
硕士学位论文

# 基于区块链与零知识证明的物联网访问控制研究

# Research on IoT Access Control Based on Blockchain and Zero-knowledge Proof

作　者：张　驰
导　师：夏士雄 教授

中国矿业大学

二〇二二年六月

# 学位论文使用授权声明

本人完全了解中国矿业大学有关保留、使用学位论文的规定，同意本人所撰写的学位论文的使用授权按照学校的管理规定处理：

作为申请学位的条件之一，学位论文著作权拥有者须授权所在学校拥有学位论文的部分使用权，即：①学校档案馆和图书馆有权保留学位论文的纸质版和电子版，可以使用影印、缩印或扫描等复制手段保存和汇编学位论文；②为教学和科研目的，学校档案馆和图书馆可以将公开的学位论文作为资料在档案馆、图书馆等场所或在校园网上供校内师生阅读、浏览。另外，根据有关法规，同意中国国家图书馆保存研究生学位论文。

（保密的学位论文在解密后适用本授权书）。

作者签名：　　导师签名：

年　月　日　　年　月　日

中图分类号 学校代码 10290

UDC 密 级 公开

# 中国矿业大学

## 硕士学位论文

# 基于区块链与零知识证明的物联网访问控制研究

# Research on IoT Access Control Based on Blockchain and Zero-knowledge Proof

作 者 张驰 导 师 夏士雄

申请学位 工学硕士学位 培养单位 计算机科学与技术学院

学科专业 计算机应用技术 研究方向 物联网安全

答辩委员会主席 王志晓 评 阅 人 盲审

二〇二二年六月

# 致谢

时间飞逝，三年的研究生时光即将结束，回首这三年，自己一步一步走至现在，得到了很多人的帮助、关怀以及指引，在此向所有曾帮助过我的人表达由衷的感谢。

感谢我的恩师夏士雄教授，老师治学严谨、认真负责，为我树立了榜样，在生活中，老师平易近人，在疫情期间时常关心我们的日常生活与学业进展，给予我们关怀和鼓励。

感谢牛强老师与陈朋朋老师，在日常科研生活中，牛老师与陈老师对我们悉心关照，并为我们提供了良好的实验室环境。在论文写作的过程中，老师们提出了很多宝贵的意见，使我能够顺利完成论文的撰写。

感谢实验室的师兄、师姐以及师弟、师妹们，感谢大家的帮助、支持以及陪伴。

感谢我的父母、爱我的人以及我爱的人，感谢你们一直默默鼓励与支持着我。最后，感谢各位专家与老师，在百忙之中抽出宝贵的时间评阅我的论文，并提出宝贵的意见。

# 摘　要

随着因特网的普及，全球信息共享的成本越来越低，物联网技术日益发展，但安全与隐私问题也随之而来，访问控制是物联网安全与隐私的重要保障。传统的物联网访问控制模型需要将核心功能委托至可信第三方，其局限性在于（1）模型性能依赖于中央服务与（2）用户隐私信息被中心化管理。区块链技术能够在没有可信第三方的条件下有效解决信任问题，以及零知识证明技术能够在实现隐私保护的同时提升验证效率。因此，本文对基于区块链与零知识证明的物联网访问控制进行了系统研究，本文的主要研究内容及成果如下：

（1）提出了基于智能合约的物联网访问控制模型（IoTAC）。针对传统物联网访问控制模型需要将核心功能委托至可信第三方的问题，IoTAC 基于以太坊平台实现了基于属性的访问控制范式，使用区块链智能合约技术实现访问控制逻辑以及属性、策略等信息的存储，凭借区块链网络的去中心化特性，IoTAC 不再需要可信第三方的支持。IoTAC 适用于访问者实体属性信息、资源实体属性信息及访问控制策略较为简易的应用场景，经实验，IoTAC 的访问控制验证速率能够满足实际应用场景的要求。

（2）提出了基于双层区块链架构的物联网访问控制模型（IoTAC-L2）。针对可扩展性受限问题，IoTAC-L2 实现了链下存储、链上存证的双层区块链架构，将属性及策略信息以默克尔帕特里夏树的形式存储至链下，链上合约只存储默克尔帕特里夏树的根哈希值，通过默克尔证明保证链下数据的完整性。由于根哈希值的长度较小且固定，与链下数据的实际大小无关，所以 IoTAC-L2 能够释放链上资源，以实现高可扩展性，并且智能合约部署方便，模型灵活性较高，能够应用于复杂应用场景。经实验，IoTAC-L2 的访问控制验证速率与可扩展性能够满足实际应用场景。

（3）提出了基于区块链与简短非交互零知识证明的物联网访问控制模型（IoTAC-ZK）。针对隐私问题，IoTAC-ZK 将访问者实体的身份验证划分为身份有效性检查与属性检查两个步骤，通过默克尔证明实现身份有效性检查，布隆过滤器实现属性检查，最终使用开源工具库 ZoKrates 实现上述步骤的简短非交互零知识证明。在 IoTAC-ZK 中，访问控制系统能够验证访问者实体身份的合法性，但无法获知关于访问者实体身份的任何信息。IoTAC-ZK 适用于安全性要求较高的应用场景，经实验，IoTAC-ZK 不仅能够满足实际应用场景，而且能够显著提高访问控制的验证速率。

该论文有图 41 幅，表 13 个，参考文献 90 篇。

**关键词：**物联网；访问控制；区块链；简短非交互零知识证明

# Abstract

With the popularity of the Internet and the increasingly low cost of global information sharing, IoT technology is developing rapidly, but security and privacy issues also come up. Access control is an important guarantee for IoT security and privacy. Traditional IoT access control models require entrusting core functions to trusted third parties, which is limited in that (1) model performance depends on centralized services and (2) user privacy information is centrally managed. Blockchain technology can effectively solve the trust problem without a trusted third party, and zero-knowledge proof technology can improve the verification efficiency while achieving privacy protection. Therefore, this thesis conducts a systematic research on the IoT access control based on blockchain and zero-knowledge proof, and the main research contents and results of this thesis are as follows:

(1) An IoT access control model (IoTAC) based on smart contract is proposed. To address the problem that traditional IoT access control models need to delegate core functions to trusted third parties, IoTAC implements an attribute-based access control paradigm based on Ethereum, using blockchain smart contract technology to realize the access control logic and the storage of attribute and policy information, IoTAC no longer needs the support of trusted third parties due to the decentralized nature of the blockchain network. IoTAC is suitable for application scenarios where visitor entity attribute information, resource entity attribute information and access control policies are relatively simple, and the access control verification rate of IoTAC can meet the requirements of practical application scenarios after experiments.

(2) An IoT access control model (IoTAC-L2) based on two-layer blockchain architecture is proposed. To address the scalability limitation issue of IoTAC, IoTAC-L2 implements a two-layer blockchain architecture with off-chain storage and on-chain deposition, storing attribute and policy information in the form of Merkle Patricia trees under the chain, and on-chain contracts only store the root hash values of the Merkle Patricia trees, ensuring the integrity of the data under the chain through Merkle proof. Since the root hash length is a smaller fixed value, which is independent of the actual size of the data under the chain, IoTAC-L2 is able to release the on-chain resources and achieve high scalability. The smart contracts in IoTAC-L2 are easy to deploy and IoTAC-L2 has high model flexibility, so IoTAC-L2 can be applied to complex application scenarios. After experiments, the access control verification rate and scalability of IoTAC-L2 can meet the actual application scenarios.

(3) An IoT access control model (IoTAC-ZK) based on blockchain and zero-knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARK) is proposed. To address the privacy issue of IoTAC-L2, IoTAC-ZK divides the identity verification of visitor entities into two steps: identity existence checking and attribute checking. IoTAC-ZK achieves the existence check of the identity by Merkle proof, attribute checking by Bloom filter, and the zk-SNARK of the above steps using ZoKrates open source tool library. In IoTAC-ZK, the access control system is able to verify the validity of the identity of a visitor entity, but it is not able to obtain any information about the identity of the visitor entity. IoTAC-ZK is suitable for application scenarios with high security requirements. After experiments, IoTAC-ZK can not only meet the actual application scenarios, but also significantly improve the verification rate of access control.

This thesis has 41 figures, 13 tables, and 90 references.

**Keywords:** Internet of Things (IoT); access control; blockchain; zero-knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARK)

# 目　录

**摘　要 ..... I**
**目　录 ..... IV**
**图清单 ..... VIII**
**表清单 ..... XI**
**变量注释表 ..... XII**
**1 绪论 ..... 1**
1.1 研究背景与意义 ..... 1
1.2 研究现状 ..... 2
1.3 研究内容及成果 ..... 4
1.4 论文组织结构 ..... 5
**2 相关技术理论及分析 ..... 6**
2.1 物联网技术及架构 ..... 6
2.2 访问控制技术分类及分析 ..... 7
2.3 区块链工作原理及分析 ..... 10
2.4 简短非交互零知识证明 ..... 18
**3 基于链上计算与存储的物联网访问控制研究 ..... 29**
3.1 区块链数据存储及分析 ..... 29
3.2 基于智能合约的物联网访问控制模型设计 ..... 33
3.3 访问控制算法设计 ..... 35
3.4 实验评估及安全分析 ..... 37
**4 基于链上计算与存储的物联网访问控制的可扩展性优化研究 ..... 45**
4.1 基于双层区块链架构的物联网访问控制模型设计 ..... 45
4.2 访问控制算法设计 ..... 47
4.3 实验评估及安全分析 ..... 49
**5 基于区块链与简短非交互零知识证明的物联网访问控制隐私保护研究 ..... 55**
5.1 基于区块链与简短非交互零知识证明的物联网访问控制模型设计 ..... 55
5.2 访问控制算法设计 ..... 60
5.3 实验评估及安全分析 ..... 61
**6 总结与展望 ..... 69**
6.1 工作总结 ..... 69

6.2 工作展望 . . . . . 70
**参考文献 ..... 71**
**作者简历 ..... 78**
**学位论文原创性声明 ..... 79**
**学位论文数据集 ..... 80**

# Contents

**Abstract** ..... **II**

**Contents** ..... **VI**

**List of Figures** ..... **VIII**

**List of Tables** ..... **XI**

**List of Variables** ..... **XII**

**1 Introduction** ..... **1**

1.1 Research Background and Significance ..... 1

1.2 Research Status ..... 2

1.3 Research Content and Results ..... 4

1.4 Thesis Outline ..... 5

**2 Related Technical Theory and Analysis** ..... **6**

2.1 IoT Technology and Architecture ..... 6

2.2 Access Control Technology Classification and Analysis ..... 7

2.3 Blockchain Working Principle and Analysis ..... 10

2.4 Zero-knowledge Succinct Non-interactive Arguments of Knowledge ..... 18

**3 Research on IoT Access Control Based on On-chain Computing and Storage 29**

3.1 Blockchain Data Storage and Analysis ..... 29

3.2 IoT Access Control Model Design Based on Smart Contract ..... 33

3.3 Access Control Algorithm Design ..... 35

3.4 Experimental Evaluation and Safety Analysis ..... 37

**4 Research on Scalability Optimization of IoT Access Control Based on On-chain Computing and Storage** ..... **45**

4.1 Two-layer Blockchain Architecture-based IoT Access Control Model Design ..... 45

4.2 Access Control Algorithm Design ..... 47

4.3 Experimental Evaluation and Safety Analysis ..... 49

**5 Research on IoT Access Control Privacy Protection Based on Blockchain and zk-SNARK** ..... **55**

5.1 IoT Access Control Model Design Based on Blockchain and zk-SNARK ..... 55

5.2 Access Control Algorithm Design ..... 60

5.3 Experimental Evaluation and Safety Analysis ..... 61

**6 Conclusion and Future Work ..... 69**

6.1 Conclusion ..... 69

6.2 Future Work ..... 70

**References ..... 71**

**Author's Resume ..... 78**

**Declaration of Thesis Originality ..... 79**

**Thesis Data Collection ..... 80**

# 图清单


| 图序号 | 图名称 | 页码 |
| --- | --- | --- |
| 图 1-1 | 本文研究内容及思路 | 4 |
| Figure 1-1 | The research content and ideas of this thesis | 4 |
| 图 2-1 | 物联网体系架构示意图 | 6 |
| Figure 2-1 | IoT architecture diagram | 6 |
| 图 2-2 | 访问控制模型示意图 | 7 |
| Figure 2-2 | Access control model schematic | 7 |
| 图 2-3 | ABAC 模型示意图 | 9 |
| Figure 2-3 | ABAC model schematic | 9 |
| 图 2-4 | 默克尔树示意图 | 11 |
| Figure 2-4 | Merkle tree schematic | 11 |
| 图 2-5 | 交易的发起、验证与传播示意图 | 14 |
| Figure 2-5 | Schematic diagram of transaction initiation, validation and propagation | 14 |
| 图 2-6 | 交易的存储过程 | 15 |
| Figure 2-6 | Storing procedures of transactions | 15 |
| 图 2-7 | 账户结构示意图 | 16 |
| Figure 2-7 | Account Structure Schematic | 16 |
| 图 2-8 | EVM 工作原理示意图 | 17 |
| Figure 2-8 | Schematic diagram of how EVM works | 17 |
| 图 2-9 | zk-SNARK 工作流程示意图 | 19 |
| Figure 2-9 | Workflow of zk-SNARK schematic | 19 |
| 图 2-10 | 算术电路示意图 | 20 |
| Figure 2-10 | Arithmetic circuit schematic | 20 |
| 图 2-11 | R1CS 矩阵编码示意图 | 23 |
| Figure 2-11 | R1CS matrix coding schematic | 23 |
| 图 3-1 | 将数据以 string 类型或 bytes 存储至智能合约的 gas 消耗 | 30 |
| Figure 3-1 | Store data as string types or bytes to the smart contract's gas consumption | 30 |
| 图 3-2 | 将数据以不同数据类型存储至智能合约的 gas 消耗 | 31 |
| Figure 3-2 | Gas consumption for storing data in different data types to smart contracts | 31 |
| 图 3-3 | IoTAC 示意图 | 33 |
| Figure 3-3 | IoTAC schematic | 33 |
| 图 3-4 | IoTAC 系统架构图 | 34 |
| Figure 3-4 | IoTAC system architecture diagram | 34 |
| 图 3-5 | IoTAC 实验场景架构图 | 38 |
| Figure 3-5 | Experimental scenario architecture diagram of IoTAC | 38 |
| 图 3-6 | 链上存储实验结果 | 39 |
| Figure 3-6 | On-chain storage experimental results | 39 |
| 图 3-7 | 链上合约信息修改实验结果 | 40 |
| Figure 3-7 | Experimental results of on-chain contract information modification | 40 |

| | | |
|---|---|---|
| 图 3-8 | 实体信息删除示意图 | 41 |
| Figure 3-8 | Entity information deletion schematic | 41 |
| 图 3-9 | IoTAC 访问控制验证时间与属性数量的关系 | 42 |
| Figure 3-9 | Access control verification time versus number of attributes in IoTAC | 42 |
| 图 3-10 | IoTAC 时间性能测试结果 | 43 |
| Figure 3-10 | IoTAC time performance test results | 43 |
| 图 4-1 | IoTAC-L2 双层区块链架构工作原理 | 45 |
| Figure 4-1 | IoTAC-L2 two-layer blockchain architecture working principle | 45 |
| 图 4-2 | IoTAC-L2 系统架构图 | 46 |
| Figure 4-2 | IoTAC-L2 system architecture diagram | 46 |
| 图 4-3 | 数据完整性校验示意图 | 47 |
| Figure 4-3 | Data Integrity Check Schematic | 47 |
| 图 4-4 | IoTAC-L2 实验场景架构图 | 49 |
| Figure 4-4 | Experimental scenario architecture diagram of IoTAC-L2 | 49 |
| 图 4-5 | gas 消耗与二进制序列长度的关系 | 51 |
| Figure 4-5 | Gas consumption versus binary sequence length | 51 |
| 图 4-6 | IoTAC-L2 访问控制验证时间与属性数量的关系 | 52 |
| Figure 4-6 | Access control verification time versus number of attributes in IoTAC-L2 | 52 |
| 图 4-7 | IoTAC-L2 时间性能测试结果 | 53 |
| Figure 4-7 | IoTAC-L2 time performance test results | 53 |
| 图 5-1 | IoTAC-ZK 实现思路示意图 | 56 |
| Figure 5-1 | IoTAC-ZK implementation idea schematic | 56 |
| 图 5-2 | 访问者实体身份有效性验证的实现思路 | 56 |
| Figure 5-2 | Visitor entity identity validation implementation ideas | 56 |
| 图 5-3 | 访问者实体属性验证的实现思路 | 57 |
| Figure 5-3 | Visitor entity attribute verification implementation ideas | 57 |
| 图 5-4 | 布隆过滤器示意图 | 57 |
| Figure 5-4 | Bloom filter schematic | 57 |
| 图 5-5 | 布隆过滤器误报率模拟图 | 58 |
| Figure 5-5 | Bloom filter false alarm rate simulation chart | 58 |
| 图 5-6 | IoTAC-ZK 系统结构图 | 60 |
| Figure 5-6 | IoTAC-ZK system architecture schematic | 60 |
| 图 5-7 | IoTAC-ZK 实验场景架构图 | 62 |
| Figure 5-7 | Experimental scenario architecture diagram of IoTAC-ZK | 62 |
| 图 5-8 | 默克尔证明时间消耗 | 63 |
| Figure 5-8 | Merkle proof time consumption | 63 |
| 图 5-9 | 布隆过滤器时间消耗 | 64 |
| Figure 5-9 | Bloom filter time consumption | 64 |
| 图 5-10 | EdDSA 时间消耗统计图 | 65 |
| Figure 5-10 | EdDSA time consumption statistics chart | 65 |
| 图 5-11 | 访问者实体身份验证时间消耗统计图 | 66 |

| | | |
|---|---|---|
| Figure 5-11 | Visitor entity authentication time consumption statistics chart | 66 |
| 图 5-12 | 访问控制模型性能比较统计图 | 67 |
| Figure 5-12 | Access control model performance comparison statistics chart | 67 |

