中国科学: 信息科学
SCIENTIA SINICA Informationis

《中国科学》杂志社
SCIENCE CHINA PRESS

论文

可验证布隆过滤器: 加速区块链中的不存在查询与证明

徐松松[1,2], 过晓冰[3], 徐恪[1,2*]

1. 清华大学计算机科学与技术系, 北京 100084
2. 北京信息科学与技术国家研究中心, 北京 100084
3. 联想研究院, 北京 100085
* 通信作者. E-mail: xuke@tsinghua.edu.cn

收稿日期: 2023–02–25; 修回日期: 2023–05–08; 接受日期: 2023–06–25; 网络出版日期: 2023–12–12

国家重点研发计划 (批准号: 2022YFB3102301)、国家杰出青年科学基金 (批准号: 61825204)、国家自然科学基金 (批准号: 61932016, 62132011)、北京高校卓越青年科学家计划 (批准号: BJJWZYJH01201910003011) 和联想青年科学家资助项目

摘要 传统基于区块链的 "真实存储" 系统在过滤 "无效查询请求" 时忽略了提供 "不存在证明", 恶意节点可以随时对指定用户发动拒绝服务攻击. 本文提出了可验证布隆过滤器的一种构建方式, 基于布隆过滤器快速过滤无效查询请求的同时能有效提供证据证明数据不存在; 此外, 针对证明过程中可能造成的隐私泄露问题, 本文提出了 "隐秘的可验证布隆过滤器" 和 "数据混淆" 两种方式, 前者确保每个 "不存在证明" 只会泄露布隆过滤器的一位置零位, 减少了数据泄露量; 后者则是在前者的基础上进一步降低用户从泄露的布隆过滤器中推测出真实内容准确率. 实验数据表明, 当无效查询请求量占比为 35% 时, 读取性能提升大约 30%; 当无效查询请求量占比为 95% 时, 读取性能可提升十倍以上.

关键词 区块链, 可验证布隆过滤器, 可认证数据结构, 不存在证明, 隐私保护

1 引言

区块链最初是作为分布式、去中心化的公开账本 [1] 被提出的, 每个共识节点在本地存储该账本并基于共识机制来实现账本的一致性, 少数恶意节点无法篡改账本内容, 从而确保账本的真实性. 近年来, 区块链技术快速发展并拓展应用到身份管理 [2~5]、数据管理 [6~8]、网络安全 [9~12] 等领域, 此类应用所要求的存储数据量相比于最初数字货币的场景有显著提升, "真实存储" 这一存储模式也因此被广泛研究. "真实存储" 的主要思想是将区块链和可认证数据结构 (authenticated data structure, ADS) [13~15] 结合, 其架构如图 1 所示, 区块链不再确保所有数据的不可篡改, 而是通过 ADS 来存储数据的哈希、索引等开销小的信息, 确保 "数据存储" 这一行为的真实性以及实际数据的可验证性, 从而降低存储开销.

引用格式: 徐松松, 过晓冰, 徐恪. 可验证布隆过滤器: 加速区块链中的不存在查询与证明. 中国科学: 信息科学, 2023, doi: 10.1360/SSI-2023-0048
Xu S S, Guo X B, Xu K. Verifiable Bloom filter (VBF): accelerate the query and proof of nonexistent data in a blockchain (in Chinese). Sci Sin Inform, 2023, doi: 10.1360/SSI-2023-0048

© 2023《中国科学》杂志社

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

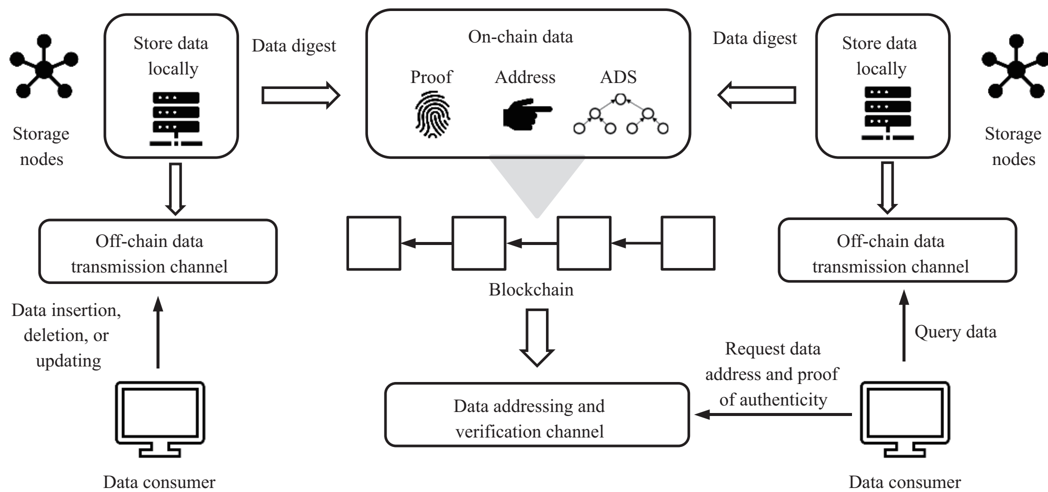

图 1　真实存储架构图

Figure 1　The architecture of true-storage

存储节点在回应用户的数据访问请求时, 则基于 ADS 来构造并提供该查询结果的完整性证据, 确保用户可以验证数据的真实性. 然而, 针对 ADS 的传统研究 [16,17] 以及现阶段的相关研究 [18~20] 都忽略了 "真实存储" 所需的重要特性: 针对结果为 "不存在" 的查询请求, 如何快速生成 "不存在" 证明.

在 "真实存储" 模式中, 若采用默克尔 (Merkel) 树等只能提供存在性证明的 ADS, 恶意存储节点可以针对特定用户返回 "数据不存在" 这一结果, 从而发动拒绝服务攻击; 而 MPT 树、密码累加器等技术虽然能够提供 "不存在证明", 但却因为证明构造开销大, 在需要实时提供 "不存在证明" 的场景中 (例如证书撤销列表 [2,5]、BGP 宣告验证 [10]) 会严重影响整个系统的查询性能. 针对这一问题, 传统数据库通常使用布隆过滤器来快速过滤无效查询请求, 布隆过滤器能够以非常小的存储开销标记大量数据的存在性, 在实现 $O(1)$ 查询复杂度的同时能确保 "不存在" 这一查询结果的 100% 准确 ("存在" 结果则有一定的错误率). 因此, 布隆过滤器不仅能够快速过滤无意义的数据查询请求 [21,22], 还因为其非常小的存储开销可以直接放在更快的存储介质中辅助实现双通道查询方案 [23~25]. 然而, 在 "真实存储" 这一模式中, 直接使用布隆过滤器虽然能够快速确认无意义的数据查询, 却无法有效构造证据证明 "不存在" 这一查询结果, 区块链节点在查询布隆过滤器确认数据不存在后, 依然需要按照原来的 ADS 结构来查询并构造不存在证明, 布隆过滤器的加速效果严重受到影响. 为了赋予布隆过滤器可验证的特性, 一种简单的方式是将布隆过滤器的哈希记录在区块头部, 并将完整的布隆过滤器返回给验证者 [2,5], 但这种简单提供整个布隆过滤器的处理方法不仅严重增加证明的数据大小, 在诸如智慧医疗等隐私较为重要的场景中还会泄露其中所包含的数据隐私 [26,27].

针对上述问题, 本文首先将布隆过滤器和默克尔树相结合, 提出了布隆过滤器的一种可验证构造方法, 该方法将布隆过滤器以二进制矩阵的形式进行初始化, 并结合数据撤销列表, 来实现数据的删除功能, 解决布隆过滤器无法删除数据的问题; 区块链系统每产生新区块时, 会将布隆过滤器按行构造默克尔树, 当节点查询到数据不存在后, 只须返回布隆过滤器中的一行, 用其中的置零位来构造不存在证明. 相比于将整个布隆过滤器返回给用户, 该方案节省了对 "不存在" 查询结果的验证开销和传输开销.

针对布隆过滤器在验证过程中的隐私保护问题, 本文从减少布隆过滤器数据泄露量和降低攻击者识别准确率两个方面分别提出隐私保护方案, 前者是构造隐秘的可验证布隆过滤器 (hidden verifiable

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

Bloom filter, H-VBF), 主要思路是用随机二进制矩阵 (random binary matrix, RBM) 对原布隆过滤器进行隐藏处理并得到隐秘布隆过滤器 (hidden Bloom filter, HBF), 然后分别将 HBF 和 RBM 按行构造默克尔树, 并在提供证明时分别返回 HBF 和 RBM 中的一行, 确保用户只能恢复原布隆过滤器中的一位置零位, 减少布隆过滤器的泄露量; 后者是在布隆过滤器中加入少量混淆数据, 降低攻击者从泄露的布隆过滤器中猜测 "存在数据" 的准确率.

本文对可验证布隆过滤器及其隐私保护方案进行了实现, 并对其进行性能测试. 实验数据表明, 当无效查询请求量占比为 35% 时, 查询性能提升约 30%; 当无效查询请求量占比为 95% 时, 查询性能可提升十倍以上.

2 背景技术及相关工作

本节主要对布隆过滤器、默克尔哈希树、MPT 树等背景技术及不存在证明相关工作进行介绍, 并阐述可验证布隆过滤器与现有相关工作相比的主要优势.

2.1 布隆过滤器

布隆过滤器 (Bloom filter, BF) 主要用于检索某元素是否存在于特定集合中, 其具备极高的空间效率和 $O(1)$ 的查询时间. 布隆过滤器通常由长度为 m 的二进制数组构成, 当插入数据时, 首先获取其 k 个特征字段, 然后将每个字段映射到 BF 中的某个位置, 把对应的 BF 位设置为 1; 当需要查询某数据是否存在时, 用同样的方法获取该数据的 k 个特征字段, 并查询 BF 中对应的 k 个位置是否同时为 1, 若是则说明该数据极大概率存在; 若任意一位为 0, 则该数据必然不存在. 在布隆过滤器的实际应用中, 特征字段的获取方法有多种, 常见的为 k 个哈希函数, 或者将哈希值等分为 k 段.

