计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(2):132−137 [doi: 10.15888/j.cnki.csa.006202]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于哈希表的 RPKI 证书验证优化方法①

安春林[1,2], 马　迪[3], 王　伟[2,3], 毛　伟[2,3]

[1](中国科学院 计算机网络信息中心, 北京 100190)
[2](中国科学院大学, 北京 100190)
[3](互联网域名系统北京市工程研究中心, 北京 100190)
通讯作者: 安春林, E-mail: 981499008@qq.com

摘　要: 在互联网码号资源公钥证书体系 (Resource Public Key Infrastructure, RPKI) 中, 依赖方 (Relying Party, RP) 负责从资料库同步并验证资源证书和签名对象 (ROAs, Manifests, Ghostbusters), 而后将有效的 ROA 处理成用于指导 BGP 路由的 IP 地址块和 AS 号的真实授权关系. 在当前的实现方式中, 验证证书模块主要通过数据库查询递归查找待验证证书的父证书从而构建完整的证书链并由 OpenSSL 完成最终验证. 由于 RPKI 体系中证书量较大, 导致基于数据库查询的方法效率不足. 结合 RPKI 运行机制中将计算代价由 BGP 路由器 (用户) 迁移到 RP 服务器 (服务器) 的特点和“空间换时间”的思想, 可以将证书信息读取到内存中从而减少 I/O 的时间消耗. 本文基于上述思想基础, 结合哈希表中条目查询的时间复杂度最优为 O(1) 的特点, 设计并实现了基于哈希表的 RPKI 证书验证优化方法. 实验结果表明, 在设计的 3 种实验场景中, 平均时间加速比分别为 99.03%、98.45% 和 97.48%, 有效的减少了时间的消耗.

关键词: 互联网码号资源公钥证书体系; 空间换时间; 哈希表; 证书验证

引用格式: 安春林,马迪,王伟,毛伟.基于哈希表的 RPKI 证书验证优化方法.计算机系统应用,2018,27(2):132–137. http://www.c-s-a.org.cn/1003-3254/6202.html

Optimization Method of RPKI Certificate Verification Based on Hash Table

AN Chun-Lin[1,2], MA Di[3], WANG Wei[2,3], MAO Wei[2,3]

[1](Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100190, China)
[3](Internet Domain Name System Beijing Engineering Research Center Ltd., Beijing 100190, China)

Abstract: In RPKI (Resource Public Key Infrastructure), RP (Relying Party) downloads and verifies certificates and signed objects (ROA, Manifest, Ghostbusters) from repository, and then processes those valid ROA objects into authorized relations between IP addresses and AS number that is used to guide the BGP routing. In the current implementation, the certificate verification module recursively finds the parent certificate of the certificate to be verified through the database query to construct the complete certificate chain and complete the final verification by OpenSSL. Because of the large number of certificates in the RPKI system, the method based on database query is inefficient. Combining the characteristic of RPKI running mechanism that transfers the calculation cost from the BGP router (user) to the RP server (server) and the idea of “space-time tradeoff”, we can read information of certificates into memory to reduce the time consumption of I/O. Based on the ideas above, combined with the characteristics of the time complexity that finding item in hash table is optimal O(1), we design and implement an optimization method of RPKI certificate validation based on hash table. The experimental results show that the average time acceleration ratio is 99.03%, 98.45%, and 97.48% in the three designed scenarios, which has effectively reduced the time consumption.

Key words: RPKI; space-time tradeoff; hash table; certificate verification

① 收稿时间: 2017-05-16; 修改时间: 2017-06-05; 采用时间: 2017-06-08

RPKI 是一种用于保障互联网基础码号资源 (包含 IP 地址, AS 号) 安全使用的公钥基础设施[1]. 通过对 X.509 公钥证书进行扩展, RPKI 依托资源证书实现了对互联网基础码号资源使用授权的认证, 并以路由源声明 (Route Origin Authorization, ROA) 的形式帮助域间路由系统, 验证某个 AS 针对特定 IP 地址前缀的路由通告是否合法[2].

