计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(12):87−94 [doi: 10.15888/j.cnki.csa.008826]

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

基于可验证延迟函数的 Hyperledger Fabric 背书策略优化方案①

陈　楠[1,2], 周赵斌[1,2], 陈志德[1,2]

[1](福建师范大学 计算机与网络空间安全学院, 福州 350007)
[2](福建省网络安全与密码技术重点实验室 (福建师范大学), 福州 350007)
通信作者: 周赵斌, E-mail: 1334672047@qq.com

摘　要: 为了解决 Hyperledger Fabric 使用固定背书节点进行背书而引发的安全问题, 提出了一种基于可验证延迟函数的 Hyperledger Fabric 背书策略优化方案. 利用可验证延迟函数不可并行计算以及可高效验证的特点, 设计了匿名化、随机化选取背书节点的 Fabric 交易模型, 增强了 Fabric 交易背书的安全性. 通过实验对比优化方案与原始方案, 验证了优化方案在增强安全性的同时, 在效率上也有一定的提升.

关键词: Hyperledger Fabric; 背书策略; 可验证延迟函数; 交易模型

引用格式: 陈楠,周赵斌,陈志德.基于可验证延迟函数的 Hyperledger Fabric 背书策略优化方案.计算机系统应用,2022,31(12):87–94. http://www.c-s-a.org.cn/1003-3254/8826.html

Optimization Scheme of Hyperledger Fabric Endorsement Policy Based on Verifiable Delay Function

CHEN Nan[1,2], ZHOU Zhao-Bin[1,2], CHEN Zhi-De[1,2]

[1](College of Computer and Cyberspace Security, Fujian Normal University, Fuzhou 350007, China)
[2](Fujian Provincial Key Laboratory of Network Security and Cryptology (Fujian Normal University), Fuzhou 350007, China)

Abstract: To solve the security problem of Hyperledger Fabric caused by the use of fixed endorsement nodes for endorsement, this study proposes an optimization scheme based on a verifiable delay function for the endorsement policy of Hyperledger Fabric. Considering that the verifiable delay function cannot be calculated in parallel but can be efficiently verified, a Fabric transaction model that anonymously and randomly selects endorsement nodes is designed to enhance the security of Fabric transaction endorsement. The experimental comparison between the optimized scheme and the original one verifies that the optimized scheme not only enhances security but also improves efficiency.

Key words: Hyperledger Fabric; endorsement policy; verifiable delay function; transaction model

区块链技术是结合了密码学、数学、分布式计算等一系列学科的跨学科技术, 被广泛应用于农业、金融业、工业等现实场景中, 对现代经济社会的发展作出了重要贡献[1]. 在区块链技术被提出之前, 数据往往被存储于少量的中心化机构中, 一旦这些数据集中存储的节点被攻击, 就会造成数据泄漏, 带来相当大的安全隐患. 区块链技术的核心在于去中心化, 数据被分布式地存储于链上的各个节点当中, 这在一定程度上解决了数据过于集中, 安全隐患大的问题.

虽然去中心化的区块链技术解决了数据过于集中易被攻击的安全问题, 但也引发了一个新的问题, 即如何在一个去中心化、弱信任的分布式系统中让各个节点对系统的某一行为产生共识, 让系统节点产生共识的机制也被称为共识机制[2–4]. 共识机制是区块链技术

① 基金项目: 福建省自然科学基金 (2020J01171); 国家自然科学基金 (61841701)
收稿时间: 2022-03-17; 修改时间: 2022-04-14; 采用时间: 2022-04-29; csa 在线出版时间: 2022-07-25

的核心, 它在保证区块链上的数据安全可靠的同时也要兼顾效率. 随着区块链技术的发展, 诸如工作量证明 (proof of work, POW)、股权证明 (proof of stake, POS)、委任权益证明 (delegated proof of stake, DPOS)、实用拜占庭容错算法 (practical Byzantine fault tolerance, PBFT)、授权拜占庭容错算法 (delegated BFT, dBFT) 等共识机制被相继提出[3,4].

联盟链是在区块链技术基础上衍生的一种新的实用型分布式处理数据技术, 而 Hyperledger Fabric 正是在联盟链技术的基础上发展起来的. 它由 IBM、DAH 等企业提交到 Linux 基金会[5], 现在已经发展成了一个成熟的商业级区块链平台. Hyperledger Fabric 将系统中的节点分为客户节点、对等节点以及背书节点. 客户节点负责提交相关交易给背书节点, 背书节点接收到交易之后进行检查, 检查无误之后对交易进行背书签名, 对等节点则检查交易的一致性问题并将交易打包成块提交到区块链中. 不同于传统的区块链共识机制, Hyperledger Fabric 对系统节点的功能进行了划分, 从而通过“背书-排序-验证”的方式使得系统节点形成共识[5].