# 表清单


| 表序号 | 表名称 | 页码 |
| --- | --- | --- |
| 表 2-1 | 访问控制模型的分类 | 8 |
| Table 2-1 | Classification of access control models | 8 |
| 表 2-2 | 交易包含的字段以及每个字段的含义 | 14 |
| Table 2-2 | The fields contained in the transaction and the meaning of each field | 14 |
| 表 3-1 | IoTAC 硬件参数与操作系统信息 | 37 |
| Table 3-1 | Hardware parameters and operating system information of IoTAC | 37 |
| 表 3-2 | IoTAC 编译器与开源库版本信息 | 38 |
| Table 3-2 | Compiler and open source library version information of IoTAC | 38 |
| 表 3-3 | 合约部署的 gas 消耗 | 39 |
| Table 3-3 | Contracted deployment gas consumption | 39 |
| 表 3-4 | IoTAC 时间性能测试结果 | 42 |
| Table 3-4 | IoTAC Time Performance Test Results | 42 |
| 表 4-1 | IoTAC-L2 硬件参数与操作系统信息 | 49 |
| Table 4-1 | Hardware parameters and operating system information of IoTAC-L2 | 49 |
| 表 4-2 | IoTAC-L2 编译器与开源库版本信息 | 50 |
| Table 4-2 | Compiler and open source library version information of IoTAC-L2 | 50 |
| 表 4-3 | IoTAC-L2 时间性能测试结果 | 52 |
| Table 4-3 | IoTAC-L2 Time Performance Test Results | 52 |
| 表 5-1 | IoTAC-ZK 硬件参数与操作系统信息 | 61 |
| Table 5-1 | Hardware parameters and operating system information of IoTAC-ZK | 61 |
| 表 5-2 | IoTAC-ZK 相关编译器与开源库版本 | 62 |
| Table 5-2 | Compiler and open source library version information of IoTAC-ZK | 62 |
| 表 5-3 | EdDSA 时间消耗 | 65 |
| Table 5-3 | EdDSA time consumption | 65 |
| 表 5-4 | 访问者实体身份验证时间消耗 | 66 |
| Table 5-4 | Visitor entity authentication time consumption | 66 |
| 表 5-5 | 访问控制模型性能比较 | 67 |
| Table 5-5 | Access control model performance comparison | 67 |

# 变量注释表

| | |
|---|---|
| RLP | 递归长度前缀编码 |
| $A$ | 由属性名称$A_0$与属性值$A_1$构成的键值对 |
| $\mathbb{A}$ | 属性键值对集合 |
| $\mathcal{A}$ | 由所有操作构成的集合 |
| $\mathbb{C}$ | 智能合约 |
| $h$ | 哈希值 |
| $id_x$ | 实体$x$的标识 |
| $\mathcal{K}(\mathcal{M})$ | 获取由映射表$\mathcal{M}$的所有键构成的集合 |
| $\mathcal{M}$ | 映射表 |
| $[m]$ | 集合$\{1,2,\dots,m\}$的简写 |
| $N_{\mathcal{O}}$ | 资源实体的属性数量 |
| $N_{\mathcal{S}}$ | 访问者实体的属性数量 |
| $N_{\mathcal{P}}^{\mathcal{O}}$ | 访问控制策略中资源实体部分的属性数量 |
| $N_{\mathcal{P}}^{\mathcal{S}}$ | 访问控制策略中访问者实体部分的属性数量 |
| $\mathcal{O}$ | 由所有资源实体构成的集合 |
| $\mathcal{P}$ | 访问控制策略 |
| $\mathcal{R}_{\mathcal{O}}$ | 资源拥有者 |
| $\mathcal{S}$ | 由所有访问者实体构成的集合 |
| $sig$ | 签名 |
| $\mathbb{T}$ | 默克尔树或默克尔帕特里夏树 |
| $[\![\vec{x}]\!]$ | 由$\vec{x}$所有元素的加密结果构成的集合 |
| $\pi$ | 证明 |
| $0^n$ | 长度为$n$的零向量 |

# 1 绪论

# 1 Introduction

## 1.1 研究背景与意义（Research Background and Significance）

随着因特网的普及，全球信息共享的成本日益降低，物联网（Internet of Things，IoT）已经在当今世界中扮演着重要角色。在实际生活中，物联网设备已经遍布生活中的各个角落，每一个人的身边都存在着各式各样的物联网设备，这些智能设备为我们生活中的方方面面提供便利，比如，我们手中的智能手机、家中的智能家居、甚至路边的监控设施等，我们每一个人都生活在一个由相互连接的物联网设备编织成的“巨网”下，这个“巨网”就是物联网。梅特卡夫定律（Metcalfe's Law）[1]指出当一个网络的用户数量（或节点数量）极大时，该网络的价值与该网络的用户数量的平方成正比，据估计，到 2025 年全世界将有超过 750 亿台联网的物联网设备实现互联互通[2]，如果依据梅特卡夫定律，物联网对于当今世界的价值已难以估量或者物联网已与当今世界密不可分。

虽然物联网领域发展迅速、规模日益庞大，但是物联网领域目前仍存在着诸多挑战，其中包括安全与隐私问题[3,4]，这主要有以下三个方面的原因，首先，由于物联网是一个巨大的异构网络，设备、通信协议、网络结构等皆具有多样性，网络的安全性难以得到保证[5]。其次，传感器设备由于内存、能量以及计算能力有限，必须将大部分资源用于执行核心应用功能，难以使用传统的加密系统[6]。最后，由于物联网领域每时每刻都在产生着海量数据，敏感的私人数据在物联网中传播，而这些数据隐藏着使用者的行为与偏好，隐私问题也愈发尖锐[7]。

访问控制是维护物联网安全与隐私的重要保障。由于物联网设备计算与存储资源的受限性，传统的物联网访问控制技术往往需要将访问控制功能委托至可信第三方，其局限性在于可信第三方与物联网设备之间必须具有很强的信任关系，并且所有通信都是安全的、能够相互验证的[8]，这严重限制了传统访问控制模型的安全性及可拓展性。物联网需要一个适合其分布式性质的访问控制技术，对于隐私保护应当符合 GDPR①等标准，使用者能够管理自身隐私，而不是由可信第三方集中管理[9]。

区块链技术与零知识证明技术的出现给物联网访问控制技术的发展带来了新的机遇。区块链网络是一个去中心化的分布式系统，不会出现单点故障，并且能够在没有可信第三方的条件下有效解决网络节点间的信任问题，这些特性对于

①《通用数据保护条例》（General Data Protection Regulation，缩写作 GDPR；欧盟法规编号：(EU) 2016/679），又名《通用数据保护规则》，是在欧盟法律中对所有欧盟个人关于数据保护和隐私的规范，涉及了欧洲境外的个人数据出口。

物联网领域而言是极具吸引力的[10]。另外，近年来零知识证明技术日益发展，并逐渐应用于区块链领域，零知识证明技术可以在保护使用者隐私的同时提升区块链网络的验证效率[11]，因此将零知识证明技术应用于物联网访问控制中是极具价值，并且前景广阔的。

## 1.2 研究现状（Research Status）

### 1.2.1 传统物联网访问控制模型

传统的物联网访问控制架构主要分为 XACML、OAuth 及 UMA 三种[9,12]，详细介绍如下：

XACML[13]是基于属性的访问控制[14]（Attribute-based Access Control，ABAC）的具体实现标准，其中，ABAC 是一种支持布尔逻辑，使用属性组合策略授予使用者访问权限的访问控制范式。XACML 是一个细粒度、较为灵活的访问控制标准，但是 XACML 需要引入可信第三方作为授权引擎来处理访问控制逻辑。

OAuth[15]是一种基于代币（或令牌、Token）的授权协议，能够在不提供用户名与密码的条件下，完成对第三方应用的授权，资源访问者需要向资源所有者申请访问资源的权限，资源所有者对资源访问者进行授权，即允许访问控制服务（或可信第三方）向资源访问者发送代币，最终，资源访问者可以访问代币呈现的资源。

UMA（User-Managed Access）[16]是在 OAuth 协议的基础上进行扩展而来的，UMA 从资源所有者的角度重新对访问控制流程中的参与者进行了定义，UMA 能够在 OAuth 的基础上实现对资源的细粒度控制[17]。

但是，传统物联网访问控制架构都需要引入可信第三方，传统物联网访问控制架构在物联网的实际应用场景中具有局限性。首先是可扩展性问题，当实际应用场景中的物联网设备数量逐渐增大时，中央服务的信任管理将变得复杂，并且当中央服务的性能不足以支撑数量巨大的物联网设备时，就需要对中央服务进行扩展，此时访问控制就转化为分布式系统的协调合作问题；其次是单点故障问题，如果中央服务出现故障，这将严重影响访问控制架构的正常运行，并且每个对物联网资源的请求都需要包含特定的中央服务实体，中央服务实体易遭受恶意攻击；最后是隐私问题，用户隐私数据被中央服务中心化管理，用户隐私的安全性存在风险。

### 1.2.2 基于区块链的物联网访问控制模型

FairAccess[9]是第一个将区块链技术应用于物联网访问控制领域的物联网访问控制模型。FairAccess 使用智能合约技术表达细粒度的访问控制策略，并执行访问控制逻辑，即使用智能合约代替传统物联网访问控制模型所依赖的可信第三

方。在 FairAccess 中，使用代币作为判断访问者实体是否具有访问权限的依据，FairAccess 的主要贡献如下，首先，FairAccess 可以在没有中央服务的分布式环境中执行访问控制策略；其次，FairAccess 能够保证所有交互实体正确地执行访问控制策略；最后，FairAccess 具有代币重用检测机制。但是 FairAccess 存在一些缺陷，首先，FairAccess 生成一个新的代币至少要在区块链中挖掘出两个区块，即代币是昂贵的；其次，FairAccess 没有给出关系授权的解决方案，这里的关系授权是指如果一个未被授权的访问者与一个已被授权的访问者具有某种联系，那么此未被授权的访问者是否也应该具有访问权限。

Controlchain[12]给出了一个称为关系链的设计用以解决 FairAccess 的关系授权问题，并具有完整、透明的去中心化授权过程。另外，Controlchain 还提出了一种称为解码器的设计，解码器可以使 Controlchain 只需要略微调整就能够与其他访问控制范式兼容。

还有一些成果实现了基于区块链与 ABAC 的物联网访问控制架构。文献[18]提出了一个基于区块链与 ABAC 的物联网访问控制设计方案，该方案使用 Hyperledger Fabric 作为底层区块链平台，并将同一联盟下的物联网设备划分为一个域，在该方案中，物联网设备能够作为区块链节点，并支持物联网设备的跨域访问。文献[19]给出了一个将以太坊作为底层区块链平台的基于 ABAC 的物联网访问控制设计方案，该方案将所有数据存储至智能合约中。

还有一些成果将区块链应用于物联网领域。文献[20]提出了一个基于区块链的物联网框架，在该框架中，物联网设备能够在不同的区块链平台间共享，并且能够通过智能合约技术共享数据。文献[21]提出了一个基于区块链的智能家居框架，该框架使用本地私有区块链对物联网设备及其数据进行安全访问控制，并且链上数据可用来提供相应服务。文献[22]提出了一个适用于物联网的公钥基础设施设计方案。

### 1.2.3 基于区块链与零知识证明的物联网访问控制模型

将零知识证明技术引入基于区块链的物联网访问控制模型中，能够为使用者的隐私提供安全保障，甚至能够在完成访问控制的同时隐匿使用者身份。目前，关于基于区块链与零知识证明的物联网访问控制模型的研究成果还比较少。

BZBAC[23]在 ABAC 的基础上使用以太坊智能合约与简短非交互零知识证明算法 Groth16[24]实现了一个基于区块链与零知识证明的物联网访问控制架构，该架构在提高访问效率的同时，还能够隐藏使用者的隐私信息，BZBAC 提出了一个称为零知识访问代币的设计，此设计能够提升使用者的访问速率，并且减少对链上计算资源的占用，另外，BZBAC 使用代理设备作为物联网网关以增强访问控制策略的适用性。

另外，还有一些成果将区块链与零知识证明技术应用于其他领域，例如，文献[25]将区块链技术应用于能源交易中，并使用开源工具库 ZoKrates 实现基于简短非交互零知识证明的能源数据隐私保护模型。

## 1.3 研究内容及成果（Research Content and Results）

传统物联网访问控制模型的局限性在于模型性能依赖于中央服务，并且用户隐私信息被中心化管理，本文基于传统物联网模型的上述局限性，对基于区块链与零知识证明的物联网访问控制模型进行了系统研究，研究内容及思路如图 1-1 所示。

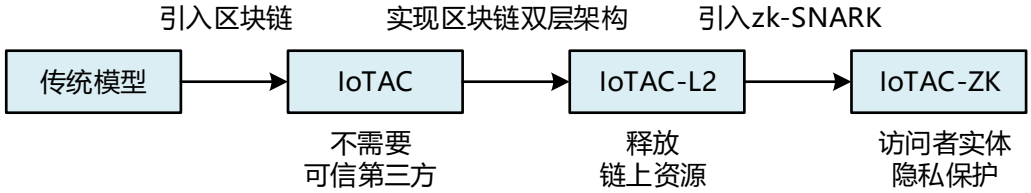



图 1-1 本文研究内容及思路

Figure 1-1 The research content and ideas of this thesis

本文的主要研究内容及成果如下：

（1）提出了基于智能合约的物联网访问控制模型（IoTAC）。传统物联网访问控制模型需要将核心功能委托至可信第三方，其局限性在于，一方面，模型性能依赖于中央服务，存在单点故障等问题；另一方面，使用者的隐私信息被可信第三方中心化管理，存在隐私问题。针对传统模型的上述局限性，IoTAC 使用智能合约技术完成访问控制逻辑的计算以及属性、策略等信息的存储，由于区块链网络具有去中心化特性，IoTAC 不再依赖于可信第三方，并且资源拥有者能够自行管理隐私信息。

（2）提出了基于双层区块链架构的物联网访问控制模型（IoTAC-L2）。在 IoTAC 中，访问控制逻辑的计算与相关数据的存储皆由链上合约完成，这导致 IoTAC 严重依赖于链上资源，而链上资源的昂贵性导致 IoTAC 的可扩展性受限。针对 IoTAC 的可扩展性受限问题，IoTAC-L2 在 IoTAC 的基础上实现了基于链下存储、链上证明的双层区块链架构。在 IoTAC-L2 中，访问控制逻辑的计算与相关数据的存储皆于链下完成，链上只存储数据的证明信息（默克尔帕特里夏树的根哈希值），由于证明信息的长度较小且固定，与实际数据的大小无关，所以 IoTAC-L2 能够在保证数据完整性的前提下，释放链上资源，获得较好的可扩展性。

（3）提出了基于区块链与简短非交互零知识证明的物联网访问控制模型（IoTAC-ZK）。IoTAC-L2 虽然实现了较好的可扩展性，但资源拥有者能够从物联网网关中获知访问者实体的具体访问信息，即存在隐私问题。针对 IoTAC-L2

的隐私问题，本文在 IoTAC-L2 的基础上引入了简短非交互零知识证明技术，将其拓展至 IoTAC-ZK。IoTAC-ZK 满足：1）访问者实体无法欺骗访问控制系统；2）访问者实体能够向访问控制系统证明自身身份的合法性；3）访问控制系统无法获知关于访问者实体身份的任何信息。另外，IoTAC-ZK 能够提高访问控制验证的效率。

## 1.4 论文组织结构（Thesis Outline）

本文共分为六个章节，对基于区块链与零知识证明的物联网访问控制进行了系统研究，各章节的主要内容安排如下：

第一章：绪论。首先，介绍了本文（基于区块链与零知识证明的物联网访问控制）的研究背景与意义；其次，介绍了传统物联网访问控制模型以及将区块链与零知识证明技术应用于物联网访问控制的研究现状；最后，介绍了本文的主要研究内容及成果。

第二章：相关技术理论及分析。此章对本文涉及的相关技术及原理进行了介绍，本文主要涉及物联网、访问控制、区块链以及零知识证明等技术。

第三章：基于链上计算与存储的物联网访问控制研究。首先，介绍了将数据存储至链上的实现方式，并对每种实现方式进行了实验及分析；其次，为解决传统物联网访问控制模型需要依赖可信第三方的问题，构建了基于智能合约的物联网访问控制模型（IoTAC）；最后，对 IoTAC 进行了相关实验及分析。

第四章：基于链上计算与存储的物联网访问控制的可扩展性优化研究。针对 IoTAC 的可扩展性受限问题，提出了基于双层区块链架构的物联网访问控制模型（IoTAC-L2），并对 IoTAC-L2 进行了相关实验及分析。

第五章：基于区块链与简短非交互零知识证明的物联网访问控制隐私保护研究。首先，介绍了简短非交互零知识证明的实现方式与数学原理；其次，针对 IoTAC-L2 的隐私问题，提出了基于区块链与简短非交互零知识证明的物联网访问控制模型（IoTAC-ZK）；最后，对 IoTAC-ZK 进行了相关实验及分析。

第六章：总结与展望。对本文的研究成果与不足进行了归纳总结，并对未来研究方向进行了探讨。

# 2 相关技术理论及分析

# 2 Related Technical Theory and Analysis

## 2.1 物联网技术及架构（IoT Technology and Architecture）

国际电信联盟（International Telecommunication Union，ITU）①对物联网的定义为“信息社会的全球性基础设施，基于现有的以及不断演进的、可互操作的信息与通信技术，通过（物理和虚拟）设备的互联互通来提供更加高级的服务”。一般来说，物联网是由可以捕获和共享数据的具有唯一可识别身份（MAC 地址或 IP 地址）的实体组成的巨大异构网络，其目标是为用户提供相应的智能服务。对于物联网的体系架构，物联网领域并没有统一的标准，但现有的物联网体系架构但大同小异，本课题依据文献[4],[26-28]将物联网体系架构分为四层，即感知层、网络层、中间件层及应用层，如图 2-1 所示，各层的详细解释如下：