RPKI 体系可以分为三个组成部件: 认证权威 (Certificate Authority, CA)、依赖方 (Relying Party, RP) 和资料库 (Repository). 分别负责签发、验证、存储各类数字对象 (包括各种签名对象和证书) 来彼此协作, 共同完成 RPKI 的功能; 其中 CA 签发的各类签名对象分为资源证书 (Resource Certificate, RC)、ROAs[3]、Manifests[4]、Ghostbusters[5]. 资料库负责收集保存所有的签名对象, 同时当有新的签名对象被创建时也要上传到资料库, 以供全球 RPs 同步下载[6]. RP 负责从资料库中同步下载资源证书和签名对象, 并验证资源证书和签名对象的有效性, 而后将有效的 ROA 处理成 IP 地址块和 AS 号的真实授权关系, 用于指导 BGP 路由.

同时, RP 端作为同步和验证 INR 分配/授权关系与实际 BGP 路由之前沟通的桥梁, RP 的运行效率影响着整个 RPKI 体系的效率. 针对 RP 同步签名对象的效率研究, 有 htsync[7]和 Delta 协议[8]. 目前 RPKI 体系中的 RP 在验证证书的过程中, 将已经处理过的证书信息保存到数据库中, 而后在验证当前证书的时候, 通过循环查询数据库来完成当前证书到信任锚点 (Trust anchor, TA) 的证书链的构建, 并将构建好的证书链交由 OpenSSL 处理来完成整个验证证书的过程[9]. 因此数据库查询的操作影响着整个验证证书过程的效率.

在常用的 PKI 身份认证中, 计算代价由终端用户负载. 如图 1, 在访问 https 网站的过程中, 首先目标网站 (https://a.com) 会将向 CA(如 CA1) 申请的证书 a.cer 发送给终端用户, 然后终端用户会通过本地保存的 CA1.cer(通常由系统内置) 来验证 a.cer 的有效性, 以此来验证目标网站的身份真实性. 在此过程中, 验证证书的计算代价由终端用户 (通常是浏览器) 负担.

而在 RPKI 运行机制中, 终端用户为 BGP 路由器, 而为了减轻 BGP 路由器的负担, 优化效率, 需要将证书验证的计算代价转移. 如图 2, RPKI 将计算代价转移到一个单独的服务器 (RP 服务器), RP 服务器负责同步资料库和验证证书, 而终端用户 (BGP 路由器) 只需要向 RP 服务器发起源验证请求并根据 RP 服务器返回的验证结果更新路由表即可, 有效的减轻了 BGP 路由器的负担.

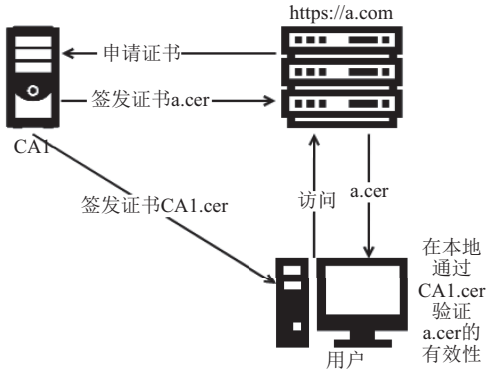

图 1 https 验证过程

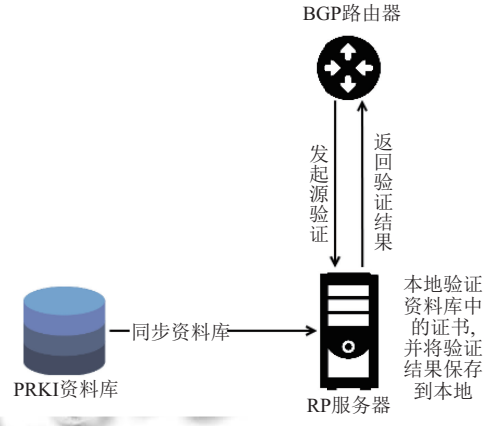

图 2 RPKI 运行机制

基于这种特点, 结合“空间换时间”的思想, 针对 RP 服务器在构建证书链是效率不足的问题, 本文设计并实现了一种基于哈希表的 RPKI 证书验证优化方法. 通过将证书的主体标识 (subject) 和签发人标识 (issuer) 当作一个键值对 (key-value) 保存到哈希表中, 来实现时间复杂度为 O(1) 的查询操作. 本文基于上述的原理, 设计实现了构建证书链的程序, 并设计实验对比基于数据库查询和基于哈希表的性能. 实验结果表明, 与基于数据库查询的方法相比, 基于哈希表的方法耗时更短, 在当前 RPKI 部署环境下, 3 种实验场景中, 平均时间加速比分别为 99.03%、98.45% 和 97.48%, 有效的减少了构建证书链的时间消耗.