背书提高了交易的效率, 排序保证了交易的一致性, 验证保证了交易的合法性, 这种将系统各个部分解耦的方式为 Hyperledger Fabric 提供了灵活的、可插拔式的功能选项, 使其广泛适用于各个场景. Hyperledger Fabric 中的背书节点[6,7] 承担了所有客户提交交易的背书工作, 中心化处理交易背书提高了交易的处理效率, 但同时也会引发安全问题. 因为背书节点身份的公开性, 当敌手想要攻击整个 Hyperledger Fabric 系统时, 只需要攻击相应的背书节点即可达到控制整个链上交易的目的. 为了增强背书节点安全性, Mazumdar 等人[8] 设计了利用环签名的方式来选取背书节点; 孟昊同等人[4] 引入了可验证随机函数, 通过随机化选取背书节点的方式来增强背书节点的安全性; 张龙静等人[9] 提出了根据当前物理资源动态调整背书节点的机制.

以上方案虽然都通过动态选取背书节点的方式将背书节点匿名化, 但没有考虑存在硬件加速计算的情况, 敌手可以通过增加算力的方式来达到伪造背书节点身份的目的. 可验证延迟函数是一类能够产生随机数且运算过程不能通过增加算力的方式加快的函数, 为了解决 Hyperledger Fabric 使用固定背书节点容易引发安全问题以及抵御并行加速计算, 本文在“背书-排序-验证”共识机制的基础上, 引入可验证延迟函数, 提出了 Hyperledger Fabric 共识机制的优化方案, 具体方案如下.

(1) 将原先固定的背书节点改为候选背书节点集合, 引入可验证延迟函数, 利用可验证延迟函数输出随机数的特性, 通过设置阈值的方式从候选的背书节点集中动态选择背书节点对交易进行背书.

(2) 因为可验证延迟函数不可并行加速计算的特性, 敌手在事先不知道密钥的情况下无法通过并行计算的方式伪造成被选择的背书节点.

(3) 当背书节点对现有交易进行背书之后, 其背书节点的身份被释放, 该节点重新回到候选背书集合中, 可对接下来的交易进行背书, 提高交易的处理效率.

1 相关知识

1.1 Hyperledger Fabric

Hyperledger Fabric 是由 IBM、DAH 等企业[5] 提交到 Linux 开源基金会的联盟链实体应用, 不同于公有链的完全去中心化, 它需要相应的中心化节点对提交到链上的交易进行认证处理. Hyperledger Fabric 由多个功能模块组成[7], 各个模块之间彼此独立并通过协同工作的方式实现交易的打包成块、上链. Hyperledger Fabric 的系统架构如图 1 所示.

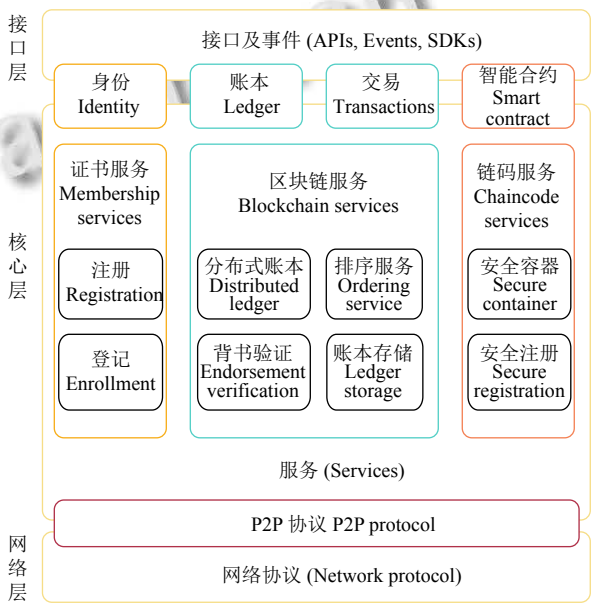

图 1 Hyperledger Fabric 系统架构

共识机制是保证交易能被正确执行并且提交到链上的关键. 区别于传统的公有链共识机制, Hyperledger

Fabric 采用“背书-排序-验证”的方式对链上的交易达成共识. Hyperledger Fabric 的原始交易流程如图 2 所示, 具体细节如下:

(1) 交易提案$tx_pros < ch, p, endor_{\text{policy}} >$: 当用户想要进行交易时, 通过客户节点发起交易提案tx_pros, 交易提案包括一系列链码 ch、相关的参数 p 以及背书策略$endor_{\text{policy}}$. 交易提案被创建之后, 客户节点检查相关的背书策略, 并依据背书策略将交易提案发送给需要对该交易进行背书的背书节点.

(2) 交易背书$endor_tx < tx_pros, sig, pa >$: 当交易提案$tx_pros$被提交到符合背书策略[10] 的背书节点之后, 背书节点检查相应的交易签名sig以及其他参数pa. 检查无误之后背书节点执行交易逻辑, 产生读写集$set < write, read >$, 并将结果签名$sig < tx_pros, set >$, 之后发送给客户节点.

