可爱的Python:用hashcash打击垃圾邮件 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/181/2021_2022__E5_8F_AF_E7_88_B1_E7_9A_84P_c103_181711.htm Hashcash 是一个拒绝服务（denial-of-service）计数器度量工具。当前它的主要作用是帮助 hashcash 用户避免因为使用了基于内容和基于黑名单的（blacklist-based）反垃圾邮件系统而丢失邮件。 可是，我认为，这项技术有着广泛的适用性，并不是只适用于电子邮件。本文还将介绍这项技术在邮件过滤方面的应用，并将提供它在其他一些方面的应用。文中将介绍我自己用 Python 完成的 hashcash 实现（它似乎是第一个当前 发布的 Python 版本），hashcash.org 站点上现在已经包含该实现。David McNab 创建了一个 Python 实现，该实现使用的协议与 hashcash 不是特别相似；其他一些开发人员也创建了部分实现 hashcash 的不完全的 Pytyhon 版本。 不过，在开始这些话题之前，让我们来回顾一下什么是 hashcash。 hashcash 基础知识 hashcash 的灵感来自于这样一个想法，即一些数学结果难于发现而易于校验。一个众所周知的例子是因数分解一个大的数字（尤其是因数较少的数字）。将一些数字相乘来获得它们的积的代价是低廉的（毕竟，CPU 周期就是金钱），但首先找到那些因数，而这项操作的代价却要高得多。 RSA 公钥密码系统就是基于这种因数分解特性的。如果应答者能够回答因数分解质询（Challenge），则说明他已经做了相当多的工作（或者偷偷地从生成那个组合的人那里得到了因数）。 对交互式质询来说，因数分解足以胜任。比如，我有一个在线资源，希望您能象征性地为其付出代价。我可以向您发送一个消息，

说“ 只要您能因数分解这个数，我将让您得到这个资源 ”。没有诚意的人将无法得到我的资源，只有那些能够证明自己有足够的兴趣、付出一些 CPU 周期来回答这个质询的人才能得到这个资源。 非交互式质询 不过，有一些资源无法很方便地进行交互式协商。 我的电子邮件收件箱是我非常重视的一个资源。但不期而至的消息占用了我的一些磁盘空间和带宽，最糟糕的是，它们吸引了我的注意力。我并不介意陌生人给我写信，但是，我希望他们能以稍微认真的态度，亲自通过对我有价值的邮件与我取得联系。至少，我不希望他们是垃圾邮件制造者，那些人向我和上百万的其他人发送包含同样消息的邮件，期望我们中的某些人能购买某种产品或陷入一个骗局。 为了实现非交互式的“ 支付（payment） ”，hashcash 让我向所有想给我发送电子邮件的人都分发一个标准质询。在您的消息头中，必须包括一个合法的 hashcash 戳记（hashcash stamp）；具体地说，该标志中包含我的收件人地址。 hashcash 提出质询的方式是，当通过安全散列算法（Secure Hash Algorithm）进行散列时，要求“ minters ”生成一个字符串（戳记，stamps），在它们的散列中有很多前导零。所找到的前导零的数目就是特定戳记的比特值。给定 SHA-1 的一致性与加密强度，找出给定比特值的 hashcash 戳记的惟一已知途径是平均 2^b 次运行 SHA-1。 然而，要确认一个戳记，只需要进行一次 SHA-1 计算即可。对于电子邮件中的应用来说，当前推荐使用的是 20-比特值：为了找到一个合法的戳记，发送者需要进行大约一百万次尝试，在最新的 CPU 和经过编译的应用程序上，这将需要不到一秒的时间。在相对旧一些的机器上它也只需要几秒钟的时间。 虽然我们

已经开始讨论 bashcash 基础知识，但在继续讨论之前，让我们先领略一下 SHA 算法的强大功能。 SHA 有多么强大？ 在一次被证明是密码界中具有重大意义的事件中，披露出一个对 SHA-0 的碰撞（collision）（请参阅参考资料中指向 Pascal Junod 的电子邮件的链接，它给出了实际碰撞的细节）。所使用的攻击需要大约 2^51 步，远远少于我们所期望的暴力构造碰撞所需要的大约 2^80 步（以及存储空间）（遵循“ 生日悖论（birthday paradox）；关于生日悖论以及如何将它应用于散列函数的更多信息的链接，请参阅参考资料）。 在过于担心这种与 bashcash 相关的攻击之前，要紧记两点：一是这种方法攻击的是 SHA-0，不是 SHA-1（目前还不是）。另一相关的保证是，在当前最快的 CPU 上，2^51 步需要的时间仍会超过 9 CPU 年。即便有类似的方法可以应用于 SHA-1，构造虚假碰撞的代价也不可能低于构造更大数量的 20-位戳记（或者甚至是 40-位 hashcash 戳记）。 hashcash（版本 1）格式 只有一个特定的 SHA-1 散列值是不够的。我们还希望戳记特定于被请求的资源 也就是说，用于 mertz@gnosis.cx 的戳记应该与用于 someuser@yahoo.com 的戳记具有不同的适用性。如果不是这样，垃圾邮件制造者就可以只生成一个高比特值的戳记并到处去使用它。 另外，一旦生成戳记，我不希望每一个想给我发送邮件的垃圾邮件制造者都能共享它。所以，hashcash 采用了以下两个额外步骤（或者至少建议它们应该作为协议的一部分）： 首先，戳记携带一个日期。用户可能会决定认为比特定期限更早的戳记是非法的。 其次，hashcash 客户机可能（并且多半应该）实现一个 double spend 数据库。 在 double spend 数据库中，每一个戳记都只能