1 背景

1.1 基于数据库查询的原理

现有 RP 端的实现 (Relying Party Security Technology for Internet Routing, RPSTIR) 中, 构建证书链采用的策略为数据库查询, 假设当前待验证证书 *A* 的主体标识为 *A.subject*、签发人标识为 *A.issuer*、待验证证书链 *CTX*, 具体的过程如算法 1.

算法1. 向上构建证书链

1. 查询数据库中满足条件: *B.subject*=*A.issuer*的有效证书*B*.
2. 如果*B*存在, 将*B*添加到*CTX*中, 如果*B*为TA证书, 跳转到第4步, 否则置*A*=*B*, 返回第1步.
3. 如果*B*不存在, 结束过程.
4. 调用OpenSSL验证*CTX*; 结束过程.

上述过程为向上构建证书链并验证自身的过程.

但是, 在验证证书的过程中, 证书加载的顺序并非是严格的由上而下进行, 因此在完成向上构建证书链并完成验证有效之后还需要向下构建证书链并验证. 过程与向上构建证书链类似, 假设当前验证为有效的证书为 *A*, 具体过程如算法 2.

算法2. 向下构建证书链

1. 查询数据库中满足条件: *B.issuer*=*A.subject*的所有结果集{B_1, B_2, ⋯, B_n}.
2. 如果第1步的结果集为空, 结束过程.
3. 对于每一个B_i∈{B_1, B_2, ⋯, B_n}, 循环执行第4步.
4. 对于证书B_i执行算法1.
5. 若第4步验证有效, 置*A*=B_i并递归执行第1步.
6. 若第4步验证结果为无效, 则结束递归(若有递归), 返回.

1.2 基于数据库查询方法的弊端

数据库是为了方便用户或系统对数据进行存储和管理, 但在构建证书链的过程中需要频繁的涉及到数据的查询, 并且还需要对前一次的查询结果进行递归查询. 而在这种情况下, 数据库查询效率的弊端就会更加凸显出来.

截止 2017 年 05 月, RPKI 全球部署率仅仅 7.36% (如图 3), 证书文件总量 19897 个 (.cer 文件, 包括 ROA 等签名对象中的 EE 证书).

整个 RPKI 证书体系又可以看作 5 个 (5 个 RIR 的 TA 证书为根)N 叉树, 而经过对数据库中数据的分析, 发现 66.25% 左右的证书处于整个证书树的第 3 层 (假设根为第 0 层), 32.81% 左右的证书处于证书树的第 2 层. 基于数据库查询的方法构建证书链的原理是通过循环遍历逐个向上查询父证书, 直到找到 TA 证书, 因此整个 RPKI 体系中构建证书链过程的时间复杂度可以近似为 $O(n^4)$. 如果 RPKI 在全球广泛部署 (部署率超过 80%), 则整个 RPKI 体系中证书文件量预计将会超过 26 万个, 此时构建证书链的时间将会急剧增大, 将会影响整个 RP 服务器运行的效率.

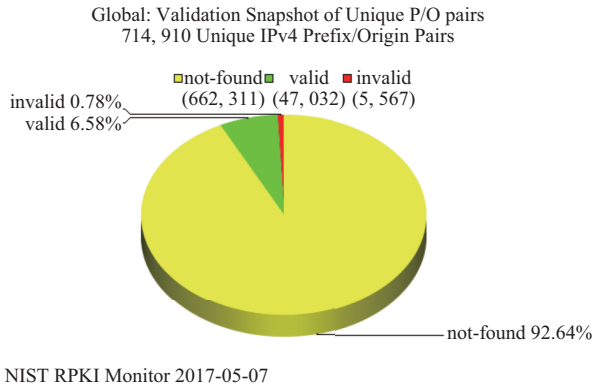

图 3 RPKI 全球部署情况

由此可以看出, 由于证书验证过程的特殊性 (需要逐层递归查询), 数据库查询的方法并不完全适合 RPKI 体系.