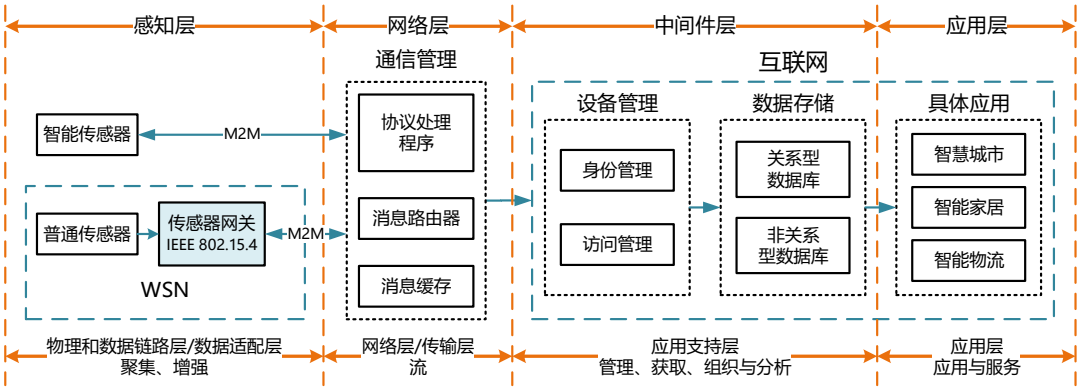



图 2-1 物联网体系架构示意图

Figure 2-1 IoT architecture schematic

感知层（Perception Layer）由传感器节点与传感器网关构成，其中，传感器节点与传感器网关可以组成无线传感网（Wireless Sensor Networks，WSN）或自组网（Ad Hoc Networks）。感知层的主要功能是从周围环境中采集数据并上传至网络层或执行网络层下达的命令。传感器节点可以采用有线方式，也可以采用无线方式传输数据，传感器网关必须实现 2.4 GHz IEEE 802.15.4 无线通信标准[29]。

网络层（Network Layer）的主要功能是提供因特网支持。网络层需要支持相应的通信协议（例如，TCP/IP 协议族与 MQTT）与通信方式（例如，P2P 与 M2M），另外，网络层是中间件层与感知层之间的桥梁，网络层需要提供相应的远程过程调用（Remote Process Call，RPC）接口，以使中间件层能够通过网络层获取到感知层中的底层物联网数据。

①国际电信联盟（International Telecommunication Union，ITU）是一个国际组织，主要负责确立国际无线电和电信的管理制度和标准。

中间件层（Middleware Layer）的主要功能是对感知层上传的数据进行存储与计算，中间件层负责构建中央云端或服务对物联网数据进行中心化管理。

应用层（Application Layer）的主要功能是以中间件层为基础，向使用者提供具体的物联网服务。

## 2.2 访问控制技术分类及分析（Access Control Technology Classification and Analysis）

在计算机安全中，访问控制一般包括识别、认证、授权、访问批准以及审计等，访问控制模型由访问者实体（也称为主体，在外文文献中通常为 Subject）、资源实体（也称为客体，在外文文献中通常为 Object）和访问控制策略三部分构成，在访问控制模型中，需要进行访问控制的资源称为资源实体，需要对资源实体进行访问的实体称为访问者实体[30]，访问者实体与资源实体都应当被视为软件实体，而不是人类使用者，人类使用者只能通过软件实体对访问控制系统产生影响[31]，图 2-2 粗略展示了访问控制模型的结构。另外，访问控制的设计应当遵循最小特权原则（Principle of Least Privilege，PoLP）[32]，PoLP 要求访问者实体（例如，进程、用户以及程序等）必须能够访问且仅能访问其合法目的所必需的信息与资源。

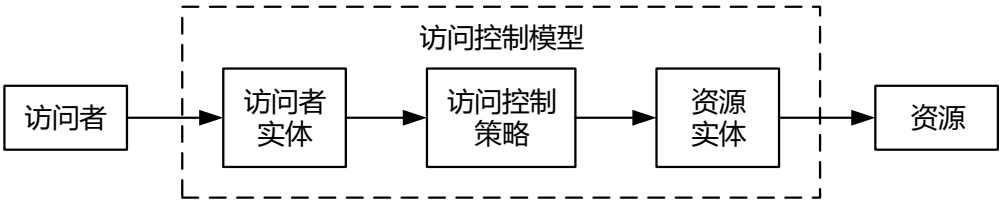



图 2-2 访问控制模型示意图

Figure 2-2 Access control model schematic

为方便下文表述，本文做如下定义：

1) 符号$\mathcal{S}$表示由所有访问者实体组成的集合；
2) 符号$\mathcal{O}$表示由所有资源实体组成的集合；
3) 符号$\mathcal{P}$表示由所有访问控制策略组成的集合；
4) 符号$\mathcal{RO}$表示由所有资源拥有者组成的集合。

### 2.2.1 访问控制模型的分类

访问控制模型主要有两种划分方法，一种是依据访问控制模型的认证方式，另一种是依据访问策略的制定者[31]，如表 2-1 所示，访问控制模型分类的详细介绍如下：

访问控制模型依据认证方式可以划分为基于功能的访问控制模型（Capability Based Access Control，CapBAC）[33]与基于访问控制列表（Access Control List，ACL）[34]的访问控制模型。在 CapBAC 模型中，如果访问者实体持

有对资源实体不可伪造的能力（在有些模型中，也被称为密钥），则访问者实体能够访问该资源实体，并且这种能力是可以被通信传输的，访问控制系统必须在安全信道上传输这种能力，以将访问权限授予给访问者实体。在基于 ACL 的访问控制模型中，访问者实体能否访问某个资源实体取决于其身份信息是否被添加至此资源实体对应的 ACL 中，访问控制系统通过修改 ACL 来授予访问者实体相应的访问权限，另外，对于 ACL 的维护者以及维护方式，不同的基于 ACL 的访问控制模型有不同的实现标准。

表 2-1 访问控制模型的分类

Table 2-1 Classification of access control models

| 分类依据 | 类型 | 特点 | 主流模型 |
|---|---|---|---|
| 认证方式 | 基于功能的模型 | 是否持有对资源不可伪造的能力 | ABAC |
| | 基于 ACL 的模型 | 身份是否存在于资源的 ACL 中 | DAC、MAC、RBAC |
| 访问策略制定者 | 自主型 | 资源所有者确定访问策略 | DAC、ABAC |
| | 强制型 | 访问控制系统确定访问策略 | MAC、RBAC |

访问控制模型依据访问控制策略的制定者可以划分为自主型与强制型，对于自主型，访问控制策略由资源拥有者制定；对于强制型，访问控制策略由访问控制系统制定。

最广泛认可的访问控制模型主要有自主访问控制（Discretionary Access Control，DAC）[35]、强制访问控制（Mandatory Access Control，MAC）[36]、基于角色的访问控制（Role-Based Access Control，RBAC）[37,38]以及基于属性的访问控制（Attribute-Based Access Control，ABAC）[39,40]。

在 DAC 模型中，访问控制策略由资源拥有者制定，资源拥有者决定哪些访问者实体能够访问属于自己的资源实体，以及访问者实体能够对资源实体进行哪些操作。

在 MAC 模型中，访问控制策略由访问控制系统制定，访问者实体是否能够访问某个资源实体，取决于在访问控制系统中是否存在允许其访问该资源实体的规则，MAC 模型在操作系统领域中有着广泛应用。

在 RBAC 模型中，访问控制策略由访问控制系统制定，但 RBAC 模型可以视为 DAC 模型与 MAC 模型的折中，RBAC 模型与 DAC 模型的不同之处在于 DAC 模型允许资源拥有者控制访问者实体对资源实体的访问，而在 RBAC 模型中，访问控制逻辑由访问控制系统执行；虽然 RBAC 模型属于强制型，但 RBAC 模型与 MAC 模型的不同之处在于对权限的具体处理方式，MAC 模型根据访问者实体与资源实体的安全级别（或安全属性）进行访问控制，而 RBAC 模型将每个访问者实体视为一个权限集合。

在 ABAC 模型中，访问控制策略由资源拥有者制定，访问控制策略指定访问者实体需要具有具有哪些属性才能够访问某一个资源实体，另外，访问者实体必须向访问控制系统证明其对自身属性的声明。

### 2.2.2 基于属性的访问控制模型

ABAC 模型是基于 XACML 标准实现的，图 2-3 展示了 ABAC 模型的组成结构。根据 NIST[41]与 RFC 2904[42]对 XACML 标准的定义，XACML 标准由以下四部分构成：

1) 策略执行点（Policy Enforcement Point，PEP）。PEP 负责接收来自访问者实体的请求，并将请求中的相关信息发送至 PDP，最终执行由 PDP 返回的验证结果；

2) 策略决策点（Policy Decision Point，PDP）。PDP 负责对 PEP 提供的请求信息进行访问控制验证（或访问控制逻辑的计算），并将验证结果返回给 PEP；

3) 策略信息点（Policy Information Point，PIP）。PIP 负责管理与存储访问者实体与资源实体的属性信息以及当前环境信息；

4) 策略获取点（Policy Retrieval Point，PRP）。PRP 负责管理与存储访问控制策略。

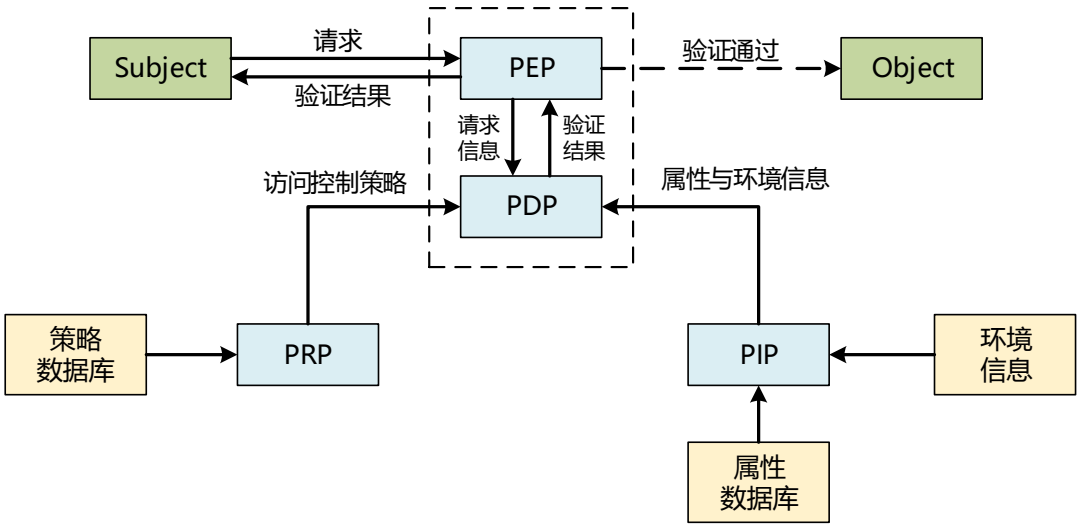



图 2-3 ABAC 模型示意图

Figure 2-3 ABAC model schematic

为方便下文表述，本文做如下定义：

1) $A = (A_0, A_1)$表示属性信息键值对，$A_0$表示属性名称，$A_1$表示属性值；

2) $[x]$表示集合$\{1,2,\ldots,x\}$。

在 ABAC 中，访问请求$R$可以表示为：

$$R = (s, o, a), \qquad s \in \mathcal{S} \wedge o \in \mathcal{O} \wedge a \in \mathcal{A} \tag{2-1}$$

当 PEP 接收到来自访问者实体的访问请求$R$后，PEP 从请求$R$中解析出访问者实体$s$、资源实体$o$以及操作$a$等信息并发送至 PDP，PDP 接收到相应的信息后，首先，PDP 从 PIP 中获取访问者实体$s$对应的属性信息集合$\mathbb{A}_S$、资源实体$o$对应的属性信息集合$\mathbb{A}_O$以及当前执行环境$\mathcal{E}$对应的属性信息集合$\mathbb{A}_\mathcal{E}$，$\mathbb{A}_S$、$\mathbb{A}_O$与$\mathbb{A}_\mathcal{E}$可以表示为：

$$\begin{cases}\mathbb{A}_s = \{A_i^s : i \in [k]\} \\ \mathbb{A}_o = \{A_i^o : i \in [m]\} \\ \mathbb{A}_e = \{A_i^e : i \in [n]\}\end{cases} \tag{2-2}$$

然后，PDP 将从 PRP 中获取操作$a$对应的访问控制策略$\mathcal{P}_a$，$\mathcal{P}_a$可以表示为：

$$\mathcal{P}_a = (\mathcal{P}_S, \mathcal{P}_O, \mathcal{P}_\mathcal{E}) \tag{2-3}$$

其中，$\mathcal{P}_S$表示访问者实体部分策略信息，$\mathcal{P}_O$表示资源实体部分策略信息，$\mathcal{P}_\mathcal{E}$表示环境部分策略信息，$\mathcal{P}_S$、$\mathcal{P}_O$与$\mathcal{P}_\mathcal{E}$可以表示为：

$$\begin{cases}\mathcal{P}_S = \{\mathbb{A}_i^S : i \in [K]\} \\ \mathcal{P}_O = \{\mathbb{A}_i^O : i \in [M]\} \\ \mathcal{P}_\mathcal{E} = \{\mathbb{A}_i^\mathcal{E} : i \in [N]\}\end{cases} \tag{2-4}$$

当且仅当访问者实体的请求$R$满足公式（2-5）时，请求$R$能够通过访问控制验证，即访问者实体$s$能够对资源实体$o$执行操作$a$。

$$\exists \mathbb{A}_i \in \mathcal{P}_S, \mathbb{A}_j \in \mathcal{P}_O, \mathbb{A}_k \in \mathcal{P}_\mathcal{E} : (\mathbb{A}_i \subseteq \mathbb{A}_s) \wedge (\mathbb{A}_j \subseteq \mathbb{A}_o) \wedge (\mathbb{A}_k \subseteq \mathbb{A}_e) \tag{2-5}$$

## 2.3 区块链工作原理及分析（Blockchain Working Principle and Analysis）

区块链是一个不断增长的记录列表，每条记录称为一个区块，这些区块使用密码学技术连接在一起，每个区块包含着时间戳、交易数据和前一个区块的加密散列（哈希值）等信息，所有区块构成一个有向无环图[43]。因为每个区块都蕴含着前一个区块的信息，所以如果攻击者篡改某一个区块，就必须篡改该区块的所有后续区块。所以区块链可以防止数据被恶意篡改，并且每新添加一个区块都会加强整个的区块链的不可篡改性。区块链具有公开透明性与不可篡改性，数据一旦写入区块链中，攻击者难以对数据完成篡改，另外任何人都可以对存储在区块链中的数据的正确性进行验证。所以可以使用区块链对物联网中的数据进行存储或者存证，以保证物联网中的数据的数据完整性。

### 2.3.1 默克尔树

默克尔树（Merkle Tree，MT）由 Ralph Merkle 于 1979 年提出[43]，图 2-4 展示了默克尔树的结构，默克尔树在密码学及计算机科学中是一种树形数据结构，每个叶节点均以数据块的哈希作为标签，叶节点之外的其他节点的标签由其子节点的标签计算而来，计算方法如下：

$$h = \mathcal{H}(h_l \mid h_r) \tag{2-6}$$

其中，$\mathcal{H}$表示哈希函数，$h_l$表示左子节点的标签，$h_r$表示右子节点的标签，“|”运算符表示将$h_l$与$h_r$拼接。默克尔树能够高效、安全地验证大型数据结构的内容，是哈希链表的推广形式[45]。

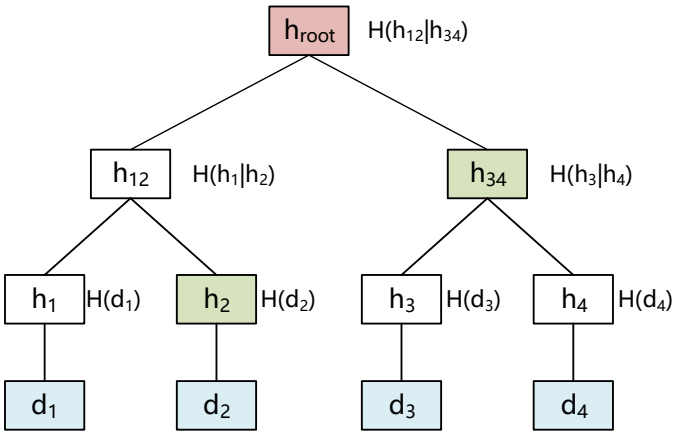



图 2-4 默克尔树示意图

Figure 2-4 Merkle tree schematic

默克尔树的一个重要应用是验证一个元素是否已被存储至默克尔树中，即提供默克尔证明（Merkle Proof），默克尔证明是待证明元素在默克尔树中的路径，可以表示为：

$$\pi = \{(p_i, h_i) : i \in [D]\} \tag{2-7}$$

其中，$D$表示默克尔树的高度，$p_i$表示被证明节点相对于节点$h_i$的位置，$p_i \in \{left, right\}$。例如，对于图 2-4 中叶子节点$d_1$的默克尔证明可以表示为$\{(left, h_2), (left, h_{34})\}$。验证方只需要依次计算出$h_1$、$h_{12}$以及$h_{root}$，如果$h_{root}$与默克尔树的根哈希值相等，则说明元素$d_1$已被存储至默克尔树中。

在本文中，使用默克尔树进行关系证明（Proof of Membership），即验证一个元素是否存在于某个集合（集合以默克尔树的形式存储）中，如果集合中元素的数量为$N$，则默克尔证明能够以时间复杂度$O(log_2 N)$快速完成关系证明。

### 2.3.2 默克尔帕特里夏树

本小节依据文献[45]，默克尔帕特里夏树（Merkle Patricia Tree，MPT）是一个广泛应用于以太坊数据存储的数据结构，默克尔帕特里夏树融合了默克尔树和帕特里夏树（Patricia Trie 或 Radix Tree）[47]两种数据结构的优点，应用于存储键值对集合，可以被视为映射表，文献[45]使用$I \equiv (I_0, I_1)$表示键值对数据结构，其中，$I_0$表示键字段，$I_1$表示值字段。

默克尔帕特里夏树的输入可以表示为：

$$\mathcal{J} = \{(k_0 \in \mathbb{B}, v_0 \in \mathbb{B}), (k_1 \in \mathbb{B}, v_1 \in \mathbb{B}), \cdots\} \tag{2-8}$$

其中，$\mathbb{B}$表示字节序列。默克尔帕特里夏树首先将$\mathcal{J}$中的键字段进行半字节（Nibble）编码，得到$y(\mathcal{J})$，$y(\mathcal{J})$的定义见公式（2-9），即将一个完整字节按照大端序拆分为两个 16 进制数，编码算法如公式（2-10）所示。

$$y(\mathcal{J}) = \{(k'_0 \in \mathbb{B}, v_0 \in \mathbb{B}), (k'_1 \in \mathbb{B}, v_1 \in \mathbb{B}), \cdots\} \tag{2-9}$$

$$k'_n[i] \equiv \begin{cases} \lfloor k_n[i/2]/16 \rfloor & if\ i/2 = 0 \\ k_n[\lfloor i/2 \rfloor] \bmod 16 & otherwise \end{cases} \tag{2-10}$$

