超酷算法：同型哈希

2016-10-29 汉无为 摘自 博客园 阅 20

本文由 伯乐在线 - 小泥鳅 翻译自 Nick Johnson。未经许可，禁止转载！
欢迎加入：翻译小组，通过 翻译频道 贡献一份力量。

从上一篇超酷算法文章中，我们学到看一个绝妙的概率算法 – 喷泉码。使用这个算法可以把大文件分解成几乎无穷的小块。这样，可以收集任意其中的小块，只要收集到的块的大小稍微比原始文件稍微大一点，我们就可以重构原始文件。这个是一个非常酷的构造。但是经过我们上次的观察发现，当它处于一个有不信任用户的情况下，譬如P2P网络，就会出现一个主要的问题：在对文件进行解码之前，目前没有可行的办法来对一端传递的数据块进行验证是否有效。但是文件解码也就意味着传输接近结束，这个时候再对使用不当的用户进行检测和惩罚，已经太迟了。

这里同型哈希可以帮我们。同型哈希原则上只是一个简单的数据结构：通过哈希函数，你可以根据每个块的哈希值计算出复合块的哈希值。我们可以使用像这样的数据结构，给用户分发哈希值列表，用户就可以通过这些哈希值验证传输过来的数据块，从而解决了我们的问题。

Krohn et al的一篇论文中有描述同型哈希 – 对于高效内容发布的无几率删除码即时验证。这是一个绝妙的结构，但是一开始会很难理解。所以在这篇文章里，开始会草拟一个可用的同型哈希结构，然后对它进行改善直到让它跟论文里面的结构差不多。通过这样，希望你能够更好得了解它的工作原理。我们也会讨论最终的哈希结构还存在哪些缺点和问题，同时介绍作者建议如何解决它们。

在开始之前，我需要稍微申明一下：我是一个计算机科学家，不是一个数学家，我所掌握的离散数学知识离我的目标还很远。这篇论文有的知识超出了我的理解范围，对它进行全理论描述，可能会使得我不知所云。所以这里我打算只对这些原理进行基本的解释，这些足以能够让你知道这个结构是如何工作的。有感兴趣的读者可以对剩下的内容进行更进一步探索研究。

不一样的同型哈希

我们可以用一个非常简单的数学恒等式给同型哈希构建一个简单的例子：$g^{x0} * g^{x1} = g^{x0 + x1}$ 。所以例如：$2^3 * 2^2 = 2^5$ 。 可以通过下面的步骤利用这个公式：

1， 选择一个随机数g

2， 数据里每个元素x作为底数g的指数。g^x为元素的哈希值。

通过上面的恒等式可以发现，如果把几个数据块合在一起，可以通过对单独块的哈希值相乘计算出他们的哈希值，结果应该是跟计算合并后的数据块的哈希值一样的。

不幸的是，这个结构仍然存在几点明显的问题：

哈希值不应该比元素本身还要长的多。
任何攻击者都可以通过数据块的哈希算法，计算出原始的数据块。如果我们有哈希冲突，他们一个简单的步骤也很容易的生成一个冲突。

运用取模运算的哈希

幸运的是，我们可以通过取模运算同时解决这两个问题。取模运算可以让数值范围受限，这样不仅可以解决第一个问题，而且还可以让攻击者破解更加困难。因为想要找到我们哈希值的原象，需要解决离散对数问题。这至今在数学家是一个尚未解决的问题，同时这也是多个密码系统的基础。

不幸的是，从这里开始文章变得有点复杂了。我也会描述的有点模糊。请忍耐一下。

首先，为了能够使数据块结合，我们需要选取一个模 – 称它为q。对于本实例，比如说我们想加上个0-255范围的数字，所以选取大于255最小的质数：257。

我们也需要另外一个模来进行取幂运算和乘法运算 – 称它为p。根据费马小马定理，p应该也是一个质数，并且p-1应该是q的倍数（ q | (p – 1) 或者 p % q == 1 ）。对于本实例，我们将选择1543，等于257 * 6 + 1。

我们也需要选取一个特定的值作为指数的底数，来对最后的数值进行限制 – 称它为g。简而言之，就是g^q mod p必须等于1。对于本实例，我们需要设定g为47，因为47^{257} % 1543 == 1。

所以现在我们可以像这样重新组织哈希：我们通过计算g^q mod p 得到一个数据块的哈希值，根据我们的例子公式应该是47^b mod 1543 ，b是数据块。为了合并哈希值，我们需要简单的哈希值相乘然后对p取余。为了计算合并的数据块，我们需要数据值相加然后对q进行取余。

下面我们试试吧。假设我们的数据是Hello，每个字母对应的ASCII数分别是72，101，108，109，111。我们可以计算出第一个字母的哈希值 47^{72} mod 1543，结果为883。对剩下的数字使用相同的步骤，最后得到的值分别是：883，958，81，81，313。

我们下面看看如何使用哈希值运算。所有元素的数据值总和是500，然后 500 mod 257 等于243。243的哈希运算： 47^{243} mod 1543 = 376。对数据所有哈希值运算：883 * 958 * 81 * 81 * 313 mod 1543 = 376！请随意使用其他数据进行计算，最后的结果肯定会跟预计的值一样。

实际的哈希

当然，我们改善后的哈希值仍然有一些问题：

输入值范围太小攻击者能够尝试所有值轻松地找到冲突。并且输出值的范围也太小，攻击者也能够强行得找到离散对数。
虽然哈希值长度比原来的值短了，但是还是仍然比输入值长。