同时, RPKI 机制中的 RP 服务器相对于使用 PKI 的浏览器有更高的性能, 因此可以使用牺牲空间换时间的方式优化构建证书链的过程. 根据上述的原理, 本文实现了一个基于哈希表的 RPKI 证书验证优化方法.

2 基于哈希表的证书链构造方法

哈希表是一种根据“键”来直接访问在内存存储位置的数据结构. 也就是说, 它通过计算一个关于键值的函数, 将所需查询的数据映射到表中一个位置来访问记录, 这加快了查找速度. 这个映射函数称为散列函数, 存放记录的数组称作哈希表.

而在本文中, 实验是基于 Python 实现的, 使用的数据结构即为 Python 中的哈希表—字典 (Dictionary).

字典是 Python 中最常用的数据类型之一, 它是一个键值对的集合, 字典通过“键”来索引, 关联到相对的值, 理论上它的查询复杂度是 O(1). 因此, 本文实现的基于哈希表的证书链构造方法将会大大的减少耗时, 从而显著的提高 RP 的效率.

2.1 数据结构设计

在实际应用场景下, 数据结构应该包含证书的所

有信息, 并增加一个区别证书有效性的字段. 在证书验证的过程中逐层的向上查找待验证证书的“有效父证书”, 构建一个完整的证书链, 然后调用 OpenSSL 验证构建的证书链的有效性. 当验证为有效后, 递归查找所有“待验证子证书”, 并验证其有效性.

由于本文主要针对构建证书链的过程的优化, 所以为了排除调用 OpenSSL 验证证书链的时间消耗以及其他的干扰项, 特将证书验证的过程简化为逐层向上查找“父证书”而非“有效父证书”, 直到查找到 TA 或者找不到“父证书”, 然后递归查找所有“子证书”而非“待验证子证书”, 直到查找失败. 也即尽可能的向上查找父证书和向下查找子证书, 忽略证书的有效性和调用 OpenSSL 验证证书链的过程.

因此, 数据结构中只需要保存构建证书链需要的信息, 即证书的主题标识 subject 和签发人标识 issuer. 而查找父证书和子证书的方法与基于数据库查询的方法相同.

考虑到验证证书不仅需要完成向上构建证书链的过程, 还需要完成向下构建证书链, 因此本文设计实现了两个数据结构: *chain_up* 和 *chain_down*, 分别用于向上和向下构建证书链.

chain_up 用来保存向上构建证书链需要用到的信息. 向上构建证书链即为查找一条满足 *B.subject*=*A.issuer* 条件的记录 (*A* 为待验证证书), 因此 *chain_up* 中的每条记录的“键”为每个证书的主体标识 subject, “值”为对应证书的签发人标识 issuer. 且由于每个证书的主体标识具有唯一性, 因此 *chain_up* 数据结构中不存在“键”冲突的情况.

chain_down 用来保存向下构建证书链需要用到的信息. 向下构建证书链即为查找每一个满足 *B.issuer*=*A.subject* 条件的记录 (A 为待验证证书), 因此 *chain_down* 中每条记录的“键”为每个证书的签发人标识 issuer. 而由于父证书与子证书的对应关系为 1: N, 所以存在一个“键”(issuer) 对应多个“值”(subject) 的情况, 所以“值”应该为所有签发人标识 issuer 等于“键”的所有证书的主体标识 subject 的数组集合.

假设证书结构如图 4 所示.

根据前文的描述, *chain_up* 应该如下所示:

{"A": "A", "B": "A", "C": "A", "D": "B", "E": "C", "F": "C", "G": "C"}

chain_down 应该如下所示:

{"A": ["A", "B", "C"], "B": ["D"], "C": ["E", "F", "G"]}

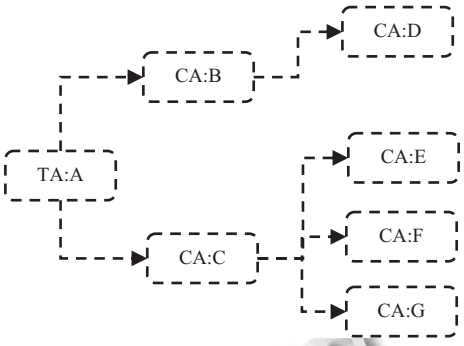

图 4 证书结构

2.2 向上构建证书链

为了排除其他实验项对结论的影响, 本文实验仅涉及证书链的构建, 并不对证书链进行有效性验证. 因此向上构建证书链的算法只进行逐层查找父证书的功能.