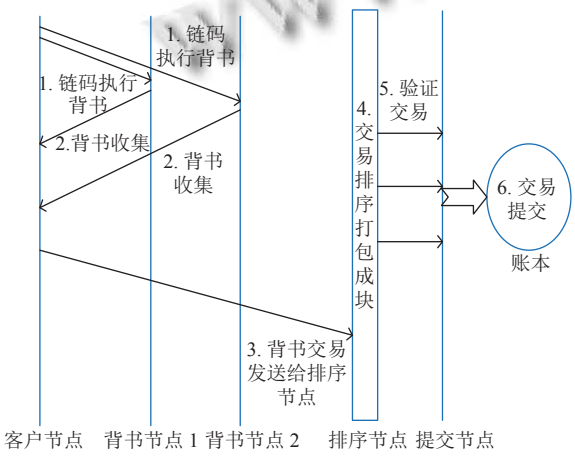

图 2 Hyperledger Fabric 原始交易流程

(3) 交易排序$order < tx_1, tx_2, \cdots, tx_n >$: 客户节点检查交易背书的签名$sig < tx_pros, set >$, 如果检查结果出现错误, 则终止交易, 如果检查结果正确则将交易发送给排序节点. 排序节点接收来自客户节点的交易之后, 检查交易的时间顺序是否符合逻辑, 即检查交易的一致性. 排序节点将符合一致性的交易排序$order < tx_1, tx_2, \cdots, tx_n >$并打包成块$block < tx_1, tx_2, \cdots, tx_n >$, 进行全网广播之后发送给提交节点;

(4) 交易提交$submit < b1, b2, \cdots, bn >$: 打包成块的交易$block < tx_1, tx_2, \cdots, tx_n >$被发送给提交节点之后, 提交节点对交易进行验证$verify < b_1, b_2, \cdots, b_n >$. 提交节点首先验证交易背书是否符合背书策略, 之后检查执行交易生成的读写集的有效性. 通过验证的交易被提交节点提交到账本中, 进行交易的存储.

1.2 可验证延迟函数

可验证延迟函数 (verifiable delay function, VDF) 是由 Boneh 等人[11] 最早提出来的, 它是一类数学函数, 当输入相应的数值之后, 需要消耗一定的时间来计算出结果. 计算所需的时间不能通过简单的增加算力的方式来缩短, 即不可并行加速运算. 该类函数还必须保证计算结果的可验证性. 为了达到不可并行加速以及可验证的目的, 可验证延迟函数由如下算法组成:

(1) $Setup(\lambda, t) \rightarrow (ek, vk)$; 由于可验证延迟函数初始化算法, 以安全参数以及时间参数t为输入, 输出一个用于计算的参数ek以及用于验证的参数vk, 输出的参数都是公开透明的, 所以也被称为公共参数.

(2) $Eval(ek, x) \rightarrow (y, \pi)$: 计算算法, 接受输入参数ek以及具体值x作为输入, 得到结果输出y以及证明π.

(3) $Verify(vk, x, y, \pi) \rightarrow (accept, reject)$: 验证算法, 接收一系列的输入, 输出结果验证通过 ($accept$) 或者验证失败 ($reject$).

可验证延迟函数的运行流程如图 3 所示.

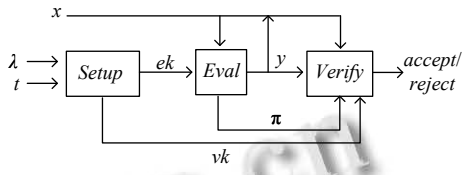

图 3 可验证延迟函数运行流程

可验证延迟函数的概念被提出之后, 许多有效的构造也被相继提出. Wesolowski[12] 基于未知阶的群构造了陷门可验证延迟函数, 该方案构造了可以加速计算的陷门计算方法; Pietrzak 等人[13] 基于 Rivest-Shamir-Wagner 的 Time-lock 难题构造了一个简单的可验证延迟函数; Ephraim 等人[14] 提出了连续可验证延迟函数, 通过迭代可验证延迟函数子过程的每一个输出来进行过程验证, 提高了的输出验证效率; Döttling 等人[15] 基于自举定理构造了结构紧凑的可验证延迟函数, 该类函数具有更高的时间效率; Rotem[16] 提出了批量处理可验证延迟函数中幂运算的技术, 降低了函数的通信复杂度.

可验证延迟函数不可并行加速以及验证高效的特性可以用来增强公共可验证随机数的安全性, 是一个十分强大的工具, 研究人员也将其功能与区块链技术结合起来. 可验证延迟函数在 Chia 区块链的架构中扮

演了重要的角色, 以太坊基金会和 Protocol Labs 也将其列为下一代区块链平台的核心角色.

2 方案设计

本文在 Hyperledger Fabric 的原始架构基础上引入了可验证延迟函数, 将原始方案中固定的背书节点改为从背书节点候选集中随机抽取背书节点.