BF 存在一定的错误率, 但可以通过调节布隆过滤器长度和特征字段个数将错误率降到极低. 当构造可记录 n 条数据的 BF 时, 其大小 m 和特征字段个数 k 的最优选择可以通过下式获得:

$$k = \frac{m}{n}\ln 2,$$

$$m = -\frac{n\ln p}{\ln 2}.$$

不同 k 和 m/n 值对应的布隆过滤器错误率 p 可以在布隆过滤器参数表中查询, 表 1 为其中的部分内容, 表中第 2 列为 k 的最优设计值. 由于在实际应用过程中, k 必须为整数, 所以 k 通常取比最优值小的最大整数值.

布隆过滤器能够显著提升存储和查询效率, 以记录十亿条 (即 1 G) 数据为例: 若选择 6 个哈希函数, 并确保错误率不超过 1%, 则 m/n 可以选择 10, 因此布隆过滤器大小 m 为 10 Gb, 即 1.25 GB, 可以直接存放于服务器内存中; 若不用布隆过滤器, 而是记录每个数据的哈希值, 每个哈希值 32 字节, 则存储开销为 32 GB, 一般的服务器无法将其完整存入内存.

正因为具备低存储开销和高查询效率, 布隆过滤器近年来也被大量用于区块链及相关应用场景中, 例如以太坊 [28] 中, 全节点会为每个区块中的交易构建一个布隆过滤器, 用于快速确认特定交易不在对应的区块中, 但是以太坊节点使用布隆过滤器的目的是优化查询效率, 并没有基于布隆过滤器给出不存在证明. 证书链 [2] 基于布隆过滤器进行改进并设计了双计数布隆过滤器, 用于记录证书状态, 实现证书状态快速查询的同时消除错误率; PROCESS [29] 采用与证书链类似的思想, 并基于改进的布隆过滤器实现数据删除的同时, 显著降低链上存储的数据量; Zhang 等 [20] 在针对关键字搜索的区块链

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

表 1 布隆过滤器参数表

Table 1 The parameter list of Bloom filter

m/n	k	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$
4	2.77	0.16	–	–	–	–
5	3.46	0.092	0.101	–	–	–
6	4.16	0.0561	0.0578	0.0638	–	–
7	4.85	0.0359	0.0347	0.0364	–	–
8	5.55	0.024	0.0217	0.0216	0.0229	–
9	6.24	0.0166	0.0141	0.0133	0.0135	0.0145
10	6.93	0.0118	0.0094	0.0084	0.0082	0.0085
11	7.62	0.0086	0.0065	0.0055	0.0051	0.0051
12	8.32	0.0065	0.0046	0.0037	0.0033	0.0032

应用场景中, 采用布隆过滤器记录数据的存在性, 并将每固定数量的数据用一个小规模布隆过滤器存储, 用于快速确认目标数据是否存在, 用户也可以浏览链上的历史布隆过滤器从而快速验证数据的存在性. 然而, 上述这些工作主要是优化服务端的查询和验证效率, 基于布隆过滤器为用户提供证明时要求用户获取整个布隆过滤器进行验证, 不仅开销大, 还会泄露其中包含的数据隐私. 相比之下, 可验证布隆过滤器的主要目标则是确保能快速过滤不存在数据的同时以更低的开销提供不存在证明, 并保证数据的隐私.

2.2 默克尔树

默克尔哈希树 [30] (Merkel Hash tree, MHT) 如图 2 所示是可认证数据结构的一种, 可对单个交易的正确性进行高效验证, 从而实现简易支付证明 [1]. 在 MHT 树中, 每个交易的哈希值作为默克尔树的叶子节点, 每个中间节点为其左右子节点相结合后的哈希值, 树根节点则通常被存储于区块链头部中作为证据. 如图 3 所示, 当用户需要对交易 Data_1 进行验证时, 可以请求区块链节点返回 Hash_2, Hash_6; 然后将 Data_1 进行哈希获得 Hash_1, 并结合所返回的哈希值计算出根哈希; 将计算出的根哈希与本地存储区块头部中的根哈希进行比对, 若相等则证明了交易的完整性. 尽管 MHT 已经在区块链中被广泛应用, 以实现对具体交易信息存在性的证明, 但其无法被用来构建不存在证明. 针对这一问题, 部分研究 [31,32] 提出有序默克尔树, 通过对默克尔树的叶子节点进行排序, 并依靠两个相邻的存在性证明来说明中间所查询的数据不存在, 但这类工作要求每次向 MHT 中加入数据时将叶子节点进行排序并重构默克尔树, 增加了默克尔树的实际维护难度. 相比之下, 可验证布隆过滤器可以直接和默克尔树相结合, 保证原本功能不变的同时实现不存在数据的快速过滤并提供不存在证明.

2.3 MPT 树及不存在证明相关研究

Merkle Patricia Trie (MPT) 树是以太坊中的一种加密认证数据结构, 既能提供存在性证明, 又能提供不存在性证明, 其结构设计中主要包含扩展节点、分支节点和叶子节点 3 种节点类型:

(1) 扩展节点 (extension node): 只有一个子节点, 其中的 Key 值为其子孙节点的部分公共前缀, 其中的 Value 值则记录了子节点的哈希值, 在对子节点进行索引的同时确保其完整性.

(2) 分支节点 (branch node): 继承从根节点到父节点路径上的公共前缀, 并开始出现分支; 分支节点最多产生 16 个分支, 每个分支的 Key 为其子节点需要继承的 4 位二进制前缀.

(3) 叶子节点 (leaf node): 没有子节点, 代表一条路径的终止, 其中 Key 记录了该叶子节点的剩余

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

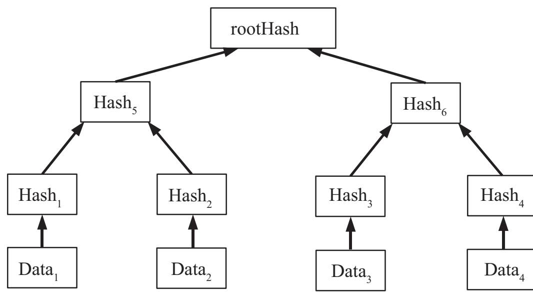

图 2 默克尔树
Figure 2 Merkel tree

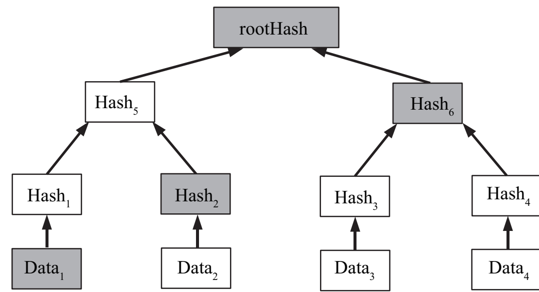

图 3 默克尔证明
Figure 3 Merkel proof

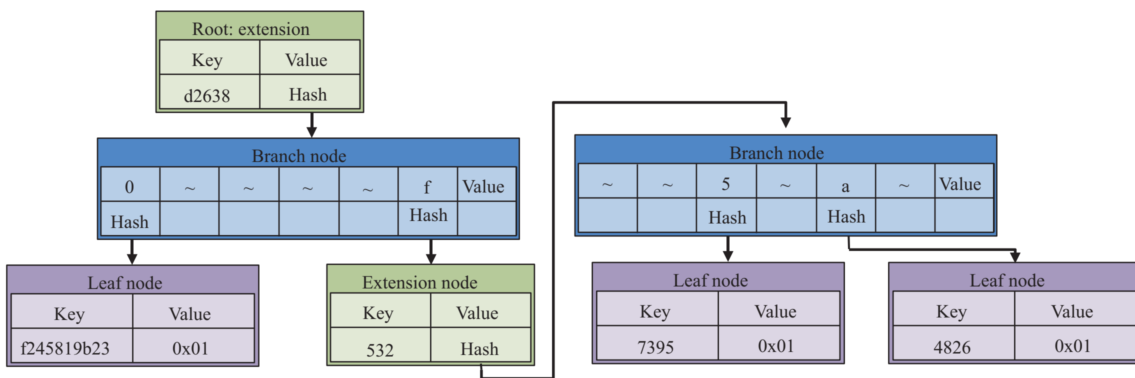

图 4 MPT 树示意图
Figure 4 Sketch of the MPT tree

前缀, 将从根节点到叶子节点路径上的 Key 值组合起来则是完整 Key 值.

如图 4 所示为 MPT 树的一个例子, 在 MPT 树中, 若要证明 Key 为 d2638f53247692c8 的键值对不存在, 只需要证明 Key 前缀为 d2638f5324 的节点指向空节点, 默克尔证明则是根节点到该证明节点路径上的所有节点, 即图中的 2 个扩展节点和 2 个分支节点.

MPT 树尽管能验证数据的不存在性, 但因为要记录所有交易 (或交易哈希) 而导致存储开销较大, 难以直接放在内存中, 需要用数据库 (例如 LevelDB) 进行存储. 因此, MPT 树在面临大量的 "不存在证明" 构造需求时, 会造成频繁的数据库访问, 导致整体效率较低. 近几年, 大量研究针对 MPT 树的不足提出了新的 ADS [18~20], 这些工作在丰富查询与验证接口的同时提高了查询效率. 然而, 现阶段的相关研究主要集中在数据真实性验证, 且对存储要求较高, 与 MPT 树一样必须存储于数据库中, 无法针对数据 "不存在" 的结果快速提供 "不存在" 证明. 此外, 密码学累加器近些年也被提出用于构建无状态区块链 [33], 试图用其替代默克尔树、MPT 树等提供数据查询结果的证明. 密码学累加器既能提供存在证明, 也能提供不存在证明, 但因为证明构造过程的计算开销太大, 无法在线实时提供证明. 相比之下, 可验证布隆过滤器存储开销较小, 能直接存放于内存中, 且计算开销极低, 能够实时提供不存在证明.