基于哈希表的向上构建证书链算法如算法 3.

算法3. 哈希表向上构建证书链

输入: RPKI资料库证书文件*A*, *chain_up*文件(Json格式)

1. 令变量*subject*=*A.subject*, *issuer*=*A.issuer*.
2. 若*subject*=*issuer*, 说明已经逐层向上找到根节点, 即TA证书, 跳转第6步; 若*subject*!=*issuer*, 继续第3步.
3. 令*subject*=*issuer*, *issuer*=*chain_up*[*issuer*].
4. 若第3步出现错误: *KeyError*, 说明在进行*chain_up*[*issuer*]时出错, 表明当前字典*chain_up*中没有“键”=*issuer*的数据, 跳转第6步.
5. 若第3步没有错误出现, 以在第3步中重新赋值的*subject*和*issuer*跳转第2步.
6. 向字典*chain_up*中插入新数据*chain_up*[*A.subject*]=*A.issuer*; 结束.

在程序的初始情况下, 字典 *chain_up* 应该为空, 随着程序的不断执行, 在程序将所有的 RPKI 资料库中的证书文件全部遍历完全之后, *chain_up* 应该包含每一个 RPKI 资料库证书的主体标识和签发人标识.

由于 Json 文件的特性 (键值对), 且 Python 对 Json 文件的解析结果即为 Python 中的字典, 因此在程序退出之前, 应该将最终的 *chain_up* 保存成 Json 文件, 以供下一次使用.

2.3 向下构建证书链

与向上构建证书链类似, 向下构建证书链的算法也只实现循环遍历所有子证书的功能.

基于哈希表的向下构建证书链算法如算法 4.

算法4. 哈希表向下构建证书链

输入: RPKI资料库证书文件*A*, *chain_down*文件(Json格式存储)

1. 令变量*subject*=*A.subject*, *issuer*=*A.issuer*.
2. 令临时变量*subject_arrary*=*chain_down*[*subject*], *issuer*=*subject*.
3. 若第2步出现错误: *KeyError*, 说明在进行*chain_down*[*subject*]时出错, 表示当前字典*chain_down*中没有“键”=*subject*的数据, 跳转至第6步.

4. 对于每一个$subject_i \in subject_arrary$递归执行第5步.
5. 如果$subject_i$=*issuer*, 跳出递归; 否则令*subject*=$subject_i$, 并跳转至第2步.
6. 若字典*chain_down*中存在"键"*A.issuer*, 则插入新数据: *chain_down*[*A.issuer*].append(*A.subject*); 不存在则新建键值对: *chain_down*[*A.issuer*]=[*A.subject*]. 结束.

与向上构建证书链类似, 字典 *chain_down* 在初始情况下应该为空, 随着程序的不断进行而不断的填充数据. 在程序完成所有 RPKI 资料库中证书文件的遍历之后, *chain_down* 应该包含所有位于证书树中的非叶子节点的证书的主体标识和对应的所有子证书的主体标识.

同样, 在程序退出之前应该将字典 *chain_down* 保存成 Json 文件.

总的来说, 本文实验分为两大步骤: 使用基于数据库查询的方法实现类似算法 3 和算法 4 的构建证书链方法; 使用基于哈希表的方法构建证书链.

在基于数据库查询的方法中, 首先遍历整个 RPKI 资料库, 并过滤出所有的证书文件. 然后对每一个证书文件, 使用 Python 中的 OpenSSL 库 pyOpenSSL 解析文件, 并提取出证书文件的主体标识 *A.subject* 和签发人标识 *A.issuer*. 接着查询数据库中满足条件 *B.subject*=*A.issuer* 的记录, 并递归查询, 完成向上构建证书链的过程. 然后查询数据库中满足条件 *B.issuer* =*A.subject* 的所有子证书, 并递归的对每一个子证书执行向下构建证书链的过程.