2.1 可验证延迟函数的设计

可验证延迟函数的设计关键是要保证匿名的攻击者在未持有背书节点私钥的情况下, 要想得出正确的运算结果以及身份证明, 需要进行大量的连续计算, 这种计算是不能并行加速的, 即不能靠增加计算资源来缩短求解结果的时间. 持有私钥的背书节点可以高效地实现计算求解, 得出正确的计算结果和证明. 本文基于 Wesolowski[12] 的工作来实现可验证延迟函数的设计. 可验证延迟函数总体包含 4 个部分: 用于生成密钥对的密钥生成算法$keygen()\rightarrow(sk,pk)$; 陷门计算算法$trapdoor_{sk}(x,t)\rightarrow(y,\pi)$; 未持有私钥情况下要想求解结果的不可并行性计算算法$verify(x,t)\rightarrow(y,\pi)$及客户端用于验证背书节点身份的验证算法$verify_{pk}(x,y,\pi,t)\rightarrow$(true, false).

2.1.1 密钥生成算法*keygen*()

本文的可验证延迟函数基于未知阶的 RSA 群, 密钥生成算法由式 (1)、式 (2) 得出:

$$pk = N = pq \tag{1}$$

$$sk = (p-1)(q-1) \tag{2}$$

其中, p,q是大素数.

2.1.2 陷门计算算法$trapdoor_{sk}()$

拥有私钥sk的用户可以通过陷门计算算法$trapdoor_{sk}()$快速得到结果y以及用于证明自己身份的证明π. 具体的计算过程如下.

(1) 计算相关参数g,e, 具体计算如算法 1 中①和②所示. 其中G表示未知阶的群, H_G表示输出结果是群G中元素的哈希函数, x是输入, sk是私钥, t是时间参数.

(2) 计算返回给验证者的结果y, 具体计算如算法 1 的③所示. 因为用户持有私钥sk, 可以进行快速运算, 详细计算过程如式 (3):

$$\begin{aligned} y = g^{2^t} &= g^{sk\cdot n+2^t \bmod sk} = g^{sk\cdot n}\cdot g^{2^t \bmod sk} \\ &= e_G\cdot g^{2^t \bmod sk} = g^{2^t \bmod sk} = g^e \end{aligned} \tag{3}$$

其中, n是一个正整数, e_G是群G中的生成元.

(3) 生成证明π, 具体计算如算法 1 的④–⑦所示, 其中, H_{prime}是输出结果为素数的哈希函数.

算法 1. 陷门计算算法$trapdoor_{sk}()$

输入: sk, x, t
输出: y, π

Begin:
Compute parameters g and e:
① $g\leftarrow H_G(x)\in G$
② $e\leftarrow 2^t \bmod sk$
Compute result y:
③ $y\leftarrow g^e$
Compute prove π:
④ $l\leftarrow H_{\text{prime}}(g,y)$
⑤ $r\leftarrow 2^t \bmod l$
⑥ $q\leftarrow (2^t-r)l^{-1} \bmod sk$
⑦ $\pi\leftarrow g^q$
return(y,π)
End

2.1.3 计算算法*eval*()

没有私钥sk的情况下, 恶意用户要想生成有效的结果y以及证明π, 需要以下计算过程.

(1) 计算参数g, 具体计算如算法 2 的①所示, 其中G代表未知阶的群, x表示输入, H_G是输出结果为群G中元素的哈希函数.

(2) 计算返回给验证者的结果y, 计算算法如算法 2 的②所示, 其中t代表时间参数.

(3) 生成证明π, 具体计算如算法 2 的③, ④所示. 其中H_{prime}表示输出结果为素数的哈希函数.

算法 2. 计算算法*eval*()

输入: x, t
输出: y, π

Begin:
Compute parameter g:
① $g\leftarrow H_G(x)\in G$
Compute result y:
② $y\leftarrow g^{2^t}$
Compute prove π:
③ $l\leftarrow H_{\text{prime}}(g,y)$
④ $\pi\leftarrow g^{\lfloor 2^t/l\rfloor}$
return(y,π)
End

在算法 2 中, 计算结果由等式$y=g^{2^t}$给出, 在没有私钥sk的情况下, 时间复杂度为$O(t)$. 随着t的增大, 计算给出结果的困难性也随之增加. 这在一定程度上增加了没有私钥的攻击者伪造计算结果以及证明的困难

度, 提高了安全性.

2.1.4 身份验证算法*verify*()

身份验证算法通过验证结果y以及证明π是否匹配来判断背书节点是否为恶意用户, 具体的步骤如下.

(1) 计算参数g, l, r, 具体计算如算法 3 的①–③所示, 其中G代表未知阶的群, x表示输入, H_G是输出结果为群G中元素的哈希函数, H_{prime}代表输出结果为素数的哈希函数, 时间参数t.

(2) 验证结果: 如果$\pi^l g^r = y$则验证通过, 返回true. 否则返回false. 在身份验证的过程中, 有式 (4):

$$\pi^l g^r = (g^{\lfloor 2^t/l \rfloor})^l \cdot g^{2^t \bmod l} = g^{\lfloor 2^t/l \rfloor l+(2^t \bmod l)} = g^{2^t} = y \quad (4)$$

只有在计算结果y以及证明π都正确的情况下, 等式才会成立.