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

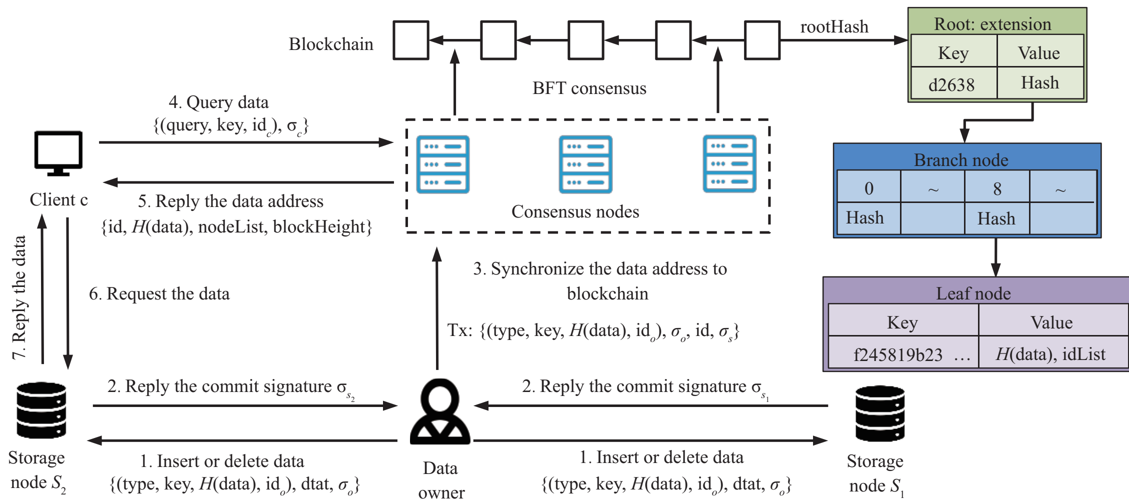

图 5 系统架构图

Figure 5 System model

3 应用场景

可验证布隆过滤器适用于任意 “真实存储” 平台中, 快速判断数据的 “不存在” 并构造相应的证据. 为了对可验证布隆过滤器及其隐私保护方案进行功能验证与性能测试, 本文设计了单键值对场景的真实存储系统, 该系统支持数据写入、数据删除、数据查询 3 个功能. 整个系统包括存储节点、共识节点、数据拥有者和客户端 4 种角色. 数据拥有者请求存储节点完成数据存储和删除操作, 并将该操作同步记录到区块链中. 共识节点维护每个数据的哈希值, 以及存储该数据的节点 id, 用户在链上获取到存储数据节点的 id 后请求对应的存储节点来获取相关的具体数据, 并结合链上返回的证据验证数据的完整性. 本文假设已经建立好 PKI 机制和地址映射机制, 客户端、存储节点、共识节点和数据拥有者能互相根据身份 id 获取对应的公钥以及访问地址. 本节将围绕图 5 简要介绍系统中的各个交互流程.

3.1 数据存储与删除

数据拥有者可以向任意存储节点发送数据存储与删除请求, 消息格式如下所示:

$$\mathrm{req} := \{(\mathrm{type}, \mathrm{key}, H(\mathrm{data}), \mathrm{id}_o), \mathrm{data}, \sigma_o\}, \tag{1}$$

其中 type 为 Insert 或 Delete, 分别代表数据插入与删除操作, key 为数据的关键字; $H(\mathrm{data})$ 为具体数据内容 data 的哈希值; id_o 为数据拥有者的身份, σ_o 则是数据拥有者对消息 $(\mathrm{type}, \mathrm{key}, H(\mathrm{data}), \mathrm{id}_o)$ 的哈希值的签名. 当存储节点收到了来自数据拥有者的请求后, 先预执行相应的数据插入、删除请求, 并返回确认执行的签名 σ_s, 该签名是存储节点用私钥对请求内容中 $(\mathrm{type}, \mathrm{key}, H(\mathrm{data}), \mathrm{id}_o)$ 的哈希值的签名.

3.2 数据信息上链

当用户收集到存储节点的签名后, 将签名与原请求相结合并构造出 “数据信息上链” 交易, 该交

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

易格式如下所示:

$$tx := \{(\text{type}, \text{key}, H(\text{data}), \text{id}_o), \sigma_o, \mathbf{id}, \boldsymbol{\sigma}_s\}. \tag{2}$$

消息包含完成了数据存储操作的存储节点签名列表 $\boldsymbol{\sigma}_s$ 和存储节点身份列表 **id**, 该交易被发往区块链后, 共识节点验证交易中数据拥有者和存储节点的签名, 成功后将对应的信息更新到链上, 消息种类包含以下两种情况.

(1) 数据添加: 当交易中包含的原请求是数据添加请求时, 完成签名验证后, 将数据的 key 编码为 MPT 树中索引时用到的 Key, MPT 树中的 Value 字段则记录数据的哈希值, 以及宣称存储了该数据的存储节点的 idList. 如图 5 所示为使用 MPT 树作为 ADS 的例子, 树根节点 rootHash 记录在区块头部中. 在实际部署过程中, 可以对未真实存储数据的节点实施经济惩罚.

(2) 数据删除: 当交易中包含的原请求是数据删除请求时, 链节点在 MPT 树中找到存储该数据的叶子节点, 从其中记录的存储节点身份列表中删除交易中附带的存储节点身份列表 **id**, 被删除的存储节点无须再受理与该数据相关的访问请求; 当对应数据无任何存储节点存储时, 则在 MPT 树中将该数据删除.

存储节点需要确认链上交易的执行情况, 当确认链上状态更新后再真正执行相应的请求, 从而避免链上链下数据不同步.

3.3 链上数据索引

客户端在查询数据之前, 先根据数据的关键字 key 向区块链发出查询请求获取数据索引, 请求格式如下所示:

$$tx := \{(\text{query}, \text{key}, \text{id}_c), \sigma_c\}. \tag{3}$$

区块链节点收到该请求后, 先验证交易的签名, 验证通过后将 key 编码为 MPT 树中 Key 并进行查询, 若数据存在, 则基于 MPT 树构造默克尔证明, 然后将数据哈希值 $H(\text{data})$ 和存储节点的身份列表 **id** 返回给用户 (少部分链节点可能会针对特定客户端拒绝回复, 此时客户端需要选择其他链节点进行查询), 回复消息格式如下所示:

$$\text{reply} := \{\mathbf{id}, H(\text{data}), \text{nodeList}, \text{blockHeight}\}. \tag{4}$$

默克尔证明是从 MPT 树根节点到存储索引信息的叶子节点路径上所有节点构成的列表 nodeList, blockHeight 是该证明所对应的区块高度; 若查询到数据不存在时, nodeList 则可直接作为不存在证明, 然后将回复消息中的 **id** 和 $H(\text{data})$ 都设置为空.

客户端在收到来自区块链节点返回的消息后, 先根据 blockHeight 确认其中的证明信息是基于最新的区块构建的 (客户端可以设置容忍值 t, 接受落后于最新区块不超过 t 个高度的证明信息, t 可以根据用户的容忍程度进行设置, 本文中 t 被设置为 6), 然后验证 nodeList 中每个中间节点所记录的哈希值确实与子节点的哈希值相同, **id** 和 $H(\text{data})$ 也确实在叶子节点中; 最后判断 nodeList 中根节点的哈希值是否和本地所同步区块头部中记录的哈希值相同; 若上述验证都通过, 即可确认存储信息的正确性, 若存储节点身份列表非空, 则从列表中随机选择一个存储节点获取真实数据, 否则放弃此次查询.

3.4 链下数据查询

客户端在获取存储节点列表后, 构造请求向存储节点查询真实数据, 请求格式如下所示:

$$\text{req} := \{\text{query}, \text{key}, \text{blockHeight}, H(\text{data})\}, \tag{5}$$

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

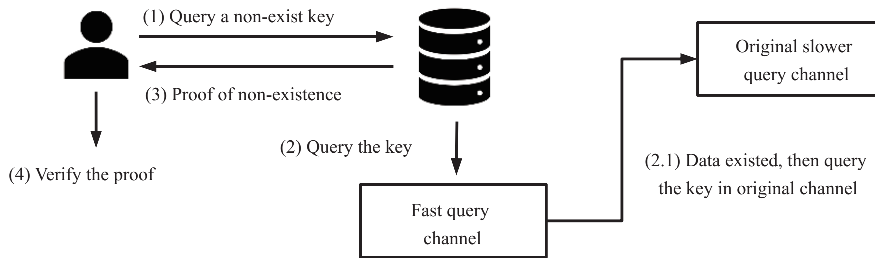

图 6　双通道查询方案

Figure 6　Dual-channel query scheme

其中 query 和 key 两个参数与链上查询时请求消息中的参数相同, blockHeight 为链上查询索引信息时链节点所返回的区块高度; 此外, 在发送请求消息时, 客户端需要附带对请求消息哈希值 $H(\text{req})$ 的签名 σ_c. 当存储节点收到该请求并完成签名验证后, 若本地存有该数据且数据哈希和 $H(\text{data})$ 相同, 则直接返回该数据; 若本地未存储该数据或者所存数据的哈希值与 $H(\text{data})$ 不同, 则构造回复消息如下所示:

$$\text{reply} := \{\text{data}, H(\text{req}), \sigma_s, \text{blockHeight}, \text{nodeList}\}, \tag{6}$$

其中 data 为 key 对应存储节点本地存储的数据 (若不存在, 则为空), $H(\text{req})$ 为用户查询请求的哈希值, σ_s 为存储节点对 $H(\text{data}, H(\text{req}))$ 的签名, blockHeight 应该大于请求消息中的区块高度, nodeList 则是用于证明 data 正确性的 MPT 证明. 当客户端完成对回复消息中数据的正确性验证后, 选择接受该数据.