第一个问题可以直接解决：可以挑选更大的质数：p和q。如果选中的值足够大，那么逐个尝试所有的输入值和强行找到离散对数都变得不切实际的。

第二个问题稍微有点棘手，但是也不是特别困难。我们必须稍微重组我们的数据。我们可以拆分数据放入长度为0到q之间的数组之中，而不是把每个元素都当成一个数据块，拆分成0到q长度的数据。例如，假设我们有个1024字节长度的数据。我们是把它拆分成64个16字节数组，而不是拆分成1024个1字节的数据块。为了做出调节，需要稍微修改下方案。我们选取16个随机数作为指数的底数，$g_0 - g_{16}$，而不只是随机选取一个数字。为了计算一个块的哈希值，我们选取每个数字g_i并把它作为底数，对应的子块为指数。输出结果的长度跟计算一个单独块的结果一样，但是我们是以16个元素作为输入而不是单独1个元素。当我们结合所有块时，对应所有的子块也会结合一起。不仅之前讨论的所有的属性依然存在着，而且又有了另外一个可调节的参数：每个块所拥有的子块数量。将会为正确权衡安全，块的粒度和协议开销方面，提供不可估量的作用。

实际应用

讲到这里，目前这个结构跟论文里面描述的差不多了。希望你能够了解它是如何应用在一个使用喷泉码算法的系统。简单挑选合适大小的质数 – 论文里建议q 为257字节，p为1024字节；并搞清楚你希望每个数据块有多大和每个数据块有多少子块。然后找到一个对每个都适合的随机数p，譬如使用一个随机数生成器，并设定一个不错的种子值。

目前的结构虽然可用，但是仍有些缺陷。首先你可能已经注意到这一点了：我们的输入值被整齐的放入字节组，在本例中数值在0到255之间。但在一个有限域里相加之后，这个域就会增加，我们就不能再把他们放入之前相同大小的字节组里面。目前有两种方案：

一个比较简洁的，一个比较散乱的。

整洁的那个是正如你所想的：既然每个值都是增加一个字节，那么就切断最前面的字节然后把它和剩下的数据块放在一起进行传递。这样做可以使得我们在传递数据块时更加合理稳健并且保持最小量得扩大数据块。但是我觉得这样有点麻烦而已显得粗俗。

散乱的方案是为q选取一个比2的给定次方大的最小质数，然后简单的忽视舍弃溢出的值。乍看下，这个好像是个比较差的解决方案，但是仔细想下：比2^{256}大的最小质数是$2^{256}+297$。

因此，我认为这理由足以让我们使用第二个方法选取一个质数，然后简单的忽视那个溢出的可能性。如果你表示还是很怀疑，你可以检查，然后扔掉任何引起溢出的编码的数据块，我敢说这量不会有很多。

性能与改善

你可能会考虑到这个方案的性能咋样。不幸的告诉你，不是很好。使用论文里面的参数作例子，对于每个子数据块是进行以257字节的数为指数以1024字节的数为指数的底数计算，即使使用先进的硬件，这样的计算也不是很快。我们是对文件每256字节的数据进行计算，所以，譬如，计算1G大小的文件，就必须进行3300万次乘方计算。这个算法能够对猜想的测试节省宽带，这个通常需要值得用消耗CPU做换取。

论文提供了两个解决方案：一个是内容生成器，一个是分发器。对于内容生成器，作者展示一个方法，通过使用密值生成随机常数g作为指数底数。使用这个密值会让内容生成器在为文件生成哈希值时更快。然而，任何人拥有这个密值都可以容易的生成哈希冲突，所以这样的话，发行人必须注意仅需要发行计算的常量g，不要把密值泄露给别人。同时，常量本身的集合也不小。就拿例子里的参数来看，一个完整的常量集合跟4个数据块的大小差不多。因此你需要一个好的方法来发布设定的常量和数据本身。谁对这个方案感兴趣的，可以查阅论文的C部分 – 预发行同型哈希（Per-Publisher Homomorphic Hashing）

对于分发器，作者提供了一个对批数据块的概率性检查，此内容在论文的D部分 – 计算效率的提高。这又是一个更容易理解的变体：累积数据块成一组，而不是单独验证每个数据块。当达到足够的数据块，将他们全部进行汇总，然后对每个单独块预期的哈希值进行乘积，计算出一个预期的哈希值。然后计算合成后块的哈希值。如果检查成功，那么所有的单独块都是有效的！如果失败，那么进行拆分测试：把批数据块对半分开并逐个检查，挑出有效的数据块，直到只剩下无效的数据。

这两个步骤都能够使得你权衡验证工作和计算机安全隐患。你可以甚至贡献一定量的CPU时间来进行验证，并简单的批量收集数据块直到当前的计算结束。同时总是需要确保你在接受下一个批数据时验证上一个批数据。

总结

当使用喷泉码系统时，同型哈希提供了一个简洁的方法来验证不信任端的数据。但是它也不是没有缺点。这个方法实现复杂，计算开销大，还需要在不影响安全性的前提下，谨慎得调节参数来最小化哈希值大小。然而，如果同型哈希能够恰当的跟喷泉码结合，那么它能够使得内容分布式网络变得无与伦比的高速与高效。

Ps: 我将 继续写更多的超酷算法相关的博客。如果你觉得哪个算法超酷，并且想更多的了解它。请在下面留言。

上一篇：一致性哈希算法原理设计

下一篇：[转] 常见的Hash算法

C -数据结构
加密概述
大型互联网技术架构3 - 分布式存储
局部敏感哈希算法
RSA算法详解及C语言实现
详解rsync算法
国家信息安全工程技术研究中心
二奇合一偶，一偶分二奇-宋世鹏

世界最伟大的25个教育法则
日军中大佐相当于什么级别?
面相十二宫详解
一日不读书,浑身不舒服!关于读书的几点建议,...
广陵琴韵
面试常问的9个问题,参考答案及解析
12个用车坏习惯 ,你有吗？
病的错误之果

发表评论