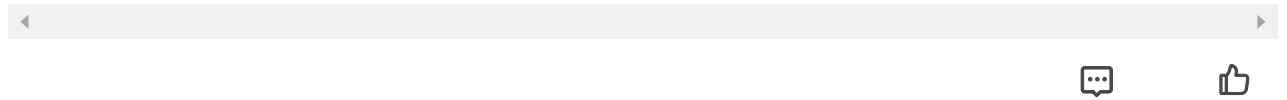
单纯使用单向散列函数就可以直接校验数据的完整性，我想到了存储在数据库中的密码，通常情况下存到数据库中的密码是不会直接存储明文的，一般都是把用户的原始密码经过散列之后保存，这样我们校验用户输入的密码经过散列之后和数据库存储的散列值进行比

较，感觉并没有考虑到数据库当中的散列值被修改的情况。

我记得在区块链中，好像是利用散列计算的困难性，找到一个某个满足要求的散列值。

展开 ∨

作者回复: 密码以散列值的情况保存，是单向散列函数一个常用的场景，还比如Unix/Linux操作系统中的影子密码。这个场景特殊的地方在于，这个散列值是本地存储、本地计算的，并不需要传递散列值。所以，攻击者没有机会既更改明文（密码），又更改散列值，除非本地被植入了恶意程序。



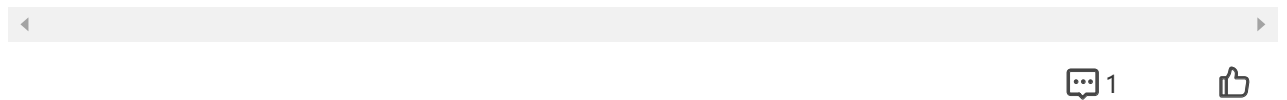
qinsi

2020-12-02

感谢解答。看来猜测数据长度是需要一个Oracle存在的（比如服务端返回请求是否合法）。下一个问题是，假设已知数据的长度，那么无论机密数据 S 在前或在后，应该都能得到补齐数据才对，为什么说放在后面的时候攻击就不起作用了呢？

展开 ∨

作者回复: 你这个理解是对的，无论S在前或者在后补齐数据都可以得到。只是，当S放在最后的时候，也就是说，S是单向散列函数输入数据的最后一段数据，这样，就不存在延展数据了。你按照专栏里延展攻击的图，画一画S放在后面的情况，就会明白了。



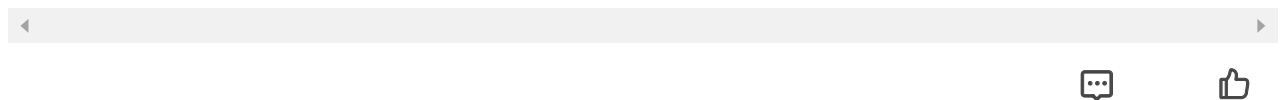
qinsi

2020-12-02

疑问：补齐数据应该是由数据长度决定的，那么在不知道机密数据 S 的长度的情况下，攻击者要如何知道该补多长的数据？跟 S 在前或在后有关吗？

展开 ∨

作者回复: 很多场景下，机密数据的长度都是固定的。另外，即使不知道长度，也可以一个字节一个字节地猜，不过是猜几十还是几百/千/万次的问题。猜中一次，以后就知道了。



天天有吃的

2020-12-02

小白追更打卡，不熟悉的感觉越来越多了，麻烦老师了..

问题1：机密数据放后面是哪个步骤的后面，是终结函数前的h5吗？

为题2: 知道了  $h(M|S)$ 也可以知道 $h(M|S|N)$ , 还是放在最后即 $h(M|N|S)$ ?

...

展开

作者回复: 都是很好的问题。你看看我有没有回答明白, 如果还有问题, 咱们继续留言。一个留言一个问题比较好, 我可以用碎片时间回复。

问题1: 机密数据不是放在那个步骤后面, 是放在输入数据的最后。如果你有两端数据, S机密, M一般, 计算 $M+S$ , 不要计算 $S+M$ 的散列值。这就是说的机密数据放后面。

问题2: 知道了  $h(M|S)$ 也可以知道 $h(M|S|N)$ , 这句话是没毛病。可是机密数据的持有者计算的时候, 是把S放在后面, 所以不会去计算 $h(M|S|N)$ . 如果数据 $A|B|C|D$ , 也是把ABCD看作M, 计算 $H(ABCD|S)$ 。

问题3: 被授权者拿这个签名, 就是到这是机密信息持有者生成的数据了。因为, 不持有私密信息, 没有办法生成这样的散列值。

问题4: 输出向量指的是压缩函数的输出结果, 用来当终结函数的输入。

问题5: 散列值能为伪造, 最轻的说明秘密信息没起作用, 就没有使用机密信息的意义了。如果是用作签名, 就是身份伪造, 就可以冒充身份诈骗了。

问题6: 相信我, 有了代码糊涂的更快。我建议你拿纸笔画一画, 画完了, 就清楚很多了。

问题7: 这个专栏应该是最基础的, 最容易看懂的了, 如果我能够表达清楚的话。你可以看看开篇词里介绍的《应用密码学》, 我认为是最值得读的就是这一本。学习密码学算法细节的话, 可能需要的数学知识就多了, 学习用的话, 初中数学加上逻辑清晰就够了。

◀

▶

💬 1

👍



孜孜

2020-12-02

突然有个脑洞, 既然散列函数在计算量足够大的情况下可逆。那么在不考虑计算量的情况下, 散列函数是不是时间上最牛的压缩算法? 或许即使可逆, 也可能逆出无数种原数据, 也无法得到真实的原数据?

作者回复: 这个有意思, 有人说单向函数的散列值是不可压缩的(也就是均匀分布的另外一个说法), 这当然是最牛的压缩。但是, 我们通常谈到压缩, 往往有对应的解压, 能从压缩过的数据复原原数据, 这个单向散列函数没有。

**sugar**

2020-12-02

老师 我有个关于密码学具体的应用场景下的问题，想听听您的看法。假设现在有一个文件，使用aes算法 进行了对称加密，想解密需要知道密钥才行。而我们这几节课探讨的加密算法的强度，我理解其实就是用程序进行枚举所有可能的密钥。但我想问，一个文件被加密之后，爆破时又是怎么知道每一次尝试时 得到的文件明文 真的是文件的真实内容呢？  
展开 ∨

作者回复: 加密算法的强度，要比枚举所有的密钥的可能性小很多。关于加密是怎么知道是不是明文的问题，这是一个好问题！后面的加密部分，我们会讲这个问题的。