3.5　双通道快速查询方案

可验证布隆过滤器的主要目的是快速过滤无效查询请求, 并提供证据证明数据确实不存在, 从而为实际的数据查询增添快通道. 基于可验证布隆过滤器的双通道数据查询模式如图 6 所示, 当用户向区块链节点查询数据时, 节点先基于可验证布隆过滤器进入快通道查询模式, 若查询数据存在于布隆过滤器中, 再进入原本的慢通道查询模式 (即 MPT 树). 在这样一种双通道查询模式下, 即使布隆过滤器中数据 “存在” 这一查询结果存在错误率, 存储节点也能基于慢通道查询模式获取最终正确结果, 从而消除布隆过滤器错误率的影响. 本文主要基于上述系统模型对可验证布隆过滤器进行部署与验证.

4　可验证布隆过滤器

本节将提出一种可验证布隆过滤器的构建方法, 该方法的主要思想是将传统布隆过滤器和默克尔树相结合, 可以快速构造证明用于用户验证某数据确实未被记录在布隆过滤器中. 该方法在提供证明时会泄露部分布隆过滤器的数据, 因此主要用于针对非隐私数据的存储系统中. 可验证布隆过滤器在使用过程中具体包含初始化, 数据记录、删除与查询, 默克尔树与 MPT 树的构建, 不存在证明的生成, 不存在证明的验证和布隆过滤器重构与扩容, 本节将对这 6 个部分进行详细介绍.

4.1　布隆过滤器初始化

与传统布隆过滤器二进制向量构造形式不同, 本方案采用二进制矩阵的形式构建布隆过滤器, 并结合数据撤销列表实现数据的删除功能.

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

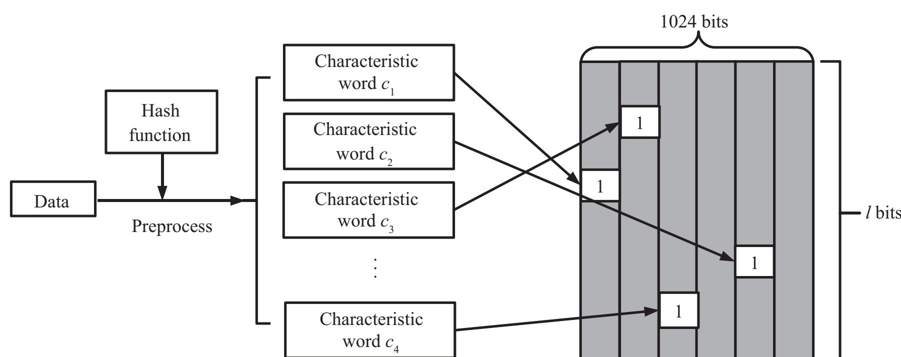

图 7　可验证布隆过滤器

Figure 7　Verifiable Bloom filter

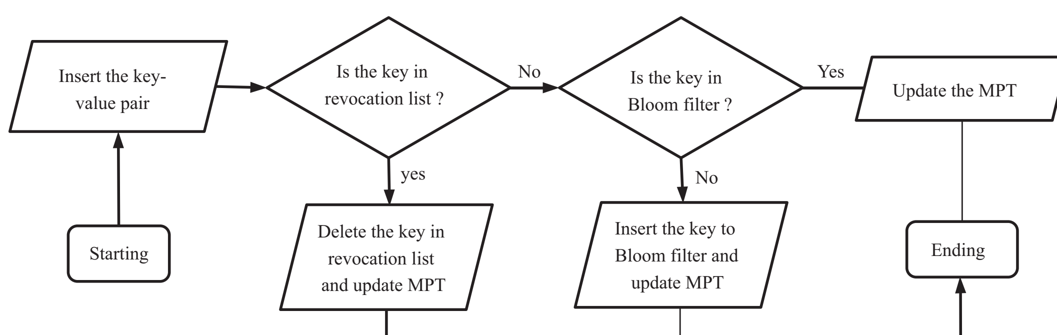

图 8　数据添加流程图

Figure 8　Data insertion process

初始化时, 先按照传统布隆过滤器的尺寸选择模式 (如表 1 所示) 选择布隆过滤器的错误率并确定布隆过滤器的容量 n 和规模 m 以及每次记录数据需要置 1 的位数 k, 然后根据布隆过滤器的规模确定具体的行数和列数. 如图 7 所示, 布隆过滤器的行数为 l, 列数为 1024, 总规模为 $m = 1024l$ 比特, 数据撤销列表是根据实际需求动态增长的哈希列表, 用于记录从布隆过滤器中撤销的数据哈希.

当向布隆过滤器中记录某数据时, 首先获取其 k 个特征字段, 然后按照如下式 (7) 将每个字段 c_k 映射到 BF 中的第 i 行和第 j 列, 并将该字段对应的 BF 位设置为 1:

$$i = \lfloor c_k/1024 \rfloor \% l, \quad j = c_k \% 1024. \tag{7}$$

当查询某数据是否存在时, 用同样的方法获取该数据的 k 个字段, 并查询 BF 中对应的 k 个位置是否同时为 1, 若是则说明该数据极大概率存在; 若任意一位为 0, 则该数据必然不存在.

4.2　数据记录、删除与查询

与传统布隆过滤器不同, 可验证布隆过滤器的数据记录、删除与查询流程还需考虑撤销列表, 并针对每一次查询过程提供证明. 其中数据记录流程如图 8 所示, 主要包含以下 3 个步骤:

(1) 先查询该 Key 值是否存在于撤销列表中, 若存在, 则说明该 Key 已记录在布隆过滤器中但曾经被撤销过, 所以只需要将其从撤销列表中删除, 并将此次记录的键值对数据更新到 MPT 树中;

(2) 若 Key 不在撤销列表中, 则需要查询其是否存在于布隆过滤器中, 即将该 Key 字段的哈希值

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

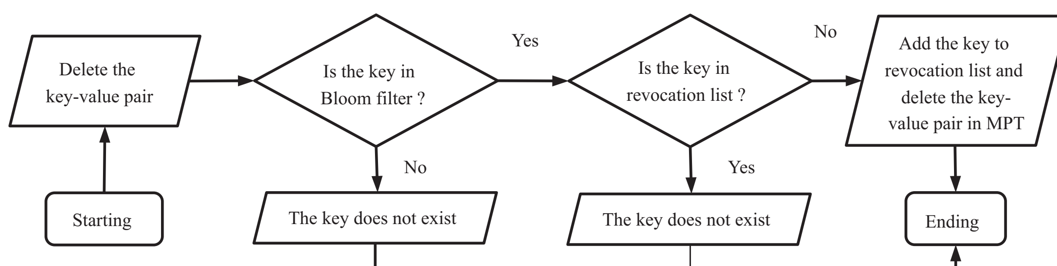

图 9　数据删除流程图

Figure 9　Data deletion process

分成 k 段并映射到布隆过滤器的 k 位, 若其中任意一位为 0, 则说明该数据定然不存在, 则将对应的 k 位全部设置为 1, 并将此次记录的键值对数据更新到 MPT 树中;

(3) 若 Key 值不存在于撤销列表但存在于布隆过滤器中, 则说明该 Key 值对应的键值对大概率已经存在, 则只需要将此次记录的数据更新到 MPT 树中即可.

当需要删除某键值对时, 则进行如图 9 所示的 3 个步骤:

(1) 先查询该 Key 值是否存在于布隆过滤器中, 即将该 Key 字段的哈希值分成 k 段并映射到布隆过滤器的 k 位, 若其中任意一位为 0, 则说明该数据定然不存在, 所以无须执行删除操作;

(2) 若 Key 值存在于布隆过滤器中, 则进一步判断该 Key 值是否存在于撤销列表中, 若是则说明该数据已经被撤销, 因此也无须执行删除操作;

(3) 若 Key 值存在于布隆过滤器中且不存在于撤销列表中, 则说明其对应的键值对大概率已经存在, 则需要将该 Key 值记录到撤销列表, 并将 MPT 树中对应的键值对删除.

在进行数据查询时, 则需要结合布隆过滤器、数据撤销列表和 MPT 树来综合判断, 其流程如图 10 所示, 主要分为如下 3 个步骤:

(1) 先查询该 Key 值是否存在于布隆过滤器中, 即将该 Key 字段的哈希值分成 k 段并映射到布隆过滤器的 k 位, 若其中任意一位为 0, 则说明所查询数据不存在, 此次查询为无效查询, 然后基于布隆过滤器构造不存在证明;

(2) 若 Key 值存在于布隆过滤器中, 则进一步判断该 Key 值是否存在于撤销列表中, 若是则说明该数据不存在, 此次查询为无效查询, 然后基于数据撤销列表构造不存在证明;

(3) 若 Key 值存在于布隆过滤器中且不存在于撤销列表中, 则说明所查询的数据极大概率存在, 此时通过慢通道 MPT 树进行查询, 并基于 MPT 树构造相关证明.

4.3　默克尔树与 MPT 树的构建

区块链系统每产生新区块时, 对应的真实数据哈希是基于 MPT 树存储的, 其树根哈希用 root_0 表示; 此外, 共识节点重新将布隆过滤器按行构造默克尔树, 其树根哈希用 root_1 表示; 数据撤销列表中的内容则使用 MPT 树进行存储, 其树根哈希用 root_2 表示, 根哈希都将存入区块头部. 针对布隆过滤器构造默克尔树时, 将图 2 中的 Data_i 换为 $\{i, \mathrm{BF}[i][:]\}$, 其中 $\mathrm{BF}[i][:]$ 代表布隆过滤器中的一行, i 则是对应的行号.