其中，$i < 2\|k_n\|$，$\|\cdot\|$表示“$\cdot$”的长度（单位：byte）。默克尔帕特里夏树的节点生成方法如下：

$$n(\mathcal{J}, i) = \begin{cases} () & if\ \mathcal{J} = \emptyset \\ c(\mathcal{J}, i) & if\ \|c(\mathcal{J}, i)\| < 32 \\ \mathrm{KEC}\big(c(\mathcal{J}, i)\big) & otherwise \end{cases} \tag{2-11}$$

其中，KEC表示 Keccak-256 哈希函数[49]，$c$为压缩函数，$c$的定义见公式（2-12），另外，为节约存储空间，对于压缩后长度仍不小于 32 byte 的节点，在默克尔帕特里夏树中只存储其哈希值。默克尔帕特里夏树由以下几种节点构成：

叶节点。叶节点是由保留路径与值字段$I_1$计算而来，叶节点的定义见公式（2-12）的第一部分，其中，$i$表示深度，$I_0[i..(\|I_0\| - 1)]$为保留路径，从根节点一直向下遍历至此叶节点的父节点所累积的键字段可以表示为$I_0[0..i-1]$。$I_0[i..(\|I_0\| - 1)]$为从键字段$I_0$中去除$I_0[0..(i-1)]$的剩余部分。根据公式（2-14）及公式（2-15）能够推导出，$\mathrm{HP}(I_0[i..(\|I_0\| - 1)], 1)$拥有固定前缀，如果$\|I_0[i..(\|I_0\| - 1)]\|$为偶数，则前缀为“0x2”，反之，则前缀为“0x3”。

扩展节点。扩展节点是由保留路径与值字段$I_1$计算而来，扩展节点的定义见公式（2-12）的第二部分，其中，函数$\alpha(\mathcal{J})$的返回值$j$是$\mathcal{J}$中所有键字段最长公共前缀的长度。$\mathrm{HP}(I_0[i..(j-1)], 0)$也拥有固定前缀，如果$\|I_0[i..(j-1)]\|$为偶数，则前缀为“0x0”，反之，则前缀为“0x1”。计算出$j$后还需再次调用函数$n(\mathcal{J}, j)$进行递归运算，直至得出最终结果。

分支节点。分支节点是由 16 个半字节数值（即 0 至 15）与终结标志字段计算而来，分支节点的定义见公式（2-12），其中，函数$\beta(\mathcal{J}, i, j)$是一个递归运算，功能是先搜索出由$\mathcal{J}$中第$i$个半字节为$j$的键$I_0$对应的键值对$I$构成的集合，再将此集合代入公式（2-11）中，直至得出最终结果，另外，对于$\gamma(\mathcal{J}, i)$，如果在集合$\mathcal{J}$中存在一个键值对的键字段与当前分支节点的键字段相同，那么返回该键值对的值字段，否则返回空值。

$$c(\mathcal{J}, i) = \begin{cases} \mathrm{RLP}\big((\mathrm{HP}(I_0[i..(\|I_0\| - 1)], 1), I_1)\big) & if\ \|\mathcal{J}\| = 1 \quad where\ \exists I: I \in \mathcal{J} \\ \mathrm{RLP}\Big(\big(\mathrm{HP}(I_0[i..(j-1)], 0), n(\mathcal{J}, j)\big)\Big) & if\ i \neq j \quad where\ j = \alpha(\mathcal{J}) \\ \mathrm{KEC}\big((u(0), u(1), \cdots, u(15), v)\big) & otherwise \quad where\ u(j) = \beta(\mathcal{J}, j) \\ & \qquad\qquad\qquad\qquad\quad v = \gamma(\mathcal{J}, i) \end{cases} \tag{2-12}$$

其中，$\alpha$、$\beta$和$\gamma$的定义如下：

$$\begin{aligned}\alpha(\mathcal{J}) &= \text{argmax}_x : \exists \mathbf{I} : \|\mathbf{I}\| = x : \forall I \in \mathcal{J} : I_0[0..(x-1)] = \mathbf{I} \\ \beta(\mathcal{J}, i, j) &\equiv n(\{I : I \in \mathcal{J} \wedge I_0[i] = j\}, i+1) \\ \gamma(\mathcal{J}, i) &= \begin{cases} I_1, & if\ \exists I : I \in \mathcal{J} \wedge \|I_0\| = i \\ (), & otherwise \end{cases}\end{aligned} \tag{2-13}$$

RLP表示递归长度前缀（Recursive Length Prefix，RLP）编码[55]，HP表示 16 进制前缀编码，即将一个半字节序列编码为字节序列，编码算法如下：

$$\text{HP}(x,t) : x \in \mathbb{Y} \equiv \begin{cases} (16f(t), 16x[0] + x[1], 16x[2] + x[3], \dots) & if\ \|x\|/2 = 0 \\ (16(f(t)+1) + x[0], 16x[1] + x[2], 16x[3] + x[4], \dots) & otherwise \end{cases} \tag{2-14}$$

其中，$\mathbb{Y}$表示半字节序列，$f(t)$的定义如下：

$$f(t) = \begin{cases} 2 & if\ t \neq 0 \\ 0 & otherwise \end{cases} \tag{2-15}$$

与默克尔树类似，默克尔帕特里夏树也能够提供默克尔证明，即验证一个键值对$I$是否已被存储至默克尔帕特里夏树中，默克尔证明是待证明元素在默克尔帕特里夏树中的路径，验证函数如下：

$$f : (root, I_0, proof) \longrightarrow I_1' \tag{2-16}$$

验证者只需比较$I_1'$是否等于$I_1$，如果相等则验证通过，反之不通过，计算复杂度为$O(log\ N)$，$N$为存入默克尔帕特里夏树的键值对数量。

默克尔树与默克尔帕特里夏树的相同点在于两者都可以提供默克尔证明，从而以时间复杂度$O(log\ N)$完成关系证明，但默克尔帕特里夏树相较于默克尔树有以下优点：

1) 默克尔帕特里夏树具有搜索功能，默克尔树不具有；
2) 如果给定数据集合，则默克尔帕特里夏树的结构是固定的，但默克尔树的结构是随机的；
3) 默克尔帕特里夏树不会出现哈希碰撞，默克尔树有可能出现哈希碰撞。

### 2.3.3 交易

交易是区块链中非常重要的概念，区块链中交易与现实生活中的交易有所不同，并不仅限于买卖双方之间的利益交换（或金钱的转移），区块链中的交易是由外部账户发出的经过签名的消息，消息中可能包含代币的转移、对区块链中存储信息的修改等，交易通过区块链网络传播，最终由矿工记录在区块链上。以太坊网络使用洪泛（Flooding）路由协议[50]，交易的广播是从一个以太坊节点（发起节点）创建交易（或从离线设备接收）并签名开始的，交易通过验证后，会传送给所有跟这个节点直接相连的以太坊节点[52]，图 2-5 展示了交易的发起、验证

与传播过程。区块链可以视为一个基于交易的状态机，起始于一个创世区块，之后随着交易的执行状态逐步改变直至最终状态[45]，状态改变可以表示为：

$$\gamma(S_i, T) = S_{i+1} \tag{2-17}$$

其中，$\gamma$表示以太坊的状态计算引擎，$S_i$表示以太坊在第$i$个时刻的世界状态，$T$表示在从第$i$个时刻至第$i+1$个时刻的时间内打包的交易列表。

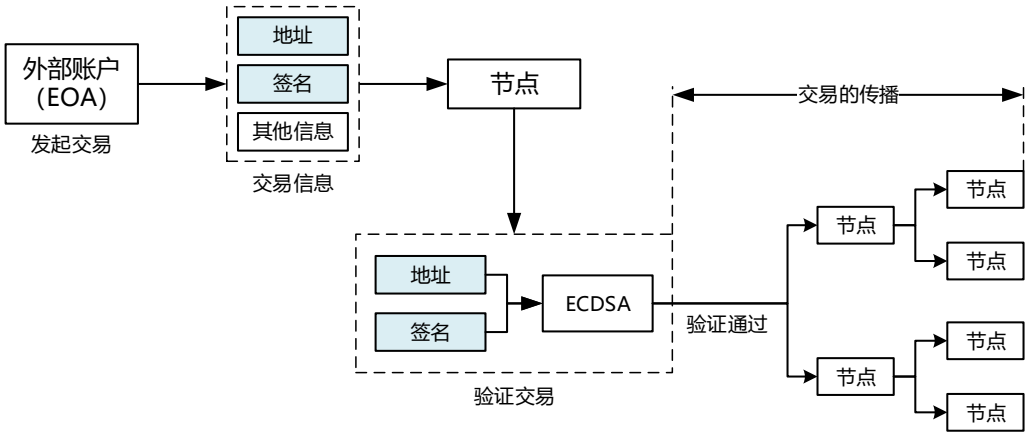



图 2-5 交易的发起、验证与传播示意图

Figure 2-5 Schematic diagram of transaction initiation, validation and propagation

在以太坊中，交易包含的所有字段以及每个字段的含义如下：

表 2-2 交易包含的字段以及每个字段的含义

Table 2-2 The fields contained in the transaction and the meaning of each field

| 字段名称 | 说明 |
|---|---|
| blockHash | 存储当前交易的区块的散列值。 |
| blockNumber | 存储当前交易的区块的高度（从 0 开始）。 |
| from | 交易发起方的地址。 |
| to | 目标以太坊的地址。 |
| gasPrice | 交易发起方愿意支付的 gas 的单位价格（单位：Wei）。 |
| gas | 当前交易需要支付的 gas。 |
| hash | 当前交易的散列值。 |
| input | 附在交易中的可变长度的数据。 |
| nonce | 地址 from 发出的交易数量，当这个地址与合约关联时，是这个地址所创建的合约数量。 |
| *v*、*r* 和 *s* | *v*、*r* 和 *s* 三个字段构成交易发起方的签名。 |
| transactionIndex | 当前交易在区块中的序号。 |
| value | 交易发起方发送给目标地址的以太币数量（单位：Wei）。 |

其中，nonce 字段等于交易发起方发出的交易数量，这种设计可以防止重放攻击。对于交易的验证可以划分为以下两个步骤：

1) 验证交易的签名是否合法；

2) 查看状态树中交易发起方的地址下是否有足够的余额以支持完成交易。

在以太坊中，使用椭圆曲线数字签名算法（Elliptic Curve Digital Signature Algorithm，ECDSA）[53]作为签名算法，并使用 secp256k1[54]作为 ECDSA 的椭圆曲线，哈希函数使用 Keccak-256，对交易$tx$的签名过程可以表示为：

$$sig = \mathrm{F}_{\mathrm{ECDSA}}\left(\mathrm{F}_{\mathrm{Keccak256}}\left(\mathrm{RLP}(tx)\right), sk\right) = (r, s) \tag{2-18}$$

首先对交易$tx$进行 RLP 编码，对编码后的交易计算哈希值，然后交易发起方使用一个临时私钥$sk$（一个密码学安全的随机数）对哈希值$\mathrm{F}_{\mathrm{Keccak256}}\left(\mathrm{RLP}(tx)\right)$进行签名，最终得到的签名由$r$与$s$两部分构成，其中，$r$为临时公钥$pk$在椭圆曲线中的横坐标，$s$的计算方法如下：

$$s \equiv q^{-1}(\mathcal{H}(m) + r \cdot k)(\mathrm{mod}\ p) \tag{2-19}$$

其中，$m$为消息数据包，$\mathcal{H}(\cdot)$表示哈希函数，$q$为临时私钥，$k$为外部账户的私钥，$p$为椭圆曲线的阶。

对于签名的验证，验证方必须要有签名$(r, s)$、交易$tx$以及对应着交易发起方签名私钥的公钥$pk$。公钥$pk$可以通过$(r, s)$进行恢复，签名的验证算法如下：

$$Q \equiv s^{-1} \cdot \mathcal{H}(m) \cdot G + s^{-1} \cdot r \cdot K\ (\mathrm{mod}\ p) \tag{2-20}$$

其中，$G$为椭圆曲线的生成点，$K$为外部账户对应的公钥，如果$Q$的横坐标等于$r$，则签名有效，否则无效。

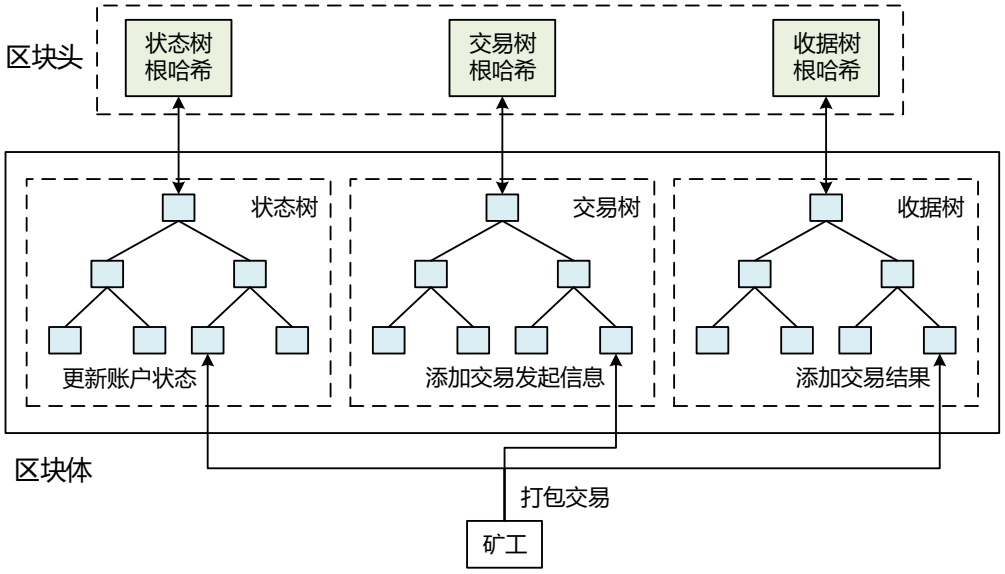



图 2-6 交易的存储过程

Figure 2-6 Storing procedures of transactions

在以太坊中，使用 RLP 编码技术对交易数据进行编码，每个区块都维护着三段以默克尔帕特里夏树为底层数据结构的存储空间，分别称为状态树（State Trie）、交易树（Transactions Trie）以及收据树（Receipts Trie），矿工将交易相关数据存储至此三段存储空间中，并且区块头存储着三者的根哈希值，图 2-6 展示了交易的存储过程，对于此三段存储空间的详细介绍如下[45]：

状态树。状态树存储所有账户的状态，即当前以太坊的世界状态，可以视为一个从账户地址到账户状态的映射表，可以使用账户地址查询对应的账户状态并提供默克尔证明。以太坊账户分为外部账户与合约账户，外部账户拥有私钥，可以控制对以太币以及合约的访问；合约账户具有合约代码，可以接收与发送以太币，但不具有私钥，所以无法启动交易[51]。图 2-7 展示了以太坊的账户结构，账户状态包括账户余额（balance，单位：Wei）、当前地址发起的交易数量（nonce）、账户对应的存储树的根哈希值（storageHash）以及 EVM 字节码的哈希值（codeHash）。以太坊将每个账户的存储内容组织为一个默克尔帕特里夏树，称为存储树（Storage Trie），并将存储树的根哈希值存入 storageHash 字段。对于 codeHash 字段，如果是外部账户，则为空字符串的哈希值，如果是合约账户，则为合约代码的哈希值。

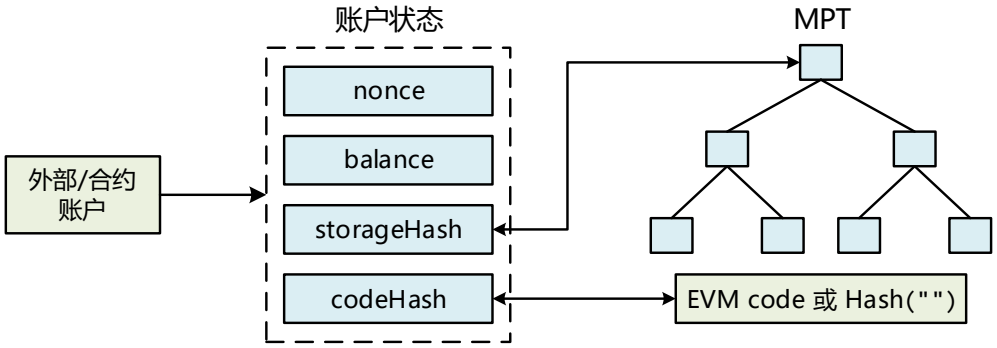



图 2-7 账户结构示意图

Figure 2-7 Account Structure Schematic

交易树。发布区块时，交易的请求或发起信息会被组织成一个交易树，可以使用交易在当前区块的交易列表中的编号查询此交易的请求信息并提供默克尔证明。

收据树。发布区块时，交易的执行状态或结果会被组织成一个收据树，可以使用交易在当前区块的交易列表中的编号查询此交易的执行状态信息并提供默克尔证明，另外，交易树和收据树的节点是一一对应的。

### 2.3.4 智能合约

智能合约的概念最早由密码学家 Nick Szabo 提出[56]，文献中将智能合约定义为一系列承诺，通过数字化的形式，包括一组协议与在协议中的各方执行的其他承诺，以太坊继承了这个概念，以太坊智能合约是运行在区块链中的一段代码（或驻留在区块中的脚本），代码的逻辑定义了合约的内容，但在以太坊的上下文中，智能合约其实既不智能，也不具有法律效力，目前适用于以太坊虚拟机的智能合约语言主要有 Solidity 与 Vyper 等。

本文使用智能合约代替中心化服务的业务逻辑，智能合约中的所有数据存储于此合约对应的合约账户的存储树中，存储树并不存储于链上，而是存储至链下的数据库（例如 LevelDB）中。