使用一次；如果第二次收到它，那么就认为它是非法的（非常类似于邮票在使用后会被做标记）。具体地说，hashcash（版本 1）戳记类似于下面的代码：
1:bits:date:resource:ext:salt:suffix 戳记包括 7 个域。 版本号（版本 0 更简单，但是有一些局限性）。 声明的比特值。如果戳记没有真正地使用声明的前导零比特进行散列，那么它就是非法的。 生成戳记的日期（和时间）。可以认为当前时间之后的戳记以及那些在很久以前的戳记是非法的。 戳记为哪个资源而生成。可能是一个电子邮件地址，但是也可能是一个 URI 或者其他命名的资源。 特定应用程序可能需要的扩展。任何附加的数据都可以放置在这里，但是，在到目前为止的使用中，这个域通常是空的。 将该戳记与其他所有人为相同的资源在同一日期生成的戳记区别开来的随机因子（salt）。例如，两个不同的人可以合情合理地在同一天向我的同一个地址发送电子邮件。他们不应该由于我使用了 double spend 数据库而无法发送成功。但是，如果他们每个人都使用一个随机因子，那么完整戳记将是不同的。 后缀是算法真正起作用的部分。假定给出了前 6 个域，为了生成一个通过期望数目的前导零进行散列的的戳记，minter 必须尝试很多连续的后缀值。 现在让我们来看 bashcash 如何在电子邮件中起作用。 bashcash 如何在电子邮件中起作用 在理想的世界中，所有发送者都应该在他们的消息中包含 bashcash 标记；接收者在接收时都将检查它们的合法性。不过，在实际生活中，hashcash 还没有得到那么广泛的应用。虽然如此，开始使用 bashcash（不管是作为发送者还是作为接收者）并不会对现有电子邮件工具产生任何影响。换句话说，在电子邮件中

使用 bashcash，您不会有任何损失。 为了给发出的消息加上戳记，只需要向电子邮件添加头文件即可：用于电子邮件的每一个 To: 或 Cc: 接收者的 X-Hashcash 头。例如，某个想给我发送消息的人可能会在消息中包含一个与示例 rfc2822 头文件类似的头文件： X-Hashcash: 1:20:040927:mertz@gnosis.cx::odVZhQMP:7ca28 显然，应该由 MUA（邮件用户代理，mail user agents）、过滤器或者 MTA（邮件传输代理，mail transport agents）来做这件事情，而不是要求用户手工完成。不过，手工完成也不太难，至少实验时如此。首先，通过查看戳记的散列来校验它，如下所示： $ echo -n 1:20:040927:mertz@gnosis.cx::odVZhQMP:7ca28 | sha 00000b50b85a61e7ba8ac4d5fed317c737706ae5 注意前导零（每一个十六进制数是 4 个比特）。当然，还需要校验哪个资源是您识别出来的那个资源（比如您的收件人地址之一），那个戳记还没有被使用过，日期是当前日期。另外，一个合法的戳记拥有的前导零的数目应该与其声明要拥有的数目相同（不过您可以决定强制实行您自己的允许邮件通过的最小代价：20 比特是一个不完全标准（semi-standard），它最终可能会随着 Moore 定律而发生改变）。 为什么这会起作用？ 生成一个 20-比特的戳记只需要几秒钟的时间。当您一天中只发送几十封电子邮件时，这个代价并不大。但是，对那些想要发送数百万消息的垃圾邮件制造者来说，不能容忍每条消息使用额外几秒的 CPU 时间。一天之中只有 86,400 秒。即使垃圾邮件制造者利用植入木马（trojans）的僵尸（zombies）的技术，需要使用具体的 hashcash 戳记至少也会减少那些僵尸进程的发出量。当然，校验一个戳记所需的时间只是一秒的一

小部分。另一方面，向您自己的 MUA 添加 hashcash 生成和校验对其他所有人没有任何负面影响（不像其他一些反垃圾邮件方法）。对那些不使用该协议的接收者而言，这些只是一个他们很容易忽略的附加头文件。对那些没有添加 hashcash 戳记的发送者而言，检验 X-Hashcash: 的接收者不用校验任何内容。如果发送者没有添加戳记，那么您的境况不会因为进行检验而变得更糟；也不会因此变得更好。