针对数据撤销列表构造 MPT 树时, 每个被撤销的 key 值被编码成一段固定长度的字节作为编码添加到 MPT 树中的 Key, Value 则为 2 个 16 进制数 0x01. 若要证明某个 key 存在于撤销列表中, 则需要返回一组节点路径, 该路径中第一个节点的哈希值为根哈希; 每个父节点记录的哈希值为子节点

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

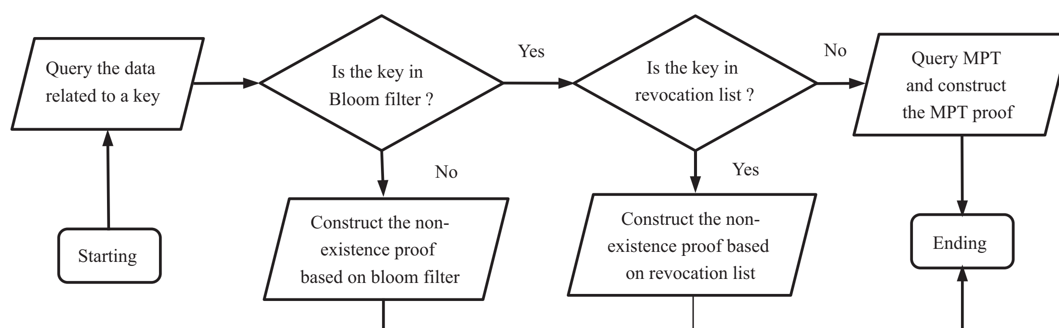

图 10　数据查询流程图

Figure 10　Data query process

的哈希; 路径中最后一个节点为叶子节点, 其记录的 Value 为 0x01; 路径中所有节点 Key 能组合成 key 编码后的完整 Key. 若要证明某个 key 不存在于撤销列表中, 同样返回一组节点路径, 但其中最后一个节点指向空节点.

4.4　不存在证明的生成

不存在证明的生成流程与数据查询结果相关, 在处理用户的查询与验证请求时, 主要存在 3 种情形:

(1) 若布隆过滤器查询结果为 "数据存在", 且在数据撤销列表中也不存在该数据, 则按照慢通道查询与验证方式, 即获取 MPT 树的查询结果并构造相应的默克尔证明.

(2) 若布隆过滤器的查询为 "不存在", 该数据映射到布隆过滤器的 k 位中必然包含置零位, 则只需要获取该位所在的行, 并针对该行构造不存在证明发给用户.

(3) 若布隆过滤器的查询为 "存在", 但在数据撤销列表中能查到数据已被撤销, 则只需要针对该撤销表项构造默克尔证明.

构造完不存在证明后, 共识节点即可构造完整的回复消息发送给用户, 其格式为

$$m := \{\mathrm{NE}, \mathrm{type}, \mathrm{proof}_{\mathrm{NE}}\}\,, \tag{8}$$

其中 NE 代表数据不存在, type 可能为 0, 1 或 2, 0 代表后续的默克尔证明是基于慢通道原数据的 MPT 树构建; 1 代表默克尔证明是基于布隆过滤器中某个置零位所在行构建的; 2 则代表证明是基于数据撤销列表中的具体表项而构建的.

type 为 0 和 2 的证明主要包含 MPT 树中的一组节点列表, 其格式如下所示:

$$\mathrm{proof}_{\mathrm{NE}} := \{\mathrm{nodeList}, \mathrm{blockHeight}\}\,, \tag{9}$$

其中 blockHeight 是构建该证明时对应的区块高度, nodeList 则是用于证明数据不存在的 MPT 节点列表, 当 type 为 0 时, 该节点列表的所有 Key 值所组成的字段能构成所查询数据 key 编码后 Key 的某个前缀, 并最终指向一个空节点; 当 type 为 2 时, 该节点列表的所有 Key 值所组成的字段能构成所查询数据 key 编码后的 Key, 且最后一个节点是 Value 为 0x01 的叶子节点.

type 为 1 的证明则是包含一行布隆过滤器和对应默克尔树的默克尔路径, 其格式如下所示:

$$\mathrm{proof}_{\mathrm{NE}} := \{x, \mathrm{BF}\,[x]\,[:]\,, y, \mathrm{hashList}, \mathrm{blockHeight}\}\,, \tag{10}$$

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

其中 BF $[x]$ [:] 是布隆过滤器中的第 x 行, y 是第 x 行中对应置零位的列号; hashList 是 $x||$BF $[x]$ [:] 对应的默克尔证明路径, blockHeight 是构建该证明时对应的区块高度.

4.5 不存在证明的验证

当用户收到回复为 "不存在" 时, 先判断其中 type 的值, 然后进入对应的验证流程:

(1) 若类型为 0, 则说明该证明是基于 MPT 树进行构建的, 需要验证式 (9) 中节点列表 nodeList, 首先验证该路径中第一个节点的哈希值为 blockHeight 高度区块头部中所记录的根哈希 root_0; 然后验证每个父节点记录的哈希值确实为子节点的哈希; 之后验证路径中最后一个节点指向空节点; 最后验证路径中所有节点 Key 能组合成 key 编码后 Key 的某个前缀; 所有验证都通过后则接受该不存在回复.

(2) 若类型为 1, 说明该证明是基于布隆过滤器所构建的, 需要结合 hashList 验证, 即结合 hashList 中的内容与 $\{x,\text{BF}[x][:]\}$ 计算出根哈希, 然后和高度为 blockHeight 的区块中记录的 root_1 进行比对, 相同则可验证该行布隆过滤器的完整性; 然后, 需要确认 $\text{BF}[x][y]$ 为 0, 且所查询 key 存在某个特征值映射在布隆过滤器的第 x 行, 第 y 列; 上述验证都通过后则可确信所查询 key 不存在.

(3) 若类型为 2, 则说明该证明是基于数据撤销列表所构建的, 需要验证回复消息中的节点路径 nodeList, 同类型 0 一样通过验证 MPT 树节点路径的完整性确认数据存在于撤销列表中, 从而确认数据已经被删除不存在.

4.6 布隆过滤器重构与扩容

当布隆过滤器记录的数据量接近容量上限时, 共识节点会对布隆过滤器进行扩容, 该过程需要将布隆过滤器的规模扩大一倍, 然后重新遍历存储真实数据的 MPT 树, 将所有的 key 存入到新的布隆过滤器中, 并将撤销列表中的数据清空; 当撤销列表记录的数据量达到布隆过滤器实际容量的 $r\%$ 时 (r 为自定义参数), 也需要对布隆过滤器进行重构, 并将撤销列表中的数据清空. 在重构与扩容布隆过滤器时, 为了避免对共识造成影响, 可以采用离线扩容与重构设计, 假设共识节点在执行完第 N 个区块中的交易后判断要进行重构与扩容操作时, 开始执行重构过程, 并从第 $N+d$ 个区块开始使用新的布隆过滤器, d 为系统的初始参数, 设置时要确保产生 d 个块后重构过程能顺利完成. 此外, 为了确保重构过程中布隆过滤器存储的数据不会超过其容量上限, 扩容条件可设置为实际存储数据达到容量上限的 $(1-u)\%$, 其中 u 的设置与布隆过滤器容量上限以及 d 有关, 目的是确保 d 个区块包含的交易总量不超过原布隆过滤器容量的 $u\%$, 从而使布隆过滤器在扩容过程中所存储的数据不会超过其容量上限.

5 隐私保护方案

在前面的分析中, 基于可验证布隆过滤器提供不存在证明时会泄露一整行布隆过滤器数据, 为了解决这一问题, 本节提出了隐秘的可验证布隆过滤器和数据混淆两种隐私保护方案, 前者主要目标是减少不存在验证时泄露的原布隆过滤器位数, 但随着用户查询次数过多, 依然存在泄露隐私的风险; 后者是在前者的基础上进一步降低攻击者根据泄露的布隆过滤器获取真实数据结果的准确率.

5.1 隐秘的可验证布隆过滤器

隐秘的布隆过滤器 HBF 构造思路如图 11 所示, 其主要思路是在布隆过滤器的初始化、重构与扩容时, 先构建与布隆过滤器相同规模的随机二进制矩阵 RBM, 并将 BF 中的一位映射到 RBM 中的一

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

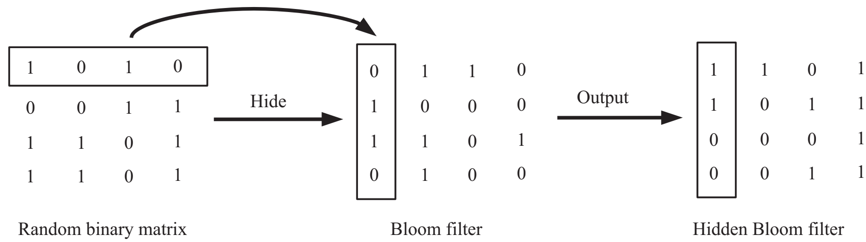

图 11　隐秘的可验证布隆过滤器构造示意图

Figure 11　Construction process of the hidden verifiable Bloom filter

位, 并用该位进行隐藏 (即进行异或计算), 从而得到 HBF. 在返回默克尔证明时, 需要同时返回 HBF 和 RBM 中的一行, 确保基于这两行数据只能恢复出原布隆过滤器的一位置零位, 从而验证查询数据的不存在性. 基于该方案, 在提供不存在证明时显著减少泄露的隐私量, 接下来将对该方案进行介绍.

(1) 布隆过滤器的初始化: 在系统初始化时, 共识节点会共识出一个 32 字节的随机种子 seed, 并基于该 seed 生成 RBM. 对于 RBM 的第 i 行, 可以基于以下公式计算 4 个哈希值后相连得到:

$$\mathrm{RBM}[i][:]=\{\mathrm{Hash}(\mathrm{seed}||4i),\mathrm{Hash}(\mathrm{seed}||4i+1),\mathrm{Hash}(\mathrm{seed}||4i+2),\mathrm{Hash}(\mathrm{seed}||4i+3)\}.\quad (11)$$

对于 BF 中的任意一位 $\mathrm{BF}[x][y]$, 可基于以下公式计算得到其映射到 RBM 中的一位 $\mathrm{RBM}[x'][y']$:

$$x'=\lfloor x/1024\rfloor\times1024+y,\quad y'=x\%1024.\quad (12)$$

在生成 RBM 后, 可以根据映射关系获取 HBF 中的每一位, 即 $\mathrm{HBF}[x][y]=\mathrm{BF}[x][y]\oplus\mathrm{RBM}[x'][y']$, 从而获得完整的 HBF.