关于智能合约运行时间的计算。依据文献[60]，在以太坊中，无法获取合约的具体执行时间，只能得到以太坊当前区块的时间戳，这是因为在以太坊的实现规范中，不存在合约执行时间的概念，并且执行时间对于 EVM 而言是没有意义的。

关于智能合约的调用权限。依据文献[61]，在以太坊中，智能合约虽然是公开的，但不是所有方法都可以被外部调用，智能合约可以实现基于 ACL 的调用权限验证，发布者可以进行自主配置允许调用的账户地址列表，即 ACL，在智能合约被调用时，可以获得调用者的账户地址，根据账户地址是否在 ACL 中判断是否允许该调用者调用智能合约。

### 2.3.5 EVM 与 gas 消耗

以太坊虚拟机（Ethereum Virtual Machine，EVM）是以太坊协议和具体操作的核心[52]，EVM 是以太坊的计算引擎，并且是一个图灵完备[57]的状态机，以太坊中所有交易与合约（EVM 是 EVM 字节码的解释器）的执行都由 EVM 完成。EVM 中存在一个基于栈的架构，以太坊在其上完成所有的数据处理，包括 EVM 字节码所有的输入与输出以及中间执行过程中的局部变量等。EVM 的数据处理单位为 256 比特，称为“字”，这种设计的目的在于可以更加方便地执行哈希运算与椭圆曲线运算等操作，图 2-8 [58]展示了 EVM 的工作原理。

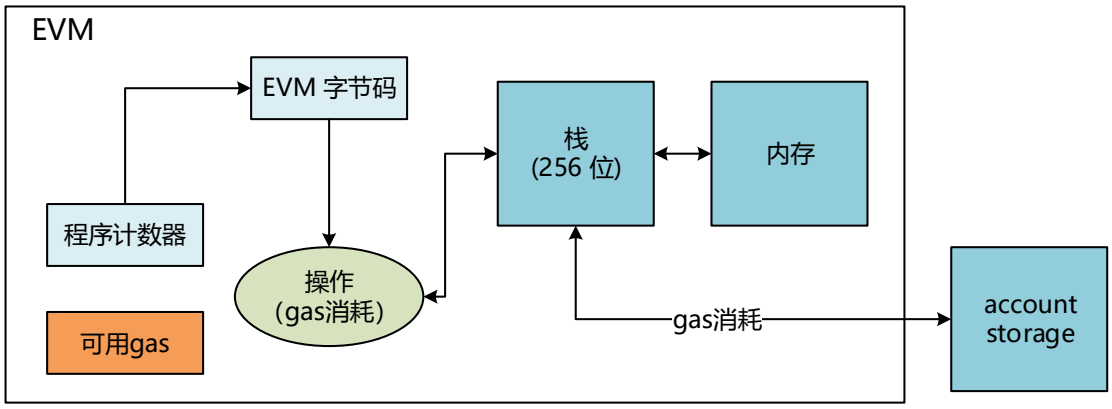



图 2-8 EVM 工作原理示意图

Figure 2-8 Schematic diagram of how EVM works

为了避免网络滥用及回避由于图灵完整性而带来的一些不可避免的问题，即避免可计算性理论中的停机问题（Halting Problem）[59]，在以太坊中所有的程序执行都需要支付费用，gas 消耗是某一操作消耗了多少链上计算资源和链上存储资源的衡量标准。

关于 gas 消耗。依据文献[62]，在智能合约中，不改变以太坊世界状态的方法称为常量方法，在执行这种方法时不需要发起交易，故执行这种方法时不消耗 gas，所以从链上合约中查询相应信息是不消耗 gas 的。这里的问题在于如何防止停机问题，在调用智能合约中的方法时，需要指定区块链网络中的一个节点，对于这种常量方法，区块链节点相当于一个提供计算或检索服务的节点，是会出现停机问题的，但只会影响接收请求的节点，对区块链网络中的其他节点不会造成影响，而且区块链节点可以自主选择是否对外提供上述服务。另外，EVM 内部对于一些常见的引发停机问题的漏洞具有防御措施，当然，为防止出现安全问题，也需要开发人员谨慎地编写智能合约。

## 2.4 简短非交互零知识证明（Zero-knowledge Succinct Non-interactive Arguments of Knowledge）

零知识证明（Zero-Knowledge Proof，ZKP）最早由 Goldwasser 等人在文献[63]中提出，将零知识证明定义为“除了给出命题的正确性之外，不会泄露其他任何信息的证明”。零知识证明并不是严格意义上的证明，而是一种概率证明，即证明者存在极小的概率可以欺骗验证者，但是这种概率小至可以忽略不计，零知识证明具有以下三个特性：

1) 完备性（Completeness）。如果命题是正确的，证明者可以说服验证者；
2) 健壮性（Soundness）。如果命题是错误的，证明者可以说服验证者的概率小至可以忽略不计；
3) 零知识性（Zero-knowledge）。验证者在证明过程中除了命题的正确性外不会获得其他任何信息。

最初的零知识证明方案是交互式的，即证明者与验证者需要多次交换信息以完成证明，随着挑战次数的增多，证明者可以欺骗验证者的概率将呈指数形式下降。简短性与非交互性是零知识证明的两个重要发展方向，对于零知识证明技术的一次提升是非交互零知识证明（Non-Interactive Zero-Knowledge Proof，NIZK）的出现，由 Blum 等人提出[64,65]，NIZK 能够在实现零知识性的同时，不需要证明者与验证者进行任何交互，只需要证明者与验证者之间共享一个公共参考字符串（Common Reference String，CRS）[66]模型。对于 NIZK 的一次提升是简短非交互零知识证明（Zero-Knowledge Succinct Non-interactive ARgument of Knowledge，zk-SNARK）的出现，由 Bitansky 等人提出[67]，zk-SNARK 的诞生是基于概率可检查性证明（Probabilistically Checkable Proofs，PCP）被证明[68]，zk-SNARK 在 NIZK 的非交互性基础上又引入了简短性，即证明者能够以一种简短的方式完成证明。

zk-SNARK 在区块链技术中有着广泛的应用，例如，以太坊以及 Zcash，ZeroCash 协议[70]是一个实现了 zk-SNARK 的隐私保护协议，在区块链中，所有账户余额都被加密存储至链上，当一个账户向其他账户发起转账交易时，通过 zk-SNARK，此账户能够证明其拥有足够的余额以完成转账，而不需要透露自身的账户余额。对于 NP 问题的简短非交互式证明一直是一个广受关注的问题，简短是指验证证明的计算复杂度与 NP 问题的复杂度无关[67]。目前，所有 NP 问题都可以使用 zk-SNARK 进行零知识证明，zk-SNARK 能够以一种通用的方式应用于所有 NP 问题，但对于 NP 问题之外的其他问题是否能够使用 zk-SNARK 进行零知识证明还尚未得到证明[79]。本节将详细介绍 zk-SNARK 的数学原理，并在后续章节中将 zk-SNARK 应用于物联网访问控制中，zk-SNARK 的工作流程由以下五部分构成：

1) 算术电路（Arithmetic Circuit）
2) 秩 1 约束系统（Rank-1 Constraint System，R1CS）
3) 二次算术程序（Quadratic Arithmetic Program，QAP）
4) 线性概率可检查证明（Linear Probabilistically Checkable Proof，LPCP）
5) 线性交互证明（Linear Interactive Proof）

图 2-9 展示了 zk-SNARK 的工作流程，本节将对上述五部分进行详细介绍，另外，本节的数学推导依据文献[80-83]。

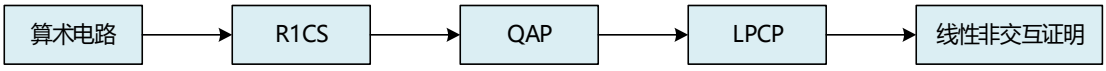



图 2-9 zk-SNARK 工作流程示意图

Figure 2-9 Workflow of zk-SNARK schematic

### 2.4.1 算术电路

zk-SNARK 的第一步是将待证明的问题编译为算术电路，从而将复杂的运算过程转化为离散的数学步骤，即将运算过程表示为只含有加法与乘法的算术形式，这个编译过程也被称为扁平化（Flattening）。算术电路由加法门与乘法门构成（这里的门通常指双线性门（Bilinear Gate），即输入数量为 2、输出数量为 1），算术电路对输入变量进行加法运算与乘法运算并输出结果，例如，图 2-10 表示$a_5 = (a_1 + a_2)(a_3 - a_4)$。

算术电路$C$的数学定义如下：

$$C: \mathbb{F}^n \times \mathbb{F}^h \longrightarrow \mathbb{F}^l \tag{2-21}$$

其中，$\mathbb{F}$表示有限域，$n$为验证方与证明方之间的共享输入总数，$h$为证明方的秘密输入总数，算术电路$C$的输入总数为$n+h$，$l$为算术电路$C$的输出总数，$C$的电路满足问题（Circuit Satisfaction Problem）[81]的数学定义如下：

$$\mathcal{L}_C = \{\vec{x} \in \mathbb{F}^n : \exists \vec{w} \in \mathbb{F}^h, C(\vec{x}, \vec{w}) = 0^l\} \tag{2-22}$$

对于算术电路在计算机系统中的具体实现，可以将算术电路中的加法门或乘法门表示为：

$$\phi = (*, a_3, a_1, a_2) \tag{2-23}$$

其中，$a_1$与$a_2$为输入，$a_3$为输出，$*$为门的运算符，即$a_1 * a_2 = a_3$，最终可以将算术电路表示为：

$$\Phi = \{\phi_i\}_{i\in[m]} \tag{2-24}$$

进而可以将算术电路存储至计算机系统中进行相关处理。在具体应用中，通常是使用高级程序设计语言将待证明的问题编写为相关代码，然后算术电路编译器将代码编译为算术电路。另外，zk-SNARK 还有一种基于布尔电路的实现方式，详见文献[79]，但是文献[82]已证明算术电路相较于布尔电路更加高效，并且算术电路对于模运算有更好的支持，算术电路的规模也通常比布尔电路小很多，所以 zk-SNARK 的实现方案中通常选择算术电路。

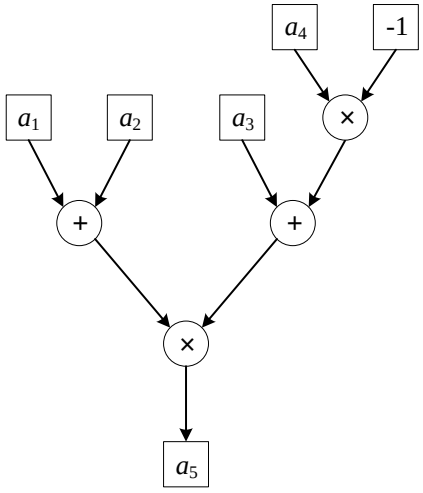



图 2-10 算术电路示意图

Figure 2-10 Arithmetic circuit schematic

## 2.4.2 秩 1 约束系统

目前，已将待证明的问题转化为算术电路，但算术电路并不是计算机系统能够直接处理的约束系统。zk-SNARK 的第二步是将算术电路转化为秩 1 约束系统（R1CS），转化方法是将算术电路中的每个逻辑门转化为一个秩 1 二次等式（Rank-1 Quadratic Equation，R1QE），然后所有秩 1 二次等式组合成最终的秩 1 约束系统。

在文献[83]中，对于秩 1 二次等式有如下数学定义：在有限域$\mathbb{F}$中，给定一个元组（Tuple）$s$，$s$可以表示为：

$$s = \left((\vec{a}, \vec{b}, \vec{c}), N, n\right) \tag{2-25}$$

其中，$\vec{a},\vec{b},\vec{c}\in\mathbb{F}^{1+N}$，$n\le N$，给定一个输入$\vec{x}\in\mathbb{F}^{n}$，如果存在一个见证值$\vec{w}\in\mathbb{F}^{N-n}$满足：

$$(\vec{a}\cdot(1,\vec{x},\vec{w}))\cdot\left(\vec{b}\cdot(1,\vec{x},\vec{w})\right)=(\vec{c}\cdot(1,\vec{x},,\vec{w})) \tag{2-26}$$

则$s(\vec{x},\vec{w})=1$，否则$s(\vec{x},\vec{w})=0$，其中，符号“·”表示向量的内积，所有秩 1 二次等式组合成一个完整的秩 1 约束系统$S$，$S$可以表示为：

$$S=\{s_k\}_{k\in[M]}=\left\{\left(\left(\overrightarrow{a_k},\overrightarrow{b_k},\overrightarrow{c_k}\right),N,n\right)\right\}_{k\in[M]} \tag{2-27}$$

其中，$M$为约束的数量，等于算术电路中逻辑门的数量，作如下定义：

$$S(\vec{x},\vec{w})=\begin{cases}1 & if\ \exists\vec{w}\in\mathbb{F}^{N-n}:\forall j\in[M]:s_j(\vec{x},\vec{w})=1\\ 0 & otherwise\end{cases} \tag{2-28}$$

其中，$S$可以表示为由矩阵$A$、$B$与$C$构成的一个约束方程，其中，$A,B,C\in\mathbb{F}^{M\times(1+N)}$，$\overrightarrow{a_k}$、$\overrightarrow{b_k}$与$\overrightarrow{c_k}$分别为矩阵$A$、$B$与$C$的第$k$行，为方便下文表述，本文定义$\vec{s}=(1,\vec{x},\vec{w})$，最终，$S$可以表示为：

$$(A\cdot\vec{s})\circ(B\cdot\vec{s})=C\cdot\vec{s} \tag{2-29}$$

其中，“∘”表示将两个矩阵相同位置的元素相乘，公式（2-29）是秩 1 约束系统的最终形式[84]，向量$\vec{s}$为约束方程的解，由算术电路中涉及到的所有变量构成，包括常量 1、输入、中间变量以及输出，所以$\vec{s}$是一个完整的计算状态，这一点是秩 1 约束系统与算术电路明显不同的地方。

对于如何将算术电路中的逻辑门$\phi$转化为秩 1 二次等式，为方便下文表述，本文定义$0^n$表示长度为$n$的零向量，$\Lambda$表示由$\vec{s}$中所有元素构成的集合，即$\Lambda=\{1,\vec{x}_1,\cdots,\vec{x}_n,\vec{w}_1,\cdots,\vec{w}_{N-n}\}$。秩 1 二次等式利用向量内积将算术电路中涉及到的任一数字$x$编码为一个向量$f(x)$，$f(x)$的数学定义如下：

$$x=f(x)\cdot\vec{s} \tag{2-30}$$

$$f(x)=\begin{cases}(0^{p-1},1,0^{N+1-p}), & x\in\Lambda\\ x(1,0^{N}), & x\notin\Lambda\end{cases} \tag{2-31}$$

其中，$p$为$x$在$\vec{s}$中的序号或者位置，$p\in[N+1]$，若$x\notin\Lambda$，则$x$为算术电路中不为 1 的常量，下面依据“$*$”表示加法或乘法两种情况介绍编码算法：

（1）“$*$”表示加法。通过$\vec{a}\cdot\vec{s}$表示$a_1+a_2$，$\vec{c}\cdot\vec{s}$表示$a_3$，并使得$\vec{b}\cdot\vec{s}=1$。此时，$(\vec{a}\cdot\vec{s})\times\left(\vec{b}\cdot\vec{s}\right)=(a_1+a_2)\times1=a_1+a_2=\vec{c}\cdot\vec{s}=a_3$，$\vec{a}$、$\vec{b}$及$\vec{c}$的求解方法如下：

$$\begin{cases}\vec{a}=f(a_1)+f(a_2)\\ \vec{b}=(1,0^{N})\\ \vec{c}=f(a_3)\end{cases} \tag{2-32}$$

（2）“$*$”表示乘法。通过$\vec{a}\cdot\vec{s}$表示$a_1$，$\vec{b}\cdot\vec{s}$表示$a_2$，$\vec{c}\cdot\vec{s}$表示$a_3$。此时，$(\vec{a}\cdot\vec{s})\times\left(\vec{b}\cdot\vec{s}\right)=a_1\times a_2=\vec{c}\cdot\vec{s}=a_3$，$\vec{a}$、$\vec{b}$及$\vec{c}$的求解方法如下：

$$\begin{cases}\vec{a}=f(a_1)\\ \vec{b}=f(a_2)\\ \vec{c}=f(a_3)\end{cases} \tag{2-33}$$

### 2.4.3 二次算术程序

二次算术程序（QAP）是 zk-SNARK 实现简短性的关键，二次算术程序的目标是将秩 1 约束系统转化为更容易验证的多项式等式。文献[85]对二次算术程序有如下定义：在有限域$\mathbb{F}$中的二次算术程序$\mathcal{Q}$包含三个多项式集合$\mathcal{V}$、$\mathcal{W}$和$\mathcal{Y}$，以及一个多项式因子$\mathcal{D}(x)$，可以表示为：

$$\mathcal{V}=\{v_k(x):k\in[m]\} \tag{2-34}$$

$$\mathcal{W}=\{w_k(x):k\in[m]\} \tag{2-35}$$

$$\mathcal{Y}=\{y_k(x):k\in[m]\} \tag{2-36}$$

$$\mathcal{D}(x)=\prod_{i=1}^{m+1}(x-i) \tag{2-37}$$

其中，$\mathcal{Q}$中所有多项式皆属于$\mathbb{F}[x]$，定义函数$f:\vec{x}\longrightarrow\vec{y}$，其中，$\vec{x}\in\mathbb{F}^p$，$\vec{y}\in\mathbb{F}^q$，$\vec{x}$与$\vec{y}$可以表示为：

$$\begin{cases}\vec{x}=(a_1,a_2,\cdots,a_p)\\ \vec{y}=(a_{m-q+1},a_{m-q+2},\cdots,a_m)\end{cases} \tag{2-38}$$

如果存在$\vec{w}=(a_{p+1},a_{p+2},\cdots,a_{m-q})\in\mathbb{F}^{m-p-q}$满足：

$$\left(v_0(x)+\sum_{k=1}^{m}a_k\cdot v_k(x)\right)\cdot\left(w_0(x)+\sum_{k=1}^{m}a_k\cdot w_k(x)\right)-\left(y_0(x)+\sum_{k=1}^{m}a_k\cdot y_k(x)\right)=\mathrm{H}(x)\cdot\mathcal{D}(x) \tag{2-39}$$