算法 3. 身份验证算法*verify*()

输入: y, x, t, π

输出: true or false

Begin:

Compute parameters g, l, r:

① $g \leftarrow H_G(x) \in G$

② $l \leftarrow H_{\text{prime}}(g,y)$

③ $r \leftarrow 2^t \bmod l$

if $\pi^l g^r = y$

return true

else

return false

End

2.2 优化交易流程

改进方案基于可验证延迟函数, 主要是从下列两个方面优化了交易流程.

(1) 原始方案中的背书节点是固定且公开的, 客户端需要根据背书策略将交易提交给特定的背书节点进行背书确认. 由于背书是交易能否顺利进行的关键, 外部攻击者为了达到破坏或者安插不法交易的目的, 势必会攻击相应的背书节点, 而背书节点身份的透明又加剧了其遭受攻击的可能性. 本方案匿名化了背书节点, 候选背书节点集只有经过可验证延迟函数的计算后才能对交易进行背书. 由于背书节点的选取是随机的, 攻击者要想控制交易, 必须攻击候选背书节点集, 这增加了攻击者的攻击成本, 提高了背书节点的安全性.

(2) 在原始方案中, 被背书策略指定的背书节点才能对交易进行背书. 在同一时间内, 如果被指定的背书节点都有交易要处理, 那么当有新交易被客户端提交给背书节点后, 只能处于等待阶段. 与此同时, 未被背书策略指定的背书节点处于空闲状态, 这造成了资源的浪费. 优化方案采取随机的方式抽取背书节点, 且候选背书集中的每个背书节点被抽取的概率都是相同的. 在同一时间下, 不同的交易可以被提交给不同的背书节点进行背书. 随着交易数量的增加, 繁忙的节点将不再被抽取为背书节点, 而空闲的背书节点被指定为交易背书, 这种随机方式提高了候选背书节点的利用率.

优化后的 Hyperledger Fabric 交易流程如图 4 所示.

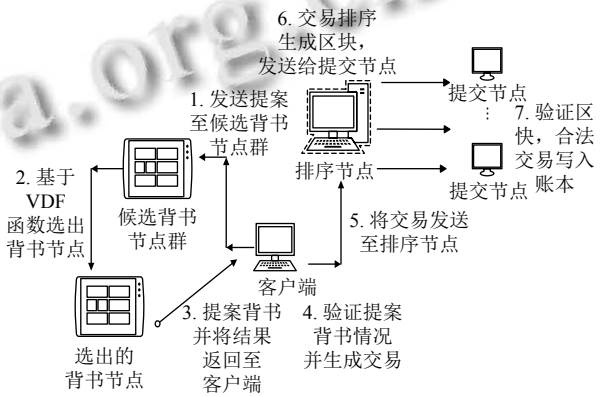

图 4 优化后的 Hyperledger Fabric 交易流程

(1) 客户端发起提案$tx_pros < ch, p, x, t, sig >$, 提案包括要调用的交易业务逻辑代码链码$ch$、相关参数$p$、候选背书节点抽取算法的输入$x$以及背书时间参数$t$. 客户端对要背书的交易签名后将其发送给候选背书节点集.

(2) 候选背书节点集收到来自客户端的需要背书的交易之后, 首先验证其签名sig的合法性. 验证签名过后, 候选背书节点运算抽取算法:

$$trapdoor_{sk}(x,t,sk) \rightarrow (y,\pi) \quad (5)$$

其中, y是运算得出的计算结果, π是生成的身份证明. 由于时间参数t的限制, 只有拥有私钥sk的背书节点才能快速计算得出结果以及相关证明, 而攻击者因为没有私钥sk, 在时间的限制下, 即使运算得出了正确结果也不能及时返回背书的交易给客户端.

(3) 随机选取的背书节点运算完抽取算法之后, 对相关交易进行背书, 得到读写结果集合$read_write$及以及背书提案tx_pros_{endor}. 将计算结果y, 证明π, 读写集$read_write$以及背书提案tx_pros_{endor}打包发送给客户端进行验证.

(4) 客户端收到背书节点发来的结果集$< y, \pi, read_write, tx_pros_{endor} >$之后, 进行背书结果的验证,

验证算法如式 (6):

$$verify_{pk}(x,y,\pi,t) \tag{6}$$

如果验证结果为true, 说明背书节点的身份合法, 该背书节点背书的交易有效; 如果验证结果为false, 说明该背书节点是恶意的, 那么需要抛弃该背书交易, 将交易重新提交给候选背书节点集进行背书.

(5) 背书有效的交易被客户端提交给排序节点进行排序, 排序节点接受到交易之后, 将不同的交易排序打包成块并进行签名, 之后将该区块全网广播.

(6) 提交节点监听网络上的交易区块, 当有新的交易区块被发送到提交节点时, 提交节点首先验证排序节点的签名合法性, 其次验证读写集的合法性. 通过两项验证之后的交易区块被提交到区块链上并更新账本, 之后提交节点进行全网广播

3 安全性分析

上述改进方案引入了可验证延迟函数 (VDF) 来处理背书节点的选取问题. 本节将从两个方面来分析方案的安全性: 其一是可验证延迟函数的可靠性; 其二是可验证延迟函数的不可并行加速性.