(2) 数据记录、查询与删除: 该过程与传统可验证布隆过滤器相同, 只是在更新布隆过滤器时, 需要结合映射到 RBM 中的对应位同时更新 HBF.

(3) 默克尔树与 MPT 树的构建: 与可验证布隆过滤器相同, 数据撤销列表中的数据用 MPT 树进行存储, 树根哈希为 root_2; 此外, 无须对原布隆过滤器构造默克尔树, 而是针对 HBF 和 RBM 分别按行构造默克尔树, 构造方式与可验证布隆过滤器相同, 树根节点分别为 root_1 和 root_3.

(4) 不存在证明的生成: 针对用户的查询与验证请求, 链节点的处理流程和可验证布隆过滤器的处理流程相同, 并同样存在 3 种类型的处理流程. 取消了类型为 1 的不存在证明, 新增类型为 3 的不存在证明, 需要先将所查询数据的 key 映射到 BF 中的 k 个位置, 选取其中一位置零位 $\mathrm{BF}[x][y]$, 并构造不存在证明 P_{NE} 如下所示:

$$P_{\mathrm{NE}}:=\{x,y,\mathrm{HBF}[x][:],\mathrm{RBM}[x'][:],\mathrm{hashList}_1,\mathrm{hashList}_2,\mathrm{blockHeight}\},\quad (13)$$

其中 x 和 y 是置零位的行号和列号, x' 则根据式 (12) 所示映射关系获得, $\mathrm{HBF}[x][:]$ 是 HBF 的第 x 行, $\mathrm{RBM}[x'][:]$ 是 RBM 的第 x' 行, $\mathrm{hashList}_1$ 和 $\mathrm{hashList}_2$ 分别为用于验证 $\{x,\mathrm{HBF}[x][:]\}$ 和 $\{x',\mathrm{RBM}[x'][:]\}$ 的默克尔哈希列表, 高度为 blockHeight 的区块包含 $\mathrm{hashList}_1$ 和 $\mathrm{hashList}_2$ 对应的默克尔树根, 即 root_1 和 root_3.

(5) 不存在证明的验证: 当用户收到 "不存在" 这一回复时, 先判断回复中 type 的值, 然后进入对不同类型不存在证明的验证流程, 针对 type 为 0 和 2 的不存在证明, 其验证流程和可验证布隆过滤器一致. 针对 type 为 3 的不存在证明, 先基于式 (12) 计算出 x' 和 y'; 然后查询高度为 blockHeight 的区块并获取其中的根哈希 root_1 和 root_3; 其次, 结合默克尔哈希路径 $\mathrm{hashList}_1$ 和 $\mathrm{hashList}_2$ 分别验证

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

$\{x, \mathrm{HBF}[x][:]\}$ 和 $\{x', \mathrm{RBM}[x'][:]\}$ 的完整性; 验证通过后, 获取所查询 key 映射到 BF 中的 k 个位置, 并判断 (x, y) 是否为其中之一; 最后, 判断 $\mathrm{HBF}[x][y] \oplus \mathrm{RBM}[x'][y']$ 的结果, 若为 0, 则相信此次查询结果, 即所查询数据不存在, 若不为 0, 或之前的任意验证失败, 则不信任该回复.

(6) 布隆过滤器重构与扩容: 在执行完第 N 个区块并确定要进行重构与扩容后, 在第 $N+1$ 个区块的共识过程中, 会重新共识新的种子 seed, 用于生成新的 RBM.

5.2 数据混淆

隐秘的布隆过滤器虽然能显著减少数据的泄露量, 但随着攻击者多次请求 "不存在证明" 并获取多行 HBF 和 RBM 中的数据后, 依然可能推测布隆过滤器中的内容, 从而获取隐私信息; 基于数据混淆的可验证布隆过滤器 (data confusion based VBF, C-VBF) 则是从降低攻击者识别准确率 (recognition accuracy, RA) 的思路出发来保护数据隐私的, 在布隆过滤器构建过程中插入部分混淆数据, 确保用户在获得布隆过滤器内容时, 无法从中获取真实的内容. 在实际使用的过程中, 该方案可以和隐秘的布隆过滤器相结合实现更强的隐私保护能力, 接下来将对该方案进行介绍.

(1) 布隆过滤器的初始化: 与可验证布隆过滤器不同的是, 引入数据混淆后, 需要先确定混淆比 c, 然后在确定布隆过滤器规模 m 后, 将布隆过滤器的实际规模设置为 $(1+c)m$; 此外, 在系统初始化时, 共识节点间会共识出一个 32 字节的随机种子 seed, 共识节点根据布隆过滤器容量 n 和混淆比 c 确定加入的混淆数据量 cn, 然后根据 $\mathrm{Hash}(\mathrm{seed}||i)$ 获取第 i 个混淆数据, 并将所有的混淆数据记录到布隆过滤器和撤销列表中. 考虑到直接将所有混淆数据加入到撤销列表中会带来大量的存储开销, 因此, 本文在原来撤销列表的基础上, 额外构建短撤销列表用于记录混淆数据. 混淆数据生成后, 短撤销列表基于映射表构建, 该映射表的 key 为混淆数据的前 4 个字节, value 为序号列表, 记录了所有前 4 个字节为 key 的混淆数据对应的序号 i.

(2) 数据记录、查询与删除: 该过程与传统可验证布隆过滤器的区别主要体现在短撤销列表上, 由于引入了短撤销列表, 在判断某 key 是否存在于布隆过滤器中时, 需要同时查询原撤销列表和短撤销列表. 查询短撤销列表时, 先基于 key 的前 4 个字节进行查询, 若不存在序号列表, 则说明所查询 key 不在混淆数据列表中; 若查询存在序号列表, 则基于 seed 和列表中的序号生成所有混淆数据, 并与 key 进行比对, 如都不相同, 说明不在混淆数据列表中, 否则构造类型为 4 的不存在证明 (如下所示).

(3) 默克尔树与 MPT 树的构建: 在可验证布隆过滤器的基础上, 需要基于混淆数据生成新的默克尔树, 为了压缩默克尔树的规模, 每个叶子结点由 4 个混淆数据组成, 即第 i 个叶子节点是由序号为 $4i$, $4i+1$, $4i+2$ 和 $4i+3$ 的 4 个混淆数据相结合而成的, 树根哈希由 root_3 表示.

(4) 不存在证明的生成: 相比于可验证布隆过滤器, 增加了 type 为 4 的不存在证明, 该证明是基于短撤销列表对应的默克尔树进行构建的, 当所查询的 key 与序号为 order 的混淆数据进行匹配时, 计算获得与该混淆数据组合为同一叶子节点的其他 3 个混淆数据组成 datas, 所构造的不存在证明 P_{NE} 如下所示:

$$P_{\mathrm{NE}} := \{\mathrm{order}, \mathrm{datas}, \mathrm{hashList}, \mathrm{blockHeight}\}, \tag{14}$$

其中 hashList 是 datas 对应的默克尔哈希证明路径, blockHeight 是该证明中根哈希的区块高度.

(5) 不存在证明的验证: 针对 type 为 0, 1, 2 的不存在证明, 其验证方式与可验证布隆过滤器相同; 针对 type 为 4 的不存在证明, 先判断 datas 中的第 order 个数据是否与所查询 key 相同; 然后查询高度为 blockHeight 的区块并获取其中的根哈希 root_3, 并结合默克尔哈希路径 hashList 验证 datas 的完整性; 验证通过后, 则相信此次查询结果, 即所查询数据不存在.

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

表 2　可验证布隆过滤器及其隐私保护方案的理论分析对比

Table 2　Theoretical analysis and comparison of the verifiable Bloom filter and its privacy protection scheme

	Total size (byte)	Analysis of privacy protection		Proof size (byte)
		Leakage of Bloom filter (bit)	Recognition accuracy	
VBF	$2.25n + R$	1024	A	$140 + 32\log_2 l$
H-VBF	$6n + R$	1	A	$266 + 64\log_2 l$
C-VBF	$(2.25 + 26.25c)n + R$	1024	$(1/(1+c))^8 A$	$140 + 32\log_2(1+c)l$

(6) 布隆过滤器重构与扩容: 当在执行完第 N 个区块并确定要进行重构与扩容后, 在第 $N+1$ 个区块的共识过程中, 会重新共识新的 seed, 用于生成新的混淆数据.

6　评估与分析

本节从实验和理论两方面对可验证布隆过滤器进行评估与分析, 针对可验证布隆过滤器及其隐私保护方案的性能, 对相关方案进行实现, 并测试读、写及默克尔证明生成开销 (详见 6.1 和 6.2 小节); 针对可验证布隆过滤器及其隐私保护方案的存储开销、隐私性以及默克尔证明大小, 则以理论分析为主, 理论分析结果如表 2 所示, 详细分析见 6.3, 6.4 和 6.5 小节.

6.1　实验设置

本文采用以太坊开源代码中的 MPT 树[1)]来存储真实数据, 并结合布隆过滤器的开源代码[2)], 在此基础上基于 go 实现了可验证布隆过滤器及其隐私保护方案. 本文在实现可验证布隆过滤器时, 获取数据特征值是将数据哈希值分为 k 段, 并将 k 设置为 8, 布隆过滤器的 m/n 值设置为 12. 所有实验均在配备 Apple M1 Pro 芯片 (10 核中央处理器和 16 核图形处理器) 和 16 GB 内存的计算机上进行.