则$\mathcal{Q}$为基于函数$f$的二次算术程序，其中，$\mathrm{H}(x)\in\mathbb{F}[x]$，公式（2-39）表示$\mathcal{D}(x)$能够整除由$\mathcal{V}$、$\mathcal{W}$与$\mathcal{Y}$组合而成的多项式。

$$P_A(x)\cdot P_B(x)-P_C(x)=\mathrm{H}(x)\cdot\mathcal{D}(x) \tag{2-40}$$

二次算术程序的目标是将秩 1 约束系统转化为如公式（2-40）所示的多项式等式，其中，多项式$P_A(x)$、$P_B(x)$与$P_C(x)$分别由$A\cdot\vec{s}$、$B\cdot\vec{s}$与$C\cdot\vec{s}$计算而来。将秩 1 约束系统转化为二次算术程序的方法是将秩 1 约束系统中涉及的矩阵$A$、$B$与$C$编码为 3 个多项式集合，即上文中的$\mathcal{V}$、$\mathcal{W}$与$\mathcal{Y}$。图 2-11 展示了将一个 4 行 6 列的矩阵编码为多项式集合$\mathcal{P}=\{P_i:i\in[6]\}$的示意图，为方便表示，将该矩阵记为$A$，从图中可以看出二次算术程序将$A$的每一列编码为一个多项式，并且$A$与$\mathcal{P}$之间满足如下关系：

$$A_{i,j}=P_j(i) \tag{2-41}$$

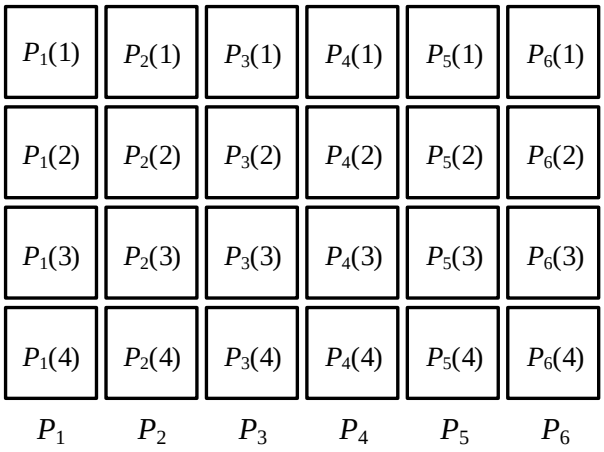



图 2-11 R1CS 矩阵编码示意图

Figure 2-11 R1CS matrix coding schematic

本小节使用基于范德蒙矩阵的方法求解上述多项式集合，除此之外还有基于拉格朗日插值法与傅里叶变换的计算方法，令$A \in \mathbb{F}^{m\times n}$，$\vec{s} \in \mathbb{F}^{n}$，依据一个$n-1$阶多项式能够表示为：

$$\begin{aligned} P(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1}x^{n-1} \\ &= |1 \quad x \quad x^2 \quad \cdots \quad x^{n-1}| \cdot |a_0 \quad a_1 \quad a_2 \quad \cdots \quad a_{n-1}|^T \end{aligned} \tag{2-42}$$

则可以将向量$A \cdot \vec{s}$中的各个数值视为多项式$P_A(x)$从$x=1$至$x=m$的取值，从而能够推导出多项式$P_A(x)$应满足：

$$\begin{vmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2^2 & \cdots & 2^{m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & m & m^2 & \cdots & m^{m-1} \end{vmatrix} \cdot \begin{vmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{m-1} \end{vmatrix} = \begin{vmatrix} P_A(1) \\ P_A(2) \\ \vdots \\ P_A(m) \end{vmatrix} = A \cdot \vec{s} \tag{2-43}$$

为方便后续表述，本文做如下定义：

$$V_m = \begin{vmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2^2 & \cdots & 2^{m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & m & m^2 & \cdots & m^{m-1} \end{vmatrix} \tag{2-44}$$

$$\boldsymbol{x}_m = |1 \quad x \quad x^2 \quad \cdots \quad x^{m-1}| \tag{2-45}$$

其中，$V_m$为范德蒙矩阵，由于范德蒙矩阵为可逆矩阵，可以推导出$P_A(x)$各项系数的求解方法如下：

$$|a_0 \quad a_1 \quad a_2 \quad \cdots \quad a_{n-1}|^T = V_m^{-1} \cdot A \cdot \vec{s} \tag{2-46}$$

其中，$V_m^{-1}$为$V_m$的逆矩阵，$V_m^{-1} \cdot A$为所有多项式的系数构成的矩阵，二次算术程序将$A$编码为如公式（2-46）所示的多项式集合。多项式$P_A(x)$的最终求解方法如公式（2-48）所示。

$$\mathcal{P} = \left\{ P_i(x) = \sum_{k=1}^{m} (V_m^{-1} \cdot A)_{k,i} \cdot x^{k-1} : i \in [n] \right\} \tag{2-47}$$

$$P_A(x) = \boldsymbol{x}_m \cdot V_m^{-1} \cdot A \cdot \vec{s} \tag{2-48}$$

多项式$P_B(x)$与多项式$P_A(x)$的求解方法相同，$P_B(x)$可以表示为：

$$P_B(x) = \boldsymbol{x}_m \cdot V_m^{-1} \cdot B \cdot \vec{s} \tag{2-49}$$

但多项式$P_C(x)$的求解方法与多项式$P_A(x)$、$P_B(x)$略有不同，根据多项式$P_A(x)$、$P_B(x)$为$m-1$阶多项式，故多项式$P_C(x)$应为$2m-2$阶多项式，求解多项式$P_C(x)$的所有系数需要$2m-1$个取值点，但是$C \cdot \vec{s}$只能提供$m$个取值点，无法求解多项式$P_C(x)$的所有系数，解决方法是使用多项式$P_A(x)$、$P_B(x)$计算出多项式$P_C(x)$额外的$m-1$个取值点，为方便后续表述，本文做如下定义：

$$\begin{aligned}\vec{\zeta} &= \left(P_C(m+1), P_C(m+2), \cdots, P_C(2m-1)\right) \\ &= \left(P_A(m+1)P_B(m+1), P_A(m+2)P_B(m+2), \cdots, P_A(2m-1)P_B(2m-1)\right)\end{aligned} \tag{2-50}$$

多项式$P_C(x)$应满足：

$$V_{2m-1} \cdot \begin{vmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \\ c_m \\ \vdots \\ c_{2m-1} \end{vmatrix} = \begin{vmatrix} P_C(1) \\ P_C(2) \\ \vdots \\ P_C(m) \\ \vec{\zeta} \end{vmatrix} = \begin{vmatrix} C \cdot \vec{s} \\ \vec{\zeta} \end{vmatrix} \tag{2-51}$$

多项式$P_C(x)$的最终求解方法如下：

$$P_C(x) = \boldsymbol{x}_{2m} \cdot V_{2m}^{-1} \cdot \begin{vmatrix} C \cdot \vec{s} \\ \vec{\zeta} \end{vmatrix} \tag{2-52}$$

所有多项式的求解方法整理如下：

$$\begin{cases} P_A(x) = \boldsymbol{x}_m \cdot V_m^{-1} \cdot A \cdot \vec{s} \\ P_B(x) = \boldsymbol{x}_m \cdot V_m^{-1} \cdot B \cdot \vec{s} \\ P_C(x) = \boldsymbol{x}_{2m} \cdot V_{2m}^{-1} \cdot \begin{vmatrix} C \cdot \vec{s} \\ \vec{\zeta} \end{vmatrix} \end{cases} \tag{2-53}$$

### 2.4.4 椭圆曲线与双线性配对

由于椭圆曲线与双线性配对的数学原理极为复杂，本小节只介绍与本文相关的部分，关于椭圆曲线与双线性配对的详细介绍可参阅文献[86-88]。

对于双线性配对的数学定义依据文献[87]，双线性配对$e$可以表示为：

$$e: G_1 \times G_2 \longrightarrow G_T \tag{2-54}$$

其中，$G_1$与$G_2$为阶数为$q$的加法循环群，$G_T$为循环群，所有运算都在基于素数$q$的有限域$\mathbb{F}_q$内完成，$e$具有以下性质：

1) 双线性：对于$\forall a, b \in \mathbb{F}_q^*$，$\forall P \in G_1$，$\forall Q \in G_1$，满足：

$$e(aP, bQ) = e(P, Q)^{ab} \tag{2-55}$$

2) 非退化性：$e \neq 1$；

3) 可计算性：存在有效的算法计算$e$。

对于在有限域$\mathbb{F}_p$上的椭圆曲线$\mathcal{E}$，双线性配对$e$的求解方法如下[88]：

$$e(P,Q)=\frac{f_P(Q+S)\,/\,f_P(S)}{f_Q(P-S)\,/\,f_Q(-S)} \tag{2-56}$$

其中，$\mathcal{O}$表示无穷远点，$P,Q\in\mathcal{E}[m]$，$\mathcal{E}[m]=\{X:X\in\mathcal{E}\wedge m\cdot X=\mathcal{O}\}$，$S\in\mathcal{E}$，$S\notin\{\mathcal{O},P,-Q,P-Q\}$，$f_P(x,y)=x-P_x$，$f_Q(x,y)=x-Q_x$，$P_x$与$Q_x$分别表示$P$与$Q$的横坐标。

在椭圆曲线加密系统中，假设椭圆曲线的生成点为$G$，如果已知$x$与$y$的加密结果$x\cdot G$和$x\cdot G$，$xy$的加密结果$xy\cdot G$，可以通过公式（5-37）验证$xy\cdot G$是否由$xy$加密而来。

$$e(x\cdot G,y\cdot G)=e(xy\cdot G,G) \tag{2-57}$$

### 2.4.5 线性概率可检查证明

LPCP 定理是 zk-SNARK 能够实现简短性的数学基础，PCP 定理的定义为所有的 NP 问题都可以在多项式时间内通过概率验证的方法证明[68]，zk-SNARK 基于线性 PCP（Linear PCP，LPCP）定理，LPCP 定理是 PCP 定理的子集，在 LPCP 定理中，证明$\pi$为有限域内的一个向量，并且只能对$\pi$进行线性运算[69]。

LPCP 定理指出并不需要真正计算等式（2-40）是否成立，只需选取一个随机点代入等式（5-20），检查等式是否成立即可，这种方法能够同时验证所有约束，证明方可以欺骗验证方的概率$P$如下：

$$P=\frac{m}{\left|\mathbb{F}_q\right|} \tag{2-58}$$

其中，$m$为多项式的阶数，$\left|\mathbb{F}_q\right|$表示有限域$\mathbb{F}_q$的大小，即$\mathbb{F}_q$所包含的元素总数，$q$通常是一个极大的数字，所以$\left|\mathbb{F}_q\right|$也是一个极大的数字，故概率$P$可以忽略不计。

为方便后续表述，本文作如下定义：

1) $r$代表随机检查点；

2) $\boldsymbol{r}_m=|r^0\quad r^1\quad \ldots\quad r^{m-1}|$。

在公式（2-53）中，$V_m$、$V_{2m}$、$A$、$B$以及$C$是验证方与证明方之间共享的数据，$\vec{s}$与$\vec{\zeta}$为证明方的输入，抽取出公式（2-52）中验证方的事先已知部分，作如下定义：

$$\begin{cases}\overrightarrow{q_A}=\boldsymbol{r}_m\cdot V_m^{-1}\cdot A\\ \overrightarrow{q_B}=\boldsymbol{r}_m\cdot V_m^{-1}\cdot B\\ \overrightarrow{q_C}=\boldsymbol{r}_{2m}\cdot V_{2m}^{-1}\cdot C'\end{cases} \tag{2-59}$$

其中，$C'$可以表示为：

$$C'=\begin{vmatrix}C & \boldsymbol{O}\\ \boldsymbol{O} & I_m\end{vmatrix} \tag{2-60}$$

将公式（2-53）转化为如下形式：

$$\begin{cases} P_A(x) = \overrightarrow{q_A} \cdot \vec{s} \\ P_B(x) = \overrightarrow{q_B} \cdot \vec{s} \\ P_C(x) = \overrightarrow{q_C} \cdot \overrightarrow{s'} \end{cases} \tag{2-61}$$

其中，$\overrightarrow{s'} = (1, \vec{x}, \vec{w}, \vec{\zeta})$，证明方只需事先计算出$\overrightarrow{s'}$交由验证方即可，然后，验证方随机选取一个检查点$r$，计算出$\overrightarrow{q_A}(r)$、$\overrightarrow{q_B}(r)$与$\overrightarrow{q_C}(r)$，验证公式（2-62）是否成立即可，如果成立，验证通过；反之不通过。

$$\left((\overrightarrow{q_A}(r), 0^{m-1}) \cdot \overrightarrow{s'}\right) \cdot \left((\overrightarrow{q_B}(r), 0^{m-1}) \cdot \overrightarrow{s'}\right) - \left(\overrightarrow{q_C}(r) \cdot \overrightarrow{s'}\right) = 0 \tag{2-62}$$

上述过程还可以进一步优化，由于验证者已知$(1, \vec{x})$，所以可以把$\overrightarrow{s'}$分割为两部分$(1, \vec{x})$和$(\vec{w}, \vec{\zeta})$，证明者只提供$(\vec{w}, \vec{\zeta})$即可，分割方法如下：

$$\vec{q} \cdot \overrightarrow{s'} = \overrightarrow{q^L} \cdot (1, \vec{x}) + \overrightarrow{q^R} \cdot (\vec{w}, \vec{\zeta}) \tag{2-63}$$

定义$\overrightarrow{w'} = (\vec{w}, \vec{\zeta})$，验证方将$\overrightarrow{q_A^R}(r)$、$\overrightarrow{q_B^R}(r)$和$\overrightarrow{q_C^R}(r)$发送给证明方，证明方根据公式（2-64）进行计算，并将计算结果$a$、$b$与$c$发送给验证方。

$$\begin{cases} a = \left(\overrightarrow{q_A^R}(r), 0^{m-1}\right) \cdot \overrightarrow{w'} \\ b = \left(\overrightarrow{q_B^R}(r), 0^{m-1}\right) \cdot \overrightarrow{w'} \\ c = \overrightarrow{q_C^R}(r) \cdot \overrightarrow{w'} \end{cases} \tag{2-64}$$

验证方只需验证公式（2-65）是否成立即可。

$$\left(\overrightarrow{q_A^L}(r) \cdot (1, \vec{x}) + a\right)\left(\overrightarrow{q_B^L}(r) \cdot (1, \vec{x}) + b\right) = \left(\overrightarrow{q_C^L}(r) \cdot (1, \vec{x}) + c\right) \tag{2-65}$$

但上述过程有一个严重漏洞，如果证明方知晓$\overrightarrow{q_A^R}(r)$、$\overrightarrow{q_B^R}(r)$和$\overrightarrow{q_C^R}(r)$的取值，那么证明方就可以解出随机抽查点$r$，从而伪造出满足公式（2-65）的$a$、$b$以及$c$。验证方希望证明方能够在不知晓$\overrightarrow{q^R}$的前提下，计算$\overrightarrow{q^R} \cdot \overrightarrow{w'}$，$\overrightarrow{q^R} \cdot \overrightarrow{w'}$是一个线性计算，可以表示为：

$$\overrightarrow{q^R} \cdot \overrightarrow{w'} = a_0 w_0' + a_1 w_1' + \cdots + a_{n-1} w_{n-1}' \tag{2-66}$$

此时，需要一种满足线性运算的加密算法，并且可以验证加密结果的乘积是否相等，而椭圆曲线加密算法满足这两个要求。

为方便后续表述，本文使用指数形式表示椭圆曲线加密，令$g$表示椭圆曲线的生成点，当验证方给出$\overrightarrow{q^R}$的加密结果$\{g^{a_i} : i \in \{0,1,\ldots,n-1\}\}$时，则证明者能够依据公式（2-67）计算出$g^{\overrightarrow{q^R} \cdot \overrightarrow{w'}}$。

$$g^{\overrightarrow{q^R} \cdot \overrightarrow{w'}} = \prod_{i=0}^{n-1} (g^{a_i})^{w_i'} \tag{2-67}$$

为方便后续表述，本文定义$[\![\vec{x}]\!]$表示由$\vec{x}$所有元素的加密结果构成的集合，此时，验证方将公式（2-68）所示的加密数据发送给证明方。

$$\left\{\left[\left[\overrightarrow{q_A^R}(r)\right]\right],\left[\left[\overrightarrow{q_B^R}(r)\right]\right],\left[\left[\overrightarrow{q_C^R}(r)\right]\right]\right\} \tag{2-68}$$

证明方依据公式（2-69）进行计算，并将计算结果发送给验证方。

$$\left\{A=g^{\left(\overrightarrow{q_A^R}(r),0^{m-1}\right)\cdot\overrightarrow{w'}},B=g^{\left(\overrightarrow{q_B^R}(r),0^{m-1}\right)\cdot\overrightarrow{w'}},C=g^{\overrightarrow{q_C^R}(r)\cdot\overrightarrow{w'}}\right\} \tag{2-69}$$

验证方只需验证公式（2-70）是否成立即可。

$$e\left(A\cdot g^{\overrightarrow{q_A^L}\cdot(1,\vec{x})},B\cdot g^{\overrightarrow{q_B^L}\cdot(1,\vec{x})}\right)=e\left(C\cdot g^{\overrightarrow{q_C^L}\cdot(1,\vec{x})},g\right) \tag{2-70}$$

其中，$e$表示椭圆曲线双线性配对，详见小节（2.4.4）。

但上述过程仍存在一个严重漏洞，证明方可以不使用验证方提供的加密数据进行计算，而是直接伪造一组满足要求的$(A,B,C)$发送给验证者，所以验证方还要能够检验证明方提供的计算结果是否由加密数据计算而来，解决方法是验证方选取随机数$\alpha$、$\beta$与$\gamma$，还需将如公式（2-71）所示的加密数据发送给证明方。

$$\left\{\left[\left[\alpha\overrightarrow{q_A^R}(r)\right]\right],\left[\left[\beta\overrightarrow{q_B^R}(r)\right]\right],\left[\left[\gamma\overrightarrow{q_C^R}(r)\right]\right]\right\} \tag{2-71}$$