3.1 可验证延迟函数的可靠性

3.1.1 可靠性的定义

首先给出可靠性的定义: 给定参数p、输入x、时间参数t, 通过计算算法$eval()$得到输出y以及证明π, 如式 (7) 所示:

$$eval(x,p,t)\rightarrow(y,\pi) \tag{7}$$

若 A 通过正确的计算步骤得到结果(y,π), B 通过伪造得到结果(y',π'), 且有$y\neq y',\pi\neq\pi'$. 如果可验证延迟函数的验证算法给出结果如式 (8)、式 (9), 则可以判定可验证延迟函数是可靠的.

$$verify_{pp}(x,y,\pi,\Delta)\rightarrow \text{true} \tag{8}$$

$$verify_{pp}(x,y',\pi',\Delta)\rightarrow \text{false} \tag{9}$$

3.1.2 生成伪证明

根据第 2.1.4 节的算法$verify()$可知在y, π都正确的情况下有$y=\pi^l g^r$. 式中r也可以写为:

$$r=2^t-l\left\lfloor 2^t/l\right\rfloor \tag{10}$$

将式 (10) 两边同乘以g^{-2^t}可得:

$$yg^{-2^t}=(\pi g^{-\lfloor 2^t/l\rfloor})^l \tag{11}$$

攻击者可以伪造证明, 即通过不等于y的结果y'来构造证明π', 构造方式如下:

① 计算$\alpha'=y'g^{-2^t}$

② 素数l ($l\rightarrow H_{\text{prime}}(g,y')$) 是已知的, 计算$\beta'=\sqrt[l]{\alpha'}$

③ 计算$\pi'=\beta' g^{\lfloor 2^t/l\rfloor}$

根据式 (11) 可以计算出$\pi'=(y'g^{-2^t})^{1/l}g^{\lfloor 2^t/l\rfloor}$.

只要攻击者发送给验证者的证明π'符合上述等式, 就可以在计算结果$y'\neq y$的情况下, 伪造证明.

3.1.3 RSA-VDF 群

$(Z/NZ)*$是一个 RSA 群, 而 RSA-VDF 群则表示为$(Z/NZ)*/\{\pm1\}$. 对于任意元素g, $h\in(Z/NZ)*$, 如果满足$g\cdot h^{-1}\in\{\pm1\}$, 则在 RSA-VDF 群中元素g, h是相同的. 也就是说在 RSA-VDF 群中, 对于输入x, y和$-y$都是正确的输出结果.

3.1.4 VDF 可靠性证明

VDF 的可靠性是建立在 RSA 的大整数分解困难问题之上的. 一般地, 定义$N=pq$是 RSA 群的模 (p, q 是未知的大素数), 如果有$u\in(Z/NZ)*/\{\pm1\}$, 那么在不知道 p,q 的情况下计算$\sqrt[l]{u}$是十分困难的 ($l=H_{\text{prime}}(g,y)\in(Z/NZ)*$) .

定义$u=\pi'/\pi$, 则有:

$$y/y'\leftarrow\sqrt[l]{u} \tag{12}$$

假设可验证延迟函数并不可靠, 即攻击者可以在多项式有界时间内产生一个错误的结果集$<y',\pi'>$, 且能通过验证者的验证, 这将违背 RSA 大整数分解困难问题.

如果能够绕过上述困难假设, 那么可以很快计算出$\sqrt[l]{u}$的值, 定义$\xi=\sqrt[l]{u}$. 根据 VDF 函数的定义有:

$$eval(x,t)\rightarrow(y,\pi) \tag{13}$$

则y', $\pi'(y'\neq y)$可以作以下变换:

$$y'=uy \tag{14}$$

$$\pi'=\xi\pi \tag{15}$$

上述不是通过$eval()$函数计算得出的y', π'也将通过验证者的验证, 即:

$$y=x^{2^t},r=2^t \bmod l \tag{16}$$

$$(\pi')^l x^r=(\xi\pi)^l=u\pi^l x^r=uy=y' \tag{17}$$

通过上述论证可知, 在大整数分解困难问题成立的前提下, VDF 函数具有很高的可靠性.

3.2 可验证延迟函数的不可并行加速性

在利用可验证延迟函数选出背书节点的过程中, 拥有私钥sk的合法背书节点将很快计算出结果y以及

证明π, 而不含私钥sk的情况下, 计算出正确的结果以及证明将耗费多项式时间O(t). 随着t的增大, 攻击者能够伪造结果的难度也将增大. 理想的情况下, 攻击者即使拥有多个计算资源能够并行处理输入x, 也将耗费单个计算资源所需要的计算时间, 也即不可并行加速.