6.2　可验证布隆过滤器性能测试

本文基于 BitSet 开源库和 MPT 树开源库对可验证布隆过滤器进行实现, 并与密码累加器进行性能对比, baseline 采用的是 RSA 密码累加器的一种开源实现[3)]. 由于 RSA 累加器的每次写和默克尔证明耗时较高, 我们测试了数据规模为 50, 100, 150, 200 时, RSA 密码累加器的写以及默克尔证明开销 (可验证布隆过滤器对应操作耗时为百纳秒级别, 故不在表中显示), 测试结果如表 3 所示. RSA 密码累加器的单次写耗时为 6 ms, 远超过可验证布隆过滤器的百纳秒级别耗时; RSA 密码累加器的单次证据构建耗时为百毫秒级, 且随着数据规模的增加而增加. RSA 密码累加器的主要优势在于存储开销要求极低, 即使随着数据规模的增大, 其要求的存储量为固定常数; 然而, RSA 密码累加器在提供证明时消耗的是 CPU 资源, 即使相关辅助信息存放于内存中也无法提升证据构造性能.

针对 MPT 树和可验证布隆过滤器相结合的双通道查询模式, 我们分别测试了 50 万、100 万、200 万数据规模下, MPT 树和可验证布隆过滤器读写开销, 布隆过滤器直接存储于内存中, MPT 树用 LevelDB 进行存储, 测试结果如表 4 所示, 随着数据存储规模的扩大, 可验证布隆过滤器的读写开销和证据构造开销变化不大, 并且远小于 MPT 树的读写及证据构造开销; 在 200 万存储规模时, 可验证布隆过滤器的读写及证据构造开销分别约占 MPT 树的 1/80, 1/110 和 1/170, 可忽略不计.

1) https://github.com/ethereum/go-ethereum/tree/master/trie.
2) https://github.com/bits-and-blooms/bloom.
3) https://github.com/oleiba/RSA-accumulator.

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

表 3　不同数据存储规模下, RSA 密码累加器性能测试 (单次操作耗时 ms)

Table 3　Performance of RSA accumulator under different data storage scales (ms/op)

	Amount of stored data			
	50	100	150	200
Writing	6.19	6.13	6.22	6.31
Proving	219	443	677	913

表 4　不同数据存储规模下, MPT 树及可验证布隆过滤器读写性能对比 (10 万次操作耗时 ms)

Table 4　Performance comparison of MPT and VBF under different data storage scales (ms/100000 op)

	Amount of stored data (10^4)					
	50		100		200	
	MPT	VBF	MPT	VBF	MPT	VBF
Writing	1427	48	2271	49	6549	57
Reading	586	35	724	38	3254	40
Proving	2458	44	3242	48	8910	52

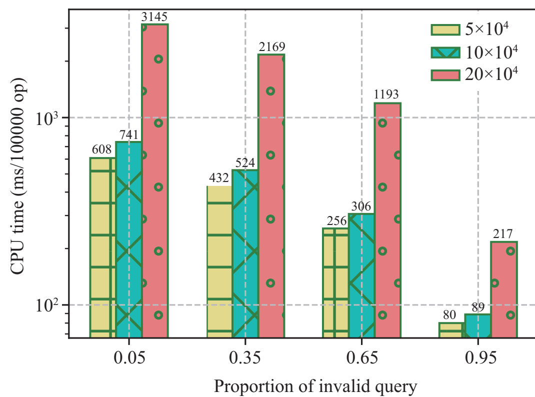

图 12　双通道模式下数据查询性能

Figure 12　The CPU time of the query operation under different data storage scales

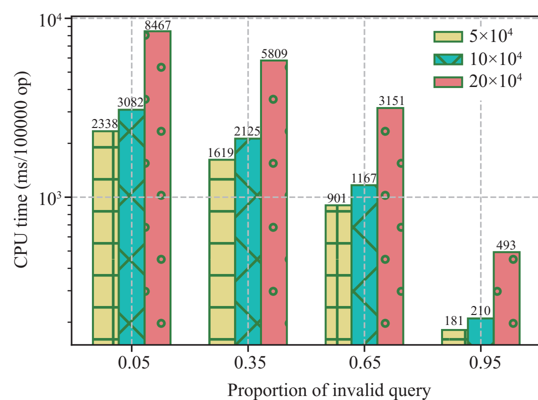

图 13　双通道模式下默克尔证明构造性能

Figure 13　The CPU time of the proof construction operation under different data storage scales

此外, 我们测试 MPT 树和可验证布隆过滤器相结合的双通道查询方案性能, 分别在 50 万、100 万和 200 万数据规模存储下进行 10 万次数据查询以及默克尔证明构造, 并分别控制无效查询比例为 5%, 35%, 65% 和 95%, 得到的查询性能和默克尔证明性能分别如图 12 和 13 所示, 针对无效查询比例越高的场景, 引入可验证布隆过滤器过滤无效查询请求后, 对整体查询效率及证据构造效率提升越大.

6.3　隐私保护方案性能测试

在可验证布隆过滤器 (VBF) 的基础上, 对隐秘的可验证布隆过滤器 (H-VBF) 以及数据混淆方案 (C-VBF) 进行实现, 并分别在 50 万、100 万和 200 万数据规模存储下进行 10 万次数据查询以及默克尔证明构造, 对性能进行分析. 表 5 所示为几种方案的读性能及默克尔证明构造性能对比, H-VBF

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

表 5 不同数据存储规模下, 可验证布隆过滤器及其隐私保护方案性能对比 (10 万次操作耗时 ms)

Table 5 Performance comparison of VBF and its privacy protection scheme under different data storage scales (ms/100000 op)

	Amount of stored data (10^4)					
	50		100		200	
	Reading	Proving	Reading	Proving	Reading	Proving
MPT	586	2458	724	3243	3254	8910
VBF	35	44	38	48	40	52
C-VBF	51	61	53	50	54	50
H-VBF	37	91	41	98	39	111

因为需要针对隐秘的布隆过滤器和随机二进制矩阵分别构造默克尔证明, 因此证据构造开销提升近一倍, 读开销则无明显变化; C-VBF 要求插入混淆数据, 在查询过程中需要额外查询短撤销列表, 因此导致查询性能略微降低, 而默克尔证据构造性能变化不大. 此外, 尽管隐私保护方案会带来一定的性能损失, 与 MPT 树的性能进行对比, 开销依然可以忽略不计.

6.4 存储开销分析

相比于可验证布隆过滤器, 两种隐私保护方案都带来了更多的存储开销, 本小节对可验证布隆过滤器及其隐私保护方案的存储开销进行分析, 并假设存储的数据量规模为 n, 布隆过滤器的行数为 l, 列数为 1024, 每个哈希值的大小为 32 字节, 在分析过程中, 我们根据表 1 选取布隆过滤器参数为 $m/n=12$, $k=8$; 此外, 在分析过程中, 假设实际数据撤销列表的开销为 R, 不做具体分析.

(1) 可验证布隆过滤器: 当直接使用可验证布隆过滤器时, 在撤销列表开销 R 的基础上, 需要额外存储布隆过滤器和对应的默克尔树. 其中布隆过滤器大小为 $12n$ 比特, 即 $1.5n$ 字节; 布隆过滤器行数 $l=12n/1024$, 对每一行构造默克尔树, 相应的默克尔树则近似包含 $2l$ 个节点, 即 $3n/128$ 个哈希值, 存储大小为 $3n/4$ 字节. 因此, 加上实际的数据撤销列表开销 R, 总的存储开销为 $(2.25n+R)$ 字节.

(2) 隐秘的可验证布隆过滤器: 相比于可验证布隆过滤器, HBF 需要额外存储与布隆过滤器相同规模的随机二进制矩阵 RBM, 以及隐秘的布隆过滤器 HBF, 存储开销为 $3n$ 字节; 此外, 该方案需要额外基于 RBM 构造默克尔树, 存储开销为 $3n/4$ 字节. 因此, 加上实际的数据撤销列表开销 R, 总的存储开销为 $(6n+R)$ 字节.

(3) 数据混淆: 该方案的存储开销与混淆比 c 相关, 由于布隆过滤器中添加了 cn 的混淆数据, 实际存储规模为 n 的布隆过滤器则需要 $(1+c)\times 1.5n$ 字节; 短撤销列表中, 每个混淆数据的存储开销为 8 字节, 存储开销为 $8cn$ 字节; 基于短撤销列表构建的默克尔树中, 包含 $cn/4$ 个叶子节点, 因此近似包含 $cn/2$ 个哈希值, 存储开销为 $16cn$ 字节. 因此, 总存储开销为 $(2.25n+26.25cn+R)$ 字节.

6.5 隐私性分析

本小节对可验证布隆过滤器及其隐私保护方案的隐私性进行分析, 假设攻击者的主要目的是尽可能地获取布隆过滤器内容, 并测试某 key 值是否大概率存在于系统当中. 例如在智慧医疗等数据隐私性强的场景, 攻击者可以大概率确认某人是否在医院就诊或患有某项疾病, 从而危害用户隐私. 在分析过程中, 主要考虑单次 “不存在证明” 验证过程中泄露的数据量, 以及根据获取的布隆过滤器测试某 key 存在时, 其真实存在的概率. 假设布隆过滤器的数据容量为 n.

(1) 可验证布隆过滤器: 每次验证 “不存在证明” 时, 都需要包含布隆过滤器的一行, 因此泄露的

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

数据量等于总列数, 即 1024 比特; 由于布隆过滤器未经过任何隐私保护措施, 攻击者基于所获取的布隆过滤器判断某数据存在时, 其准确率与布隆过滤器自身的准确率相同, 设为 A.

(2) 隐秘的可验证布隆过滤器: 每次验证 "不存在证明" 时, 用户只能恢复出一位置零位, 因此隐私泄露量为 1 比特; 然而, 当用户进行多次 "不存在证明" 验证时, 能获得更多比特的布隆过滤器数据位, 为了避免用户通过累计多次查询获得实际布隆过滤器内容, 可以定期重新生成随机二进制矩阵. 与可验证布隆过滤器相同, 用户根据泄露的布隆过滤器内容判断某数据存在时, 其准确率与布隆过滤器自身的准确率相同, 为 A.