在基于哈希表的方法中, 同样先遍历整个 RPKI 资料库, 并过滤出所有证书文件, 对每一个证书完成向上和向下构建证书链的过程.

假设 RPKI 资料库中证书文件的个数为 N, 且证书树的结构与当下情况类似, 如图 5.

证书最深层数为 5, 且有 66.25% 的证书处于证书树的第 3 层 (图 5 中 TA 为 1 层). 那么, 构建证书链的时间复杂度为:

*N**(O(向上构建)+O(向下构建))

由于基于数据库查询的方法, 每一次查找父证书或查找子证书都相当于遍历一次数据库表, 因此基于数据库查询的方法复杂度就近似为 $O(N^*(N^3+N^2))$, 也即 $O(N^4)$. 而基于哈希表的方法, 对于每一个查询操作的复杂度在最优情况下 (所有的"键"的散列值均不同) 为 O(1), 最差的情况下 (所有的"键"的散列值都相同) 为 O(*N*). 因此基于哈希表的方法, 在最优情况下的复杂度为 O(*N*), 最差情况下与基于数据库查询的方法相同, 为 $O(N^4)$.

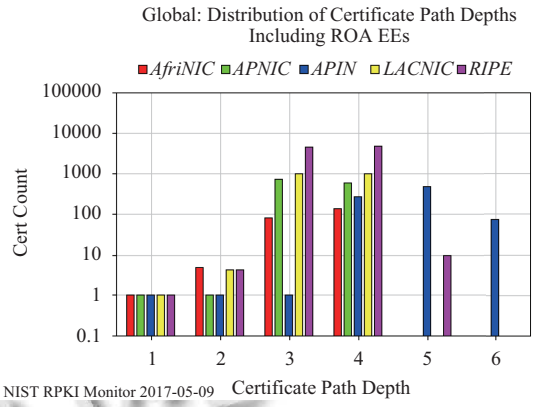

图 5 RPKI 证书结构

3 实验分析

3.1 实验设计

RP 服务器在验证 RPKI 资料库中证书文件时, 可能存在三种场景: 初始验证、增量验证和无更改验证.

初始验证: RP 服务器在收到 RPKI 资料库之后从无到有的验证每一个证书;

增量验证: RP 服务器已经完成 RPKI 资料库中的证书的验证, 此次验证时 RPKI 资料库有新的证书文件;

无更改验证: RPKI 资料库没有任何变化, 重新完成一遍验证过程.

当 RP 服务器第一次启动时, RP 服务器没有任何 RPKI 资料库文件, 因此会发生"初始验证", 此时 RP 服务器的数据库中也没有任何数据, 哈希表 *chain_up* 和 *chain_down* 也为空, 需要下载 RPKI 资料库并完成验证; 当 RPKI 资料库中有新的证书文件时, RP 服务器将会进行"增量验证", 遍历新的 RPKI 资料库, 对新的证书文件进行验证; 当 RPKI 资料库没有任何变动的时候, RP 服务器将会进行"无更改验证", 此时 RP 服务器将会遍历 RPKI 资料库, 并检查每一个证书是否已经被记录到数据库或 *chain_up* 和 *chain_down* 中.

根据以上三种情况设计实验对比基于数据库查询和基于哈希表的时间消耗. 实验环境如表 1 所示. 实验数据来自全球 RPKI 资料库, 实验数据截止到 2017 年 5 月 9 日, 共 19 897 个证书文件.

表 1 实验环境

CPU	内存	操作系统
1核Intel Ivy Bridge	4 G	Ubuntu 14.04

3.2 实验结果

3.2.1 初始验证

设定 RPKI 资料库大小分别为 2000、5000、8000、11 000、14 000、17 000、19 897 个证书文件. 7 种环境下的 RP 服务器中的已验证证书个数均为 0. 基于数据库查询 (Database) 和基于哈希表 (Hash) 的性能对比如表 2 所示.