证明方依据公式（2-72）进行计算，并将计算结果发送给验证方。

$$\left\{A'=g^{\left(\alpha\overrightarrow{q_A^R}(r),0^{m-1}\right)\cdot\overrightarrow{w'}},B'=g^{\left(\beta\overrightarrow{q_B^R}(r),0^{m-1}\right)\cdot\overrightarrow{w'}},C'=g^{\gamma\overrightarrow{q_C^R}(r)\cdot\overrightarrow{w'}}\right\} \tag{2-72}$$

验证方只需验证公式（2-73）至公式（2-75）是否成立，即可验证$(A,B,C)$是否由加密数据（如公式（2-68）所示）计算而来。

$$e(A,g^{\alpha})=e(A',g) \tag{2-73}$$

$$e\left(B,g^{\beta}\right)=e(B',g) \tag{2-74}$$

$$e(C,g^{\gamma})=e(C',g) \tag{2-75}$$

但上述过程仍有漏洞，证明方可以在计算中途使用不同的$\overrightarrow{w'}$，从而使计算结果通过验证，解决方法进行随机线性检查，令$\overrightarrow{q^*}=\alpha\overrightarrow{q_A^R}+\beta\overrightarrow{q_B^R}+\gamma\overrightarrow{q_C^R}$，验证方还需将如公式（2-76）所示的加密数据发送给证明方。

$$\left\{\left[\left[\overrightarrow{q^*}\right]\right]\right\} \tag{2-76}$$

证明方依据公式（2-77）进行计算，并将计算结果发送给验证方。

$$\left\{D=g^{\overrightarrow{q^*}\cdot\overrightarrow{w'}}\right\} \tag{2-77}$$

验证方只需验证公式（2-78）是否成立，即可判断$(A,B,C)$是否由同一个$\overrightarrow{w'}$计算而来。

$$e(D,g)=e(A'B'C',g) \tag{2-78}$$

最后，对于零知识性，实现方法是证明方在计算编码多项式时多添加一个随机数，将原有的$m-1$阶多项式转化为$m$阶多项式[80]。

上述即为完整的简短零知识证明过程。

### 2.4.6 线性非交互证明

zk-SNARK 使用 CRS 模型实现非交互性，CRS 模型是证明者与验证者共享的变量空间，此变量空间中的变量是随机生成的，变量的取值通常服从某个概率分布，CRS 模型中变量的生成过程通常被称为可信初始化，zk-SNARK 最终可以定义为如下形式：

$$\text{KeyGen: } \left(F, 1^{\lambda}\right) \longrightarrow \left(key_p, key_v\right) \tag{2-79}$$

$$\text{Proof: } \left(key_P, \overrightarrow{w'}\right) \longrightarrow \pi \tag{2-80}$$

$$\text{Verify: } \left(key_V, \pi\right) \longrightarrow \{0,1\} \tag{2-81}$$

其中，KeyGen表示可信初始化函数，$F$表示需进行零知识证明的问题逻辑，$\lambda$表示安全系数，$key_P$表示证明密钥，如公式（2-82）所示，证明方使用$key_P$生成证明$\pi$，$key_V$表示验证密钥，如公式（2-83）所示，验证方使用$key_V$验证证明方提供的证明$\pi$，证明$\pi$如公式（2-84）所示；Proof表示证明生成函数，证明方使用函数Proof生成证明，$\overrightarrow{w'}$表示见证值，即证明方的秘密输入；Verify表示验证函数，验证方使用函数Verify验证证明方提供的证明$\pi$。

$$key_P = \left\{\left[\left[\overrightarrow{q_A^R}\right]\right], \left[\left[\overrightarrow{q_B^R}\right]\right], \left[\left[\overrightarrow{q_C^R}\right]\right], \left[\left[\alpha\overrightarrow{q_A^R}\right]\right], \left[\left[\beta\overrightarrow{q_B^R}\right]\right], \left[\left[\gamma\overrightarrow{q_C^R}\right]\right], \left[\left[\overrightarrow{q^*}\right]\right]\right\} \tag{2-82}$$

$$key_V = \left\{\left[\left[\overrightarrow{q_A^L} \cdot (1, \vec{x})\right]\right], \left[\left[\overrightarrow{q_B^L} \cdot (1, \vec{x})\right]\right], \left[\left[\overrightarrow{q_C^L} \cdot (1, \vec{x})\right]\right], g^{\alpha}, g^{\beta}, g^{\gamma}\right\} \tag{2-83}$$

$$\pi = \{A, B, C, A', B', C', D\} \tag{2-84}$$

# 3 基于链上计算与存储的物联网访问控制研究

# 3 Research on IoT Access Control Based on On-chain Computing and Storage

传统物联网访问控制模型往往将核心功能委托至可信第三方，对于传统模型，一方面，系统性能依赖于中央服务，另一方面，模型使用者的隐私信息被可信第三方中心化管理，存在隐私问题，区块链技术为物联网领域的发展引入了一个契机，基于链上数据的公开透明性与不可篡改性，区块链技术能够在不需要可信第三方的条件下有效保证物联网数据的完整性与安全性，将两者结合的主要方式是链上存储与链上存证[72]。

综上所述，本章基于链上存储的方式，构建了基于智能合约的物联网访问控制模型，并将其命名为 IoTAC，IoTAC 不需要可信第三方的支持，另外，本章对将物联网数据存储至链上的不同实现方式进行了介绍与比较，为后续访问控制模型的具体实现提供参考依据。

## 3.1 区块链数据存储及分析（Blockchain Data Storage and Analysis）

对于以太坊此类公链网络，链上存储资源是非常昂贵的，并且链上存储资源要比链上计算资源更加昂贵[73]，所以在将物联网数据存储至链上之前，应当对数据存储的方式进行精心选择与设计，数据存储方式主要涉及数据类型与存储方式两方面。在以太坊中，将数据存储至链上，主要有两种途径，一种是将数据存储至交易中，交易发起方能够在交易中附加一些数据，但对于这种方式，数据难以维护，不便于后续使用；另一种是将数据存储至智能合约中，在智能合约中创建相关变量以存储数据，对于这种方式，能够通过调用合约与链上数据进行交互，所以通常使用智能合约存储数据。

### 3.1.1 数据类型选择

对于数据类型的选择，有以下两个要求：

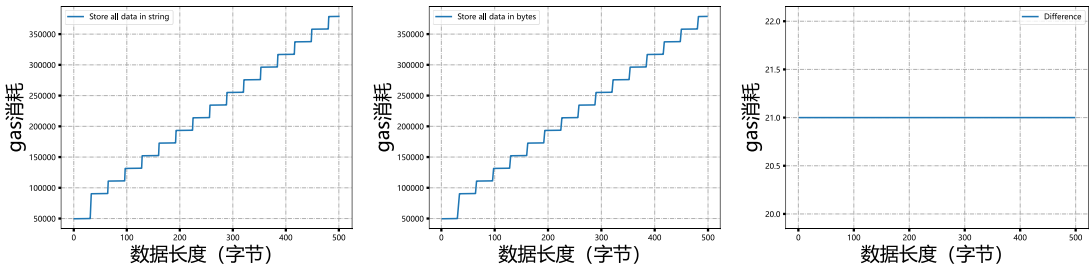
1) 对于链上存储资源的占用应当尽可能少；
2) 对于实际环境应当有较好的适用性。

本小节根据待存储数据的长度是否事先已知对数据类型的选择进行阐述：

如果待存储数据的长度事先是未知的，对于智能合约语言 Solidity，主要有两种数据类型的选择，分别为 string 类型与 bytes 类型。在以太坊中，因为 EVM 在处理 string 类型数据时需要完成编/解码运算（EVM 默认使用 UTF-8 编码），即将 string 类型数据转化为字节序列，所以相较于 string 类型，bytes 类型更贴近

于 EVM 的底层设计，EVM 处理 bytes 类型数据要略微高效，但是编/解码运算的计算开销很小，所以使用 string 类型或 bytes 类型皆可。

在以太坊中，衡量链上资源的存储性能主要有两个维度，一是存储数据所消耗的时间，二是存储数据所消耗的 gas，对于时间消耗详见小节（3.1.3），对于 gas 消耗，小节（2.3.5）介绍 gas 消耗是某一操作消耗了多少链上计算资源和链上存储资源的衡量标准。本小节对将数据以 string 类型与 bytes 类型存储的 gas 消耗进行了实验，在智能合约中分别创建 string 类型与 bytes 类型的动态数组，在实验中，将数据长度从 1 byte 逐渐递增至 500 byte，依次添加至每种类型的动态数组中，并记录每次添加的 gas 消耗，实验结果如下所示：




(a) string 类型　　(b) bytes 类型　　(c) gas 消耗差值

图 3-1 将数据以 string 类型或 bytes 存储至智能合约的 gas 消耗

Figure 3-1 Store data as string types or bytes to the smart contract's gas consumption

从图中可以看出，将数据以 string 类型与 bytes 类型存储的 gas 消耗几乎相等，string 类型仅比 bytes 类型多消耗 21 gas。另外，如果数据结构为嵌套的结构体类型，即数据能够表示为 JSON、YML 及 XML 等标记语言格式，则可以将数据表示为嵌套键值对数组形式，然后对数据进行 RLP 编码，使数据更加紧凑，从而节约链上存储资源。

如果待存储数据的长度事先是已知的，应当使用 bytes32 类型存储数据。在 Solidity 语言中，一个 bytes32 类型变量可以存储一个长度为 32 字节的字符串。小节（2.3.5）已阐述 EVM 的数据处理单位为 32 字节，EVM 对于 bytes32 类型有更好的支持，故将数据存储为 bytes32 类型的 gas 消耗要小于 string 类型与 bytes 类型。

本小节对将数据以 bytes32 类型存储进行了测试实验，数据长度从 32 byte 逐渐递增至 512 byte，每次增加 32 byte，并与 string 类型与 bytes 类型进行比较，实验结果如图 3-2 所示，从图中可以看出，使用 bytes32 类型存储数据的 gas 消耗要明显小于 string 类型与 bytes 类型[①]，约节省 17000~22000 gas，并且从图中还能够发现将数据以 string 和 bytes 存储至链上时，gas 消耗是以 32 byte 为单位进行计算的。

① 在图 3-2 中，string 类型和 bytes 类型对应的曲线几乎重合

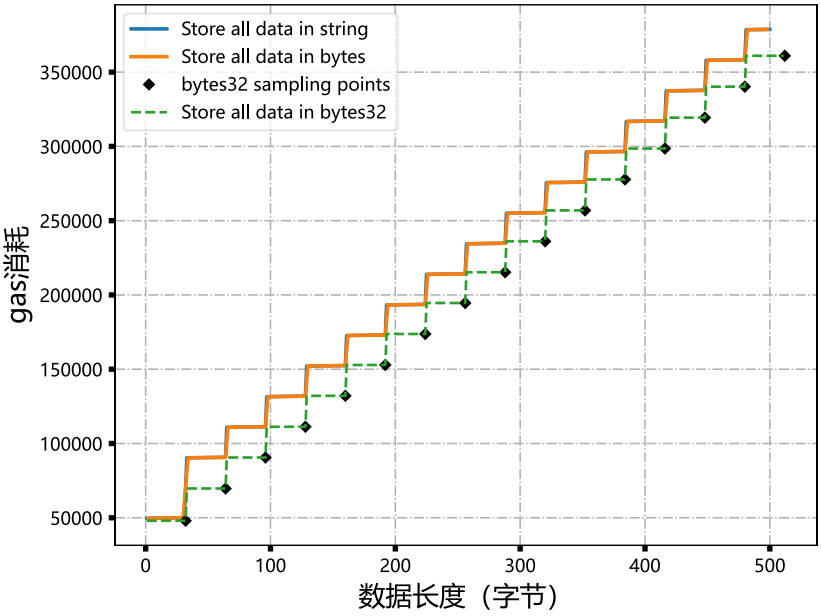



图 3-2 将数据以不同数据类型存储至智能合约的 gas 消耗

Figure 3-2 Gas consumption for storing data in different data types to smart contracts

### 3.1.2 存储方式选择

依据文献[73,74]，将数据存储至链上的存储方式主要分为全量存储、哈希存储以及根哈希存储，下面分别对上述三种方式进行介绍：

全量存储。全量存储是将所有数据存储至链上，全量存储的步骤是首先对数据$d_i$进行 RLP 编码，然后将编码后的数据添加至智能合约中的 bytes 类型动态数组中，最终存储至链上的数据集合$\mathcal{D}_A$可以表示为：

$$\mathcal{D}_A = \{\mathrm{RLP}(d_i): i \in \{0,1,\dots\}\} \tag{3-1}$$

全量存储适合存储长度较小的数据，存储长度较大的数据至链上开销巨大、极其昂贵。

哈希存储。哈希存储是只将数据$d_i$的哈希值$\mathcal{H}(d_i)$存储至链上，而数据$d_i$仍存储至链下，最终存储至链上的数据集合$\mathcal{D}_{\mathcal{H}}$可以表示为：

$$\mathcal{D}_{\mathcal{H}} = \{\mathcal{H}(d_i): i \in \{0,1,\dots\}\} \tag{3-2}$$

哈希存储适合存储长度较大的数据，哈希存储还能够统一数据长度，不同长度的原始数据存储至链上后具有相同长度，但如果原始数据的长度较小，全量存储在性能与资源开销上是优于哈希存储的。

根哈希存储。根哈希存储的思路是将数据在链下以默克尔树或默克尔帕特里夏树的形式存储，然后只将默克尔树或默克尔帕特里夏树的根哈希值存储至链上，最终存储至链上的数据$\mathcal{D}_{RH}$可以表示为：

$$\mathcal{D}_{RH} = \mathrm{root}\left(\mathrm{tree}\left(\{d_i: i \in \{0,1,\dots\}\}\right)\right) \tag{3-3}$$

其中，tree表示将数据集合$\{d_i: i \in \{0,1,\dots\}\}$以默克尔树或默克尔帕特里夏树的形式存储，root代表获取上述数据结构的根哈希值，此存储方法可以将任意大小的数据集合转化为一个固定长度的哈希值，从而节约链上存储资源，但此方法也相应地有较高的计算复杂度，当实际环境中存在大量物联网设备或需要存储大量数据的情况下，根哈希存储是最佳的选择。

其中，链上存储即指全量存储，链上存证是指哈希存储或根哈希存储，因为对于哈希存储与根哈希存储，真实数据仍然存储至链下，存储至链上的哈希值为真实数据提供数据完整性证明。经测试，在以太坊中存储或修改一条 32 字节哈希值约需要消耗 47928 gas，要远小于全量存储。

### 3.1.3 存储时间

将数据存储至链上的实质是矿工将待存储数据打包至一个新区块中，所以在区块链中存储一条数据所需的时间取决于区块链中诞生一个新区块所需的时间$t_B$，另外，由于挖矿需要交易的验证结果（即状态树、交易树以及收据树的根哈希值）作为输入，所以矿工在开始挖矿前必须完成交易的验证。综上，$t_B$由交易验证时间$t_V$与挖矿时间$t_M$两部分组成，可以表示为：

$$t_B = t_V + t_M \tag{3-4}$$

其中，$t_V$取决于待验证交易的数量以及每个交易的验证复杂度，$t_M$取决于区块链网络的挖矿难度$H_d$，挖矿函数的数学定义如下[45]：

$$n \leq \frac{2^{256}}{H_d} \wedge m = H_m, (m,n) = \mathrm{PoW}(H_\pi, H_n, \boldsymbol{d}) \tag{3-5}$$

其中，PoW函数代表以太坊的工作量证明算法 Ethash[48]，PoW可以视为一个伪随机数生成器，输入为$H_\pi$、$H_n$与$\boldsymbol{d}$，在一次挖矿的过程中，$H_\pi$与$\boldsymbol{d}$为定值，矿工需要找到一个合适的$H_n$，使得输出的结果$m$与$n$满足公式（3-5），容易看出，挖矿难度$H_d$越大，则$2^{256}/H_d$越小，$n$的可选范围越小，挖矿越难，从而诞生一个新区块所需的挖矿时间$t_M$越大，矿工执行一次PoW函数能够得到一个新区块的概率$P$可以表示为：

$$P = \frac{2^{256}/H_d}{2^{256}} = \frac{1}{H_d} \tag{3-6}$$

如果矿工执行一次PoW函数所需要的平均时间为$t_{\mathrm{PoW}}$，则从统计学的角度上看，$t_M$可以表示为：

$$t_M(t_{\mathrm{PoW}}, H_d) \approx \frac{t_{\mathrm{PoW}}}{P} = H_d \cdot t_{\mathrm{PoW}} \tag{3-7}$$

可以看出，$t_M$与$t_{\mathrm{PoW}}$成正比，比例系数为$H_d$，而$t_{\mathrm{PoW}}$取决于矿机的算力，所以最终的结论是$t_M$由挖矿难度$H_d$与矿机算力两个因素共同决定。

## 3.2 基于智能合约的物联网访问控制模型设计（IoT Access Control Model Design Based on Smart Contract）

物联网是一个复杂并且庞大的异构网络，网络中的设备种类、通信协议以及网络拓扑等可能皆不相同，所以物联网领域需要一种更加强大的访问控制模型。依据小节（2-2），相较于 DAC 模型、MAC 模型、RBAC 模型以及基于 ACL 的访问控制模型，ABAC 模型在动态环境中，具有较好的灵活性，安全性以及可扩展性并且可以实现更细粒度的控制[18]，所以在 IoTAC 中使用 ABAC 模型。另外，相较于比特币等公链系统，以太坊的智能合约技术可以在保证安全性与可靠性的前提下能够实现更为复杂的访问控制逻辑，所以在 IoTAC 中使用以太坊作为底层区块链平台。

综上所述，本节提出了一个基于链上合约与 ABAC 的物联网访问控制模型，并将其命名为 IoTAC。在 IoTAC 中，所有数据存储至链上合约中，主要包括访问者实体与资源实体（即物联网设备）的属性信息以及每种操作对应的策略信息，并且访问控制逻辑的计算也由链上合约来完成。

本节将 IoTAC 的实现范式定义为如下形式：

$$R = \left(id_s, id_o, id_a, (pk_s, sig_s)\right), \quad s \in \mathcal{S} \wedge o \in \mathcal{O} \wedge a \in \mathcal{A} \tag{3-8}$$

$$f_{\boldsymbol{D}}: \left(\vec{k}, a, \vec{\sigma}\right) \longrightarrow r, \quad a \in \mathcal{A} \tag{3-9}$$

$$\boldsymbol{D} = \left((\mathcal{M}_S, \mathcal{M}_O, \mathcal{M}_A), f_{\boldsymbol{D}}\right) \tag{3-10}$$

$$\mathcal{L}: (R, \boldsymbol{D}) \longrightarrow \{0,1\} \tag{3-11}$$

$$\Upsilon: (\mathcal{L}, \boldsymbol{D}, R) \longrightarrow \{0,1\} \tag{3-12}$$