(3) 数据混淆: 该方案与可验证布隆过滤器一样, 每次提供 "不存在证明" 都会泄露 1024 比特的数据量, 但攻击者所看到的布隆过滤器置 1 位存在 $c/(1+c)$ 的概率是由混淆数据造成的, 由于需要同时判断 8 位为 1 才能大概率确定某 key 存在于数据库中, 因此准确率为 $(1/(1+c))^8A$.

6.6 默克尔证明规模分析

本小节对可验证布隆过滤器及其隐私保护方案中默克尔证明的大小进行分析, 基于数据撤销列表 MPT 树构建默克尔证明的概率较低, 因此我们只分析基于布隆过滤器所构造的证明大小.

(1) 可验证布隆过滤器: 默克尔证明包含一行布隆过滤器, 大小为 128 字节; 默克尔哈希路径, 包含的哈希个数为基于默克尔树的高度, 即 $32\log_2 l$; 行列号以及区块链高度, 共 12 字节. 因此, 总开销为 $140+32\log_2 l$ 字节.

(2) 隐秘的可验证布隆过滤器: 相比于可验证布隆过滤器, 其默克尔证明多了一行随机二进制矩阵以及相应的默克尔哈希路径, 因此总开销为 $128+128+12+64\log_2 l=266+64\log_2 l$ 字节.

(3) 数据混淆: 默克尔证明内容与可验证布隆过滤器一致, 但布隆过滤器行数增加了 cl, 因此, 总开销为 $140+32\log_2(1+c)l$ 字节.

7 总结

本文针对基于区块链的 "真实存储" 应用场景, 提出了可验证布隆过滤器的一种构建方法, 能够加速过滤区块链中的无效查询请求, 并针对 "不存在" 这一结果生成证明. 针对证明过程中可能存在的隐私泄露问题, 提出了隐秘的可验证布隆过滤器和数据混淆两种隐私保护方案. 我们对可验证布隆过滤器及其隐私保护方案进行实现, 并对其性能进行测试, 在 200 万存储规模时, 可验证布隆过滤器的读写及证据构造开销分别约占 MPT 树的 1/80, 1/110 和 1/170, 可忽略不计; 此外, 引入可验证布隆过滤器过滤无效查询请求后, 对整体查询效果提升较大, 且随着数据存储规模的增大, 提升效果更明显.

参考文献

1 Nakamoto S. Bitcoin: a peer-to-peer electronic cash system. White Paper, 2008. https://bitcoin.org/bitcoin.pdf
2 Chen J, Yao S, Yuan Q, et al. CertChain: public and efficient certificate audit based on blockchain for TLS connections. In: Proceedings of IEEE Conference on Computer Communications, 2018
3 Kubilay M Y, Kiraz M S, Mantar H A. CertLedger: a new PKI model with certificate transparency based on blockchain. Comput Security, 2019, 85: 333–352
4 Shen M, Liu H, Zhu L, et al. Blockchain-assisted secure device authentication for cross-domain industrial IoT. IEEE J Sel Areas Commun, 2020, 38: 942–954
5 Adja Y C E, Hammi B, Serhrouchni A, et al. A blockchain-based certificate revocation management and status verification system. Comput Security, 2021, 104: 102209

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

6 Truong N B, Sun K, Lee G M, et al. GDPR-compliant personal data management: a blockchain-based solution. IEEE Trans Inform Forensic Secur, 2019, 15: 1746–1761
7 Xiong Z, Zhang Y, Luong N C, et al. The best of both worlds: a general architecture for data management in blockchain-enabled Internet-of-Things. IEEE Network, 2020, 34: 166–173
8 Shafagh H, Burkhalter L, Hithnawi A, et al. Towards blockchain-based auditable storage and sharing of IoT data. In: Proceedings of the 2017 on Cloud Computing Security Workshop, 2017. 45–50
9 Hari A, Lakshman T V. The Internet Blockchain: a distributed, tamper-resistant transaction framework for the Internet. In: Proceedings of the 15th ACM Workshop on Hot Topics in Networks, 2016. 204–210
10 Xing Q, Wang B, Wang X. POSTER: BGPCoin: a trustworthy blockchain-based resource management solution for BGP security. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2017. 2591–2593
11 Liu B-Y, Yang F, Ren S-S, et al. Decentralized internet infrastructure. Telecommun Sci, 2019, 35: 74–87 [刘冰洋, 杨飞, 任首首, 等. 去中心化互联网基础设施. 电信科学, 2019, 35: 74–87]
12 Xu K, Ling S-T, Li Q, et al. Research progress of network security architecture and key technologies based on blockchain. Chin J Comput 2021, 44: 55–83 [徐恪, 凌思通, 李琦, 等. 基于区块链的网络安全体系结构与关键技术研究进展. 计算机学报, 2021, 44: 55–83]
13 Benet J, Greco N. Filecoin: a decentralized storage network. Protocol Labs, 2018. https: //filecoin.io/filecoin.pdf
14 Peng Y, Du M, Li F, et al. FalconDB: blockchain-based collaborative database. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2020. 637–652
15 Li M, Zhu J, Zhang T, et al. Bringing decentralized search to decentralized services. In: Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21), 2021. 331–347
16 Wu H, Peng Z, Guo S, et al. VQL: efficient and verifiable cloud query services for blockchain systems. IEEE Trans Parallel Distrib Syst, 2021, 33: 1393–1406
17 Zhang Y, Katz J, Papamanthou C. IntegriDB: verifiable SQL for outsourced databases. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015. 1480–1491
18 Ruan P, Chen G, Dinh T T A, et al. Fine-grained, secure and efficient data provenance on blockchain systems. Proc VLDB Endow, 2019, 12: 975–988
19 Xu C, Zhang C, Xu J. vChain: enabling verifiable boolean range queries over blockchain databases. In: Proceedings of the International Conference on Management of Data, 2019. 141–158
20 Zhang C, Xu C, Wang H, et al. Authenticated keyword search in scalable hybrid-storage blockchains. In: Proceedings of IEEE 37th International Conference on Data Engineering (ICDE), 2021. 996–1007
21 Chang F, Dean J, Ghemawat S, et al. Bigtable: a distributed storage system for structured data. ACM Trans Comput Syst, 2008, 26: 1–26
22 Lakshman A, Malik P. Cassandra: a decentralized structured storage system. SIGOPS Oper Syst Rev, 2010, 44: 35–40
23 Dong W, Douglis F, Li K, et al. Tradeoffs in scalable data routing for deduplication clusters. In: Proceedings of the 9th USENIX Conference on File and Storage Technologies, 2011. 11: 15–29
24 Debnath B, Sengupta S, Li J, et al. BloomFlash: Bloom filter on flash-based storage. In: Proceedings of the 31st International Conference on Distributed Computing Systems, 2011. 635–644
25 Li Y, Tian C, Guo F, et al. ElasticBF: elastic Bloom filter with hotness awareness for boosting read performance in large key-value stores. In: Proceedings of USENIX Annual Technical Conference, 2019. 739–752
26 Christen P, Ranbaduge T, Vatsalan D, et al. Precise and fast cryptanalysis for Bloom filter based privacy-preserving record linkage. IEEE Trans Knowl Data Eng, 2018, 31: 2164–2177
27 Vidanage A, Ranbaduge T, Christen P, et al. Efficient pattern mining based cryptanalysis for privacy-preserving record linkage. In: Proceedings of IEEE 35th International Conference on Data Engineering (ICDE), 2019. 1698–1701
28 Wood G. Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper, 2014, 151: 1–32
29 Jia M, He K, Chen J, et al. PROCESS: privacy-preserving on-chain certificate status service. In: Proceedings of IEEE Conference on Computer Communications, 2021. 1–10
30 Merkle R C. A digital signature based on a conventional encryption function. In: Proceedings of Conference on the Theory and Application of Cryptographic Techniques. Berlin: Springer, 1987. 369–378

https://engine.scichina.com/doi/10.1360/SSI-2023-0048

31 Thotakura V, Ramkumar M. Minimal trusted computing base for MANET nodes. In: Proceedings of IEEE 6th International Conference on Wireless and Mobile Computing, Networking and Communications, 2010. 91–99
32 Bailey B, Sankagiri S. Merkle trees optimized for stateless clients in bitcoin. In: Proceedings of International Workshops on Financial Cryptography and Data Security, 2021. 451–466
33 Boneh D, B́unz B, Fisch B. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In: Proceedings of the 39th Annual International Cryptology Conference, 2019. 561–586

Verifiable Bloom filter (VBF): accelerate the query and proof of nonexistent data in a blockchain

Songsong XU[1,2], Xiaobing GUO[3] & Ke XU[1,2*]

1. *Department of Computer Science and Technology, Tsinghua University, Beijing* 100084*, China*;
2. *Beijing National Research Center for Information Science and Technology, Beijing* 100084*, China*;
3. *The Lenovo Research, Beijing* 100085*, China*
* Corresponding author. E-mail: xuke@tsinghua.edu.cn

Abstract The traditional blockchain-based “true-storage” system ignores the provision of “nonexistent proof” when filtering “invalid query requests”, and malicious nodes can launch denial of service attacks on designated users. This paper proposes a method for building a verifiable Bloom filter, by which nodes can quickly filter invalid query requests and provide valid evidence proving the data does not exist. In addition, this paper proposes two ways to solve the privacy leakage problem in the proof process: a “hidden verifiable Bloom filter” and “data confusion.” The former method ensures that each “nonexistent proof” only leaks one bit of the Bloom filter, reducing the data leakage in the proof process. The latter method reduces the accuracy for users inferring real content from the leaking Bloom filter. The experimental data show that when the proportion of invalid query requests is 35%, the read performance can be improved by approximately 30%; and when this proportion is 95%, the improvement can be more than tenfold.

Keywords blockchain, verifiable Bloom filter, authenticated data structure, nonexistence proof, privacy preserving

https://engine.scichina.com/doi/10.1360/SSI-2023-0048