表 2 初始验证性能对比

文件个数	数据库(s)	哈希表(s)	加速比(%)
2000	9.73	0.18	98.15
5000	41.85	0.45	98.92
8000	77.54	0.82	98.94
11 000	123.94	1.15	99.07
14 000	173.66	1.57	99.10
17 000	307.56	1.53	99.50
19 897	474.23	2.19	99.54

可以看出, 基于哈希表的方法在时间上明显更少, 且随着证书文件个数的增多, 时间加速比呈线性增大, 也即证书文件越多, 基于哈希表的方法就相对越有效. 7 种环境下的平均时间加速比为 99.03%.

3.2.2 增量验证

设定 RP 服务器已验证的证书个数分别为 2000、5000、8000、11 000、14 000、17 000 个, 在增量验证的实验中, RPKI 资料库发生更新, 有新的证书文件, 增量均为 3000 个 (17000 个文件的初始环境下的增量为 2897 个). 基于数据库查询和基于哈希表的性能对比如表 3 所示.

表 3 增量验证性能对比

初始个数	数据库(s)	哈希表(s)	加速比(%)
2000	27.48	0.43	98.44
5000	36.92	0.79	97.89
8000	50.32	0.97	98.07
11 000	71.29	1.35	98.11
14 000	149.13	1.75	98.83
17 000	305.17	1.97	99.35

可以看出, 基于哈希表的方法仍然优于基于数据库查询的方法, 6 种环境下的平均时间加速比为 98.45%.

3.2.3 无更改验证

设定 RP 服务器的资料库证书个数分别为 2000、5000、8000、11 000、14 000、17 000 和 19 897 个时, RPKI 资料库未发生变化, RP 服务器此时进行无更改验证, 仅需要对资料库中的每一个证书文件进行一次查询是否存在于数据库或 *chain_up* 和 *chain_down* 中即可. 两种方法的性能对比如表 4 所示.

表 4 无更改验证性能对比

文件个数	数据库(s)	哈希表(s)	加速比(%)
2000	2.95	0.17	94.24
5000	14.42	0.48	96.67
8000	32.18	0.70	97.82
11 000	52.88	1.20	97.73
14 000	87.79	1.24	98.59
17 000	129.27	1.81	98.60
19 897	179.40	2.37	98.68

可以看出, 基于哈希表的方法仍优于基于数据库查询的方法, 且加速比有随着文件个数增大而增大的趋势. 7 种环境下的平均时间加速比为 97.48%.

4 结语

针对 RPKI 中 RP 服务器使用基于数据库查询的方法进行证书链构建时效率低下的问题, 本文结合了 RPKI 运行机制中将计算代价转移至 RP 服务器的特点和“空间换时间”的思想, 设计实现了一种基于哈希表的 RPKI 证书验证优化方法, 有效的提升 RP 服务器在验证证书时的性能. 实验结果表明, 与基于数据库查询的方法相比, 基于哈希表的方法在构建证书链时效率显著提升, 在设计的 3 种实验场景下, 平均时间加速比分别为 99.03%、98.45% 和 97.48%, 有效的减少了验证证书的时间消耗.

参考文献

1 Phokeer AD. Interdomain routing security: Motivation and challenges of RPKI. Timss Technical Report RHUL-MA-2014-14, Egham, UK: Royal Holloway, University of London, 2014.

2 马迪. RPKI 概览. 电信网技术, 2012, (9): 30–33.

3 Lepinski M, Kent S, Kong D. RFC 6482: A profile for route origin authorizations (ROAs). IETF, 2012.

4 Austein R, Huston G, Kent S, *et al*. RFC 6486: Manifests for the resource public key infrastructure (RPKI). IETF, 2012.

5 Bush R. RFC 6493: The resource public key infrastructure (RPKI) ghostbusters record. IETF, 2012.

6 Huston G, Loomans R, Michaelson G. RFC 6481: A profile for resource certificate repository structure. IETF, 2012.

7 许圣明, 马迪, 毛伟, 等. 基于有序哈希树的 RPKI 资料库数据同步方法. 计算机系统应用, 2016, 25(6): 141–146. [doi: 10.15888/j.cnki.csa.005203]

8 Bruijnzeels T, Muravskiy O, Weber B, *et al*. Draft-ietf-sidr-delta-protocol, 2014.

9 Reynolds MC, Kent S. A high performance software architecture for a secure internet routing PKI. Proceedings of Cybersecurity Applications & Technology Conference for Homeland Security. Washington, DC, USA. 2009. 49–53.