3.2.1 计算模型

定义ϖ为计算模型, 即为了实现一定目的而采取的相关操作以及各自的开销. 对于特定的算法ψ, 定义$T(\psi,x)$为计算模型ϖ下输入为x的时间花销; $\theta(\psi,x)$为计算模型ϖ下所有的成本开销总和. $\theta(\psi,x)$不考虑并行性的问题, 而$T(\psi,x)$将考虑并行情况下的时间成本变化. 值得注意的是, $T(\psi,x)$只是一种分析时间花销的理论模型而不是真正的物理世界的时间. 对于方案的安全参数k, 定义$\delta: N* \to R$是进行单次模平方计算的时间刻度, 与安全参数k相关, 也可以写做$\delta(k)$.

3.2.2 Time-lock

给定安全参数k, $g \in (Z/NZ)*$且是均匀随机的取值, $N = pq$ (p,q是大素数), Time-lock 的概念如下:

$\Omega(g) \to h(g) = g^{2^t}$定义算法$\Omega(g) \to h(g) = g^{2^t}$

p,q在素数p,q未知的情况下计算开销$T(\Omega,g) < t\delta(k)$的概率是一个可以忽略的值.

由 Time-lock 可知, 在$N = pq$无法快速因式分解得出p,q的情况下, 求解$y = g^{2^t}$需要经过 t 次求解g 的平方且无法并行加速. 由于$\delta(k)$是一个与安全参数k相关的多项式表达式, 随着安全参数k的不断增大, $T(\Omega,g) < t\delta(k)$的概率将逐渐趋于零.

拥有私钥的sk的用户可以将$y = g^{2^t}$转换为求解$y = g^{2^t} \bmod sk$从而缩短计算时间.

综上所述, Time-lock 保证了可验证延迟函数的不可并行加速性.

4 实验分析

4.1 实验设计

本文基于 Hyperledger Fabric 的架构设计了相关的区块链交易系统, 该系统模拟了 Fabric 的交易流程. 作为对比, 本文将采取两种实验方案: 采用固定的方式抽取背书节点的原有方案以及采用可验证延迟函数抽取背书节点的优化方案. 两种方案都包含 4 类节点: 客户端节点、背书节点、排序节点以及提交节点. 值得注意的是, 优化方案在客户端节点刚刚发起交易时的背书节点只是候选背书节点集, 只有被可验证延迟函数抽取到的才能作为该交易流程中的背书节点对交易进行背书. 4 类节点的数量为: 200 个客户端节点, 30 个背书节点, 1 个排序节点和 150 个提交节点.

优化方案除了背书节点需要选取之外, 交易流程与原有方案是一样的: 不同的客户端节点独立提交相关的交易提案给背书节点, 交易经过背书返回客户端, 验证无误之后被提交给排序节点进行交易排序打包成区块, 最后提交节点将区块提交上链. 本文的实验环境为: Ubuntu 16.04 操作系统, 8 核 CPU, 16 GB 内存, 512 GB 硬盘.

考虑 Fabric 系统的特性, 我们将给予 Golang 语言来实现交易的业务逻辑, 可验证延迟函数的实现依赖于相关的 crypto 库以及外部已经实现的模块. 本文采用 Caliper 系统对实验结果进行测试, 并从平均交易时间以及交易延迟两个方面来进行性能分析.

4.2 平均交易时间

对原有方案以及优化方案进行平均交易时间测试, 测试结果如图 5 所示.

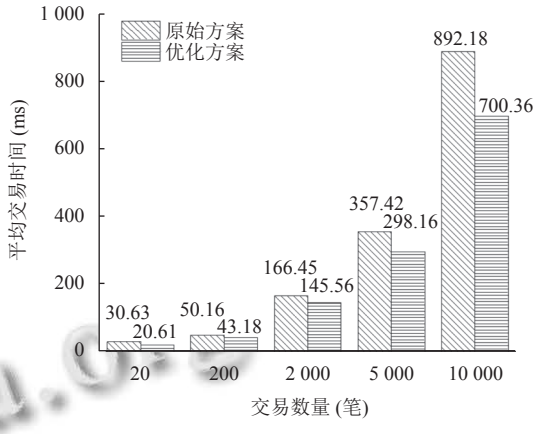

图 5 两种方案平均交易时间

从实验数据可以看出, 随着交易数量的增加, 原始方案和优化方案的平均交易时间也增加, 但优化方案平均交易时间要小于原始方案. 引入可验证延迟函数后, 一方面, 因为非恶意的背书节点拥有私钥 SK, 计算求解的速度不会延迟太多, 另一方面, 优化方案随机选取背书节点的方式使得背书节点可以交错为不同的交易背书, 提高了背书节点处理交易的效率; 而原始方案因为背书节点是固定的, 交易只能被背书策略指定的背书节点处理, 其他背书节点不能并行处理不同的交易, 造成资源的浪费.

综上, 优化方案通过引入可验证延迟函数, 既保证了背书节点的安全性, 也减少了背书节点平均交易时间.

4.3 交易延迟

本文基于 Caliper 对两种方案的交易延迟进行了测试, 实验结果如图 6 所示.

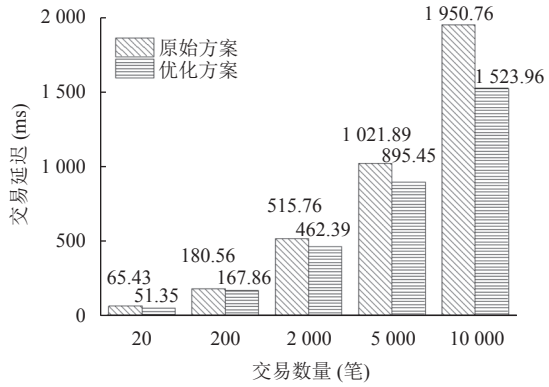

图 6 两种方案交易延迟

交易延迟是一笔交易从提交到能够被处理的时间, 由图 6 的实验结果可以看出, 两种方案的交易延迟都将随着交易数量的增加为增加且优化方案的交易延迟要小于原始方案. 在原始方案中, 所有的背书节点都是固定的, 当有一笔交易需要背书时, 所有的背书节点都需要等待, 之后根据背书策略对交易进行背书; 优化方案将背书节点划分为不同的群组, 当群组 1 的背书节点背书交易 1 时, 群组 2 的背书节点可以并行背书交易 2, 这就在一定程度上提高了背书效率.

5 总结

针对 Hyperledger Fabric 中背书节点身份暴露易被攻击的安全问题, 本文提出了基于可验证延迟函数的 Hyperledger Fabric 背书策略优化方案. 本方案将原始方案中固定背书节点划分为候选背书节点集, 通过可验证延迟函数随机抽取背书节点并完成身份验证. 最后, 将优化方案与原始方案作性能对比, 比较了两者的平均交易时间和交易延迟. 结果表明, 优化方案在保证背书节点安全性的同时在效率上也是可行的.

参考文献

1 李娟娟, 袁勇, 王飞跃. 基于区块链的数字货币发展现状与展望. 自动化学报, 2021, 47(4): 715–729.

2 Fu X, Wang HM, Shi PC. A survey of blockchain consensus algorithms: Mechanism, design and applications. Science China Information Sciences, 2021, 64(2): 121101. [doi: 10.1007/s11432-019-2790-1]

3 王群, 李馥娟, 倪雪莉, 等. 区块链共识算法及应用研究. 计算机科学与探索, 2022, 16(6): 1214–1242.

4 孟吴同, 张大伟. Hyperledger Fabric 共识机制优化方案. 自动化学报, 2021, 47(8): 1885–1898.

5 Androulaki E, Barger A, Bortnikov V, *et al*. Hyperledger fabric: A distributed operating system for permissioned blockchains. Proceedings of the Thirteenth EuroSys Conference. Porto: ACM, 2018. 30.

6 Androulaki E, de Caro A, Neugschwandtner M, *et al*. Endorsement in hyperledger fabric. Proceedings of 2019 IEEE International Conference on Blockchain (Blockchain). Atlanta: IEEE, 2019. 510–519.

7 高云, 严悍. Hyperledger Fabric 区块链软件架构中的中间件设计. 计算机与数字工程, 2020, 48(9): 2195–2200, 2274. [doi: 10.3969/j.issn.1672-9722.2020.09.023]

8 Mazumdar S, Ruj S. Design of anonymous endorsement system in Hyperledger Fabric. IEEE Transactions on Emerging Topics in Computing, 2021, 9(4): 1780–1791. [doi: 10.1109/TETC.2019.2920719]

9 张龙静. Hyperledger Fabric 背书策略的提案分发改进方案. 计算机科学与应用, 2021, 11(4): 1157–1164.

10 Manevich Y, Barger A, Tock Y. Endorsement in hyperledger fabric via service discovery. IBM Journal of Research and Development, 2019, 63(2–3): 2: 1–2: 9.

11 Boneh D, Bonneau J, Bünz B, *et al*. Verifiable delay functions. Proceedings of the 38th Annual International Cryptology Conference. Santa Barbara: Springer, 2018. 757–788.

12 Wesolowski B. Efficient verifiable delay functions. Proceedings of the 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Darmstadt: Springer, 2019. 379–407.

13 Pietrzak K. Simple verifiable delay functions. Proceedings of the 10th Innovations in Theoretical Computer Science Conference (ITCS 2019). Dagstuhl: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. 60: 1–60: 15.

14 Ephraim N, Freitag C, Komargodski I, *et al*. Continuous verifiable delay functions. Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Zagreb: Springer, 2020. 125–154.

15 Döttling N, Garg S, Malavolta G, *et al*. Tight verifiable delay functions. Proceedings of the 12th International Conference on Security and Cryptography for Networks. Amalfi: Springer, 2020. 65–84.

16 Rotem L. Simple and efficient batch verification techniques for verifiable delay functions. Proceedings of the 19th International Conference on Theory of Cryptography. Raleigh: Springer, 2021. 382–414.

(校对责编: 牛欣悦)