其中，$\Upsilon$为 IoTAC 的计算函数，负责根据访问者实体的请求$R$与已存储至链上的相关数据集合$\boldsymbol{D}$完成访问控制逻辑$\mathcal{L}$的计算。访问控制模型 IoTAC 是一个计算引擎，能够根据访问者实体的请求演算出此访问者实体是否符合对应的访问控制策略，如图 3-3 所示，下文将对 IoTAC 的实现范式、组成结构以及工作原理进行介绍。


图 3-3 IoTAC 示意图

Figure 3-3 IoTAC schematic

IoTAC 的系统架构如图 3-4 所示，由物联网网关作为 ABAC 范式中的策略执行点（PEP），访问者实体的请求$R$由物联网网关负责接收，物联网网关将$R$中信息传递至链上合约，最后执行由链上合约返回的访问控制结果。链上合约部分由 5 个智能合约构成，分别为 Subject 合约、Object 合约、Policy 合约、签名验证（ECDSA）合约以及 Verify 合约，其中，Subject 合约负责存储每个访问者实体的属性信息；Object 合约负责存储每个资源实体的属性信息；Policy 合约负责

存储每种操作对应的访问控制策略；Verify 合约负责完成访问控制逻辑的计算，Subject 合约与 Object 合约共同作为 ABAC 范式中的策略信息点（PIP），Policy 合约作为 ABAC 范式中的策略获取点（PRP），Verify 合约与 ECDSA 合约作为 ABAC 范式中的策略决策点（PDP）。

另外，物联网网关是访问者实体与物联网设备之间的桥梁，负责完成两者之间的协议转换，访问者实体与物联网网关之间通常使用 HTTP 协议进行通信，但物联网网关与物联网设备之间的协议较为复杂，可能涉及不同的有线或无线通信技术[28]。

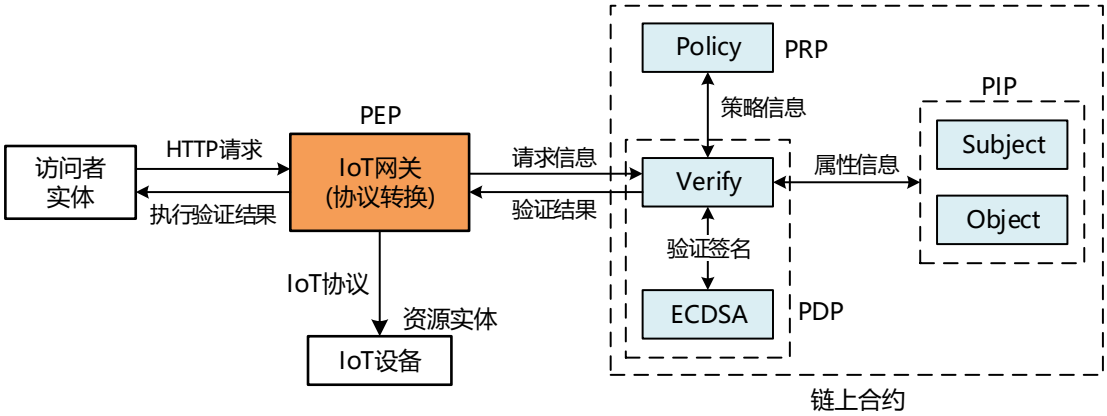



图 3-4 IoTAC 系统架构图

Figure 3-4 IoTAC system architecture diagram

为方便下文表述，本文作如下定义：

1) $\mathcal{M}: k \longrightarrow v$表示一个映射表，可以使用键$k$以时间复杂度$O(1)$从$\mathcal{M}$中得到值$v$；

2) 符号$\mathbb{C}$表示智能合约，其中，$\mathbb{C}_S$、$\mathbb{C}_O$、$\mathbb{C}_P$、$\mathbb{C}_{ecdsa}$及$\mathbb{C}_V$分别表示 Subject 合约、Object 合约、Policy 合约及 Verify 合约；

3) 符号$id_x$表示实体$x$的标识。

在本方案中，访问者实体的请求$R$的定义如公式（3-8）所示，其中，$id_s$、$id_o$与$id_a$分别表示访问者实体$s$、资源实体$o$与操作$a$的标识，$pk_s$与$sig_s$分别表示实体$s$的签名公钥与签名，当$\mathbb{C}_V$收到来自物联网网关的调用请求后，会首先验证$sig_s$是否正确，然后根据$id_s$、$id_o$与$id_a$从$\mathbb{C}_S$、$\mathbb{C}_O$与$\mathbb{C}_P$中获取对应的访问者实体属性信息、资源实体属性信息与访问控制策略，验证请求是否满足公式（2-5），最终返回验证结果至物联网网关。

为节约链上存储资源与统一数据格式，IoTAC 使用哈希存储的方式存储相关数据，哈希函数为 Keccak-256。本文使用符号$\boldsymbol{D}$表示由所有链上存储数据构成的数据集合，主要包括访问者实体属性信息、资源实体属性信息以及访问控制策略，

如公式（3-10）所示，$\boldsymbol{D}$由$\mathbb{C}_{\mathcal{S}}$、$\mathbb{C}_{\mathcal{O}}$和$\mathbb{C}_{\mathcal{P}}$存储和维护，对于$\mathbb{C}_{\mathcal{S}}$、$\mathbb{C}_{\mathcal{O}}$和$\mathbb{C}_{\mathcal{P}}$的详细介绍如下：

$\mathbb{C}_{\mathcal{S}}$将所有访问者实体的属性信息以映射表$\mathcal{M}_{\mathcal{S}}$的形式存储，$\mathcal{M}_{\mathcal{S}}$可以表示为：

$$\mathcal{M}_{\mathcal{S}}: id_s \longrightarrow \mathcal{M}_A^s, \qquad s \in \mathcal{S} \tag{3-13}$$

$$\mathcal{M}_s: A_0 \longrightarrow A_1, \qquad s \in \mathcal{S} \wedge A \in \mathbb{A}_s \tag{3-14}$$

其中，$\mathcal{M}_s$表示由访问者实体$s$的属性键值对集合$\mathbb{A}_s$构成的属性映射表，$\mathcal{M}_s$存储了访问者实体$s$的所有属性信息，$\mathcal{M}_{\mathcal{S}}$表示$id_s$与$\mathcal{M}_s$之间的映射表，$\mathcal{M}_{\mathcal{S}}$存储了每个访问者实体的所有属性信息。

$\mathbb{C}_{\mathcal{O}}$与$\mathbb{C}_{\mathcal{S}}$的数据存储结构相似，$\mathbb{C}_{\mathcal{O}}$将所有资源实体的属性信息以映射表$\mathcal{M}_{\mathcal{O}}$的形式存储，$\mathcal{M}_{\mathcal{O}}$可以表示为：

$$\mathcal{M}_{\mathcal{O}}: id_o \longrightarrow \mathcal{M}_A^o, \qquad o \in \mathcal{O} \tag{3-15}$$

$$\mathcal{M}_o: A_0 \longrightarrow A_1, \qquad o \in \mathcal{O} \wedge A \in \mathbb{A}_o \tag{3-16}$$

其中，$\mathcal{M}_o$表示由资源实体$o$的属性键值对集合$\mathbb{A}_o$构成的属性映射表，$\mathcal{M}_o$存储了资源实体$o$的所有属性信息，$\mathcal{M}_{\mathcal{O}}$表示$id_o$与$\mathcal{M}_o$之间的映射表，$\mathcal{M}_{\mathcal{O}}$存储了每个资源实体的所有属性信息。

$\mathbb{C}_{\mathcal{P}}$将每种操作对应的访问控制策略信息以映射表$\mathcal{M}_{\mathcal{A}}$的形式存储，$\mathcal{M}_{\mathcal{A}}$可以表示为：

$$\mathcal{M}_{\mathcal{A}}: id_a \longrightarrow \mathcal{P}_a, \qquad a \in \mathcal{A} \tag{3-17}$$

$$\mathcal{P}_a = \left(\left\{\mathcal{M}_i^{\mathcal{S}}\right\}_{i\in[m]}, \left\{\mathcal{M}_i^{\mathcal{O}}\right\}_{i\in[n]}\right), a \in \mathcal{A} \tag{3-18}$$

其中，$\mathcal{P}_a$为操作$a$对应的访问控制策略，由访问者实体属性映射表集合$\{\mathcal{M}_i^{\mathcal{S}}: i \in [m]\}$与资源实体属性映射表$\{\mathcal{M}_i^{\mathcal{O}}: i \in [n]\}$两部分构成，$\mathcal{M}_{\mathcal{A}}$表示$id_a$与$\mathcal{P}_a$之间的映射表，$\mathcal{M}_{\mathcal{A}}$存储了所有访问控制策略。

链上合约由资源拥有者负责创建与维护，链上合约需要对外提供相应的调用功能$f_{\boldsymbol{D}}$，以使资源拥有者可以对链上数据进行维护（包括增加、删除以及修改等），并且其他访问者实体可以对链上数据的正确性进行检查，调用功能$f_{\boldsymbol{D}}$的定义如公式（3-9）所示，其中，$k$表示$\boldsymbol{D}$的键字段，$a$表示操作名称，$\sigma$表示调用$f_{\boldsymbol{D}}$时的附加或可选参数（例如待添加或待修改的数据等）。在 IoTAC 中，链上数据只有资源拥有者能够修改，即链上数据的获取权限是公开的，任何访问者实体都可以获取，虽然链上数据是公开的，但是链上数据采用哈希存储，这保证了匿名性。另外，当$f_{\boldsymbol{D}}$被调用时，EVM 能够获取调用者实体的账户地址信息完成权限校验。

## 3.3 访问控制算法设计（Access Control Algorithm Design）

本节将 IoTAC 的访问控制逻辑$\mathcal{L}$定义为公式（3-11），访问控制逻辑$\mathcal{L}$由合约$\mathbb{C}_{\mathcal{V}}$负责完成计算，访问控制逻辑$\mathcal{L}$的执行需要将来自访问者实体的请求$R$作为

系统的外部输入，数据集合$\boldsymbol{D}$作为系统的内部输入，算法 3-1 展示了$\mathcal{L}$在$\mathbb{C}_{\mathcal{V}}$中的具体计算过程。

算法 3-1 访问控制逻辑$\mathcal{L}$在合约$\mathbb{C}_{\mathcal{V}}$中的执行过程

输入：请求$R = (id_s, id_o, id_a, (pk_s, sig_s))$;

输出：权限验证结果$r \in \{0,1\}$，其中，如果$r = 1$表示访问控制验证通过；反之，表示不通过;

1. // 验证请求者提供的签名公钥$pk_s$是否正确
2. **if** $\mathcal{H}(pk_s)! = f_C((\mathcal{S}, id_s, "pk"), "search", ())$
3. **return** 0;
4. // 验证请求者提供的签名$sig_s$是否正确
5. **if** $\mathcal{V}_{sig}(pk_s, sig_s)! = 1$
6. **return** 0;
7. // 获取操作$a$对应的访问控制策略
8. $\left(\{\mathcal{M}_i^{\mathcal{S}}\}_{i\in[m]}, \{\mathcal{M}_i^{\mathcal{O}}\}_{i\in[n]}\right) \leftarrow f_{\boldsymbol{D}}((\mathcal{P}, id_a), "search", ())$;
9. // 验证$s$与$o$是否满足访问控制策略
10. **if** $\mathcal{V}_{\mathcal{M}}\left(\{\mathcal{M}_i^{\mathcal{S}}\}_{i\in[m]}, (\mathcal{S}, id_s)\right) \wedge \mathcal{V}_{\mathcal{M}}\left(\{\mathcal{M}_i^{\mathcal{O}}\}_{i\in[n]}, (\mathcal{O}, id_o)\right)! = 1$
11. **return** 0;
12. **return** 1; // 通过验证

在算法 3-1 中，$\mathcal{V}_{\mathcal{M}}: (\{\mathcal{M}_i : i \in [m]\}, id_x) \longrightarrow \{0,1\}$表示验证访问者实体或资源实体$x$是否满足策略$\mathcal{M}_i$，算法 3-2 展示了$\mathcal{V}_{\mathcal{M}}$的具体算法流程，为方便下文表述，本文使用函数$\mathcal{K}(\mathcal{M})$表示获取由映射表$\mathcal{M}$的所有键字段构成的集合。

算法 3-2 属性映射表集合验证算法$\mathcal{V}_{\mathcal{M}}$

输入：属性映射表集合$\{\mathcal{M}_i\}_{i\in[m]}$，访问者实体或资源实体的标识$id_x$;

输出：验证结果$flag \in \{0,1\}$，其中，如果$flag = 1$表示验证通过，反之，表示不通过;

1. $flag \leftarrow 0$; // 将验证结果$flag$初始化为 0
2. // 遍历属性映射表集合$\{\mathcal{M}_i\}_{i\in[m]}$
3. **foreach** $i$ **in** $[m]$
4. $K \leftarrow \mathcal{K}(\mathcal{M}_i)$; // 获取映射表$\mathcal{M}_i$的所有键字段
5. $flag' \leftarrow 1$; // 用于记录中间验证结果
6. // 验证$x$是否满足策略$\mathcal{M}_i$
7. **foreach** $k$ **in** $K$
8. // $x$不满足策略$\mathcal{M}_i$
9. **if** $\mathcal{M}_i(k)! = f_{\boldsymbol{D}}((id_x, k), "search", ())$
10. $flag' \leftarrow 0$;
11. **break**;
12. // $x$满足策略$\mathcal{M}_i$
13. **if** $flag' = 1$
14. $flag \leftarrow flag'$;

```
15.            break;
16.  // 返回最终结果
17.  return flag;
```

## 3.4 实验评估及安全分析（Experimental Evaluation and Safety Analysis）

小节（2.3.4）介绍 EVM 的运行并不需要智能合约的具体执行时间，所以 EVM 对于智能合约的执行没有基于时间的量度，所以智能合约的执行时间是无法精确测量的，本文得到的智能合约执行时间$t_{\mathbb{C}}$可以表示为：

$$t_{\mathbb{C}} = \widetilde{t_{\mathbb{C}}} + (t_1 + 2 \cdot t_2) \tag{3-19}$$

其中，$\widetilde{t_{\mathbb{C}}}$为合约的实际执行时间，$t_1$为执行环境的上下文时延，主要包括初始化合约调用上下文，向区块链网络节点发出请求以及处理区块链网络节点返回的结果所消耗的时间，$t_2$为物联网网关与区块链网络节点之间的网络时延。测量时间$t_{\mathbb{C}}$是智能合约实际执行时间$\widetilde{t_{\mathbb{C}}}$的上限。

### 3.4.1 实验环境

本节实验所涉及的相关服务节点的硬件参数及其对应的操作系统信息如下所示：

表 3-1 IoTAC 硬件参数及操作系统信息

Table 3-1 Hardware parameters and operating system information of IoTAC

| 相关节点 | 硬件参数 | 操作系统 |
|---|---|---|
| 笔记本电脑（本机，徐州） | CPU：i5-1135G7（4 核）<br>内存：16GB | WSL2 Ubuntu 20.04 LTS 64 位 |
| 云节点（北京） | CPU：1 核<br>内存：2GB | Ubuntu Server 20.04 LTS 64 位 |
| 树莓派 4B（徐州） | CPU：Broadcom BCM2711（4 核）<br>内存：4GB | Ubuntu Server 20.04 LTS 64 位 |

实验场景如图 3-5 所示，本实验使用树莓派 4B（Raspberry Pi 4 Model B）[①]作为物联网网关，考虑真实场景的网络延迟，本实验使用一个所处位置与本机位置不同的云节点，将该云节点作为区块链网络节点。使用本机向树莓派节点发送 HTTP 请求，树莓派节点对请求进行解析，将请求中的相关信息发送至云节点（北京），云节点（北京）负责完成访问控制逻辑的计算并将访问控制验证结果返回给树莓派节点，最终由树莓派节点执行访问控制结果。

① 树莓派 4B（Raspberry Pi 4 Model B），详见：https://www.raspberrypi.com/products/raspberry-pi-4-model-b/。

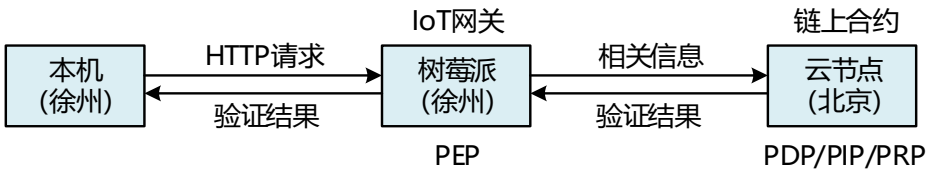



图 3-5 IoTAC 实验场景架构图

Figure 3-5 Experimental scenario architecture diagram of IoTAC

表 3-2 展示了本实验所涉及的相关编译器与开源库的版本信息，IoTAC 的软件部分由智能合约与网络服务两部分构成，智能合约部分使用 Solidity 语言编写，使用开源工具库 Truffle 与 Web3.js 对合约进行部署及交互，网络服务运行在树莓派节点中，负责监听来自访问者实体的请求并对链上合约发起调用，并执行由链上合约返回的访问控制验证结果，网络服务使用 JavaScript 语言编写，运行在 Node.js 环境中。

表 3-2 IoTAC 编译器与开源库的版本信息

Table 3-2 Compiler and open source library version information of IoTAC

| 相关编译器与开源库 | 版本 |
|---|---|
| Golang | go1.17.6 linux/amd64 |
| Solidity | v0.8.11 |
| Node | v16.13.2 |
| Geth | v1.9.25 |
| Truffle | v5.4.32 |
| Web3.js | v1.5.3 |

### 3.4.2 权限维护相关实验

本小节对 IoTAC 中访问者实体的权限维护进行了相关实验，主要涉及与权限维护相关的合约的部署、与权限维护相关的数据（包括属性及策略信息）的存储、修改以及删除。为便于实验，本实验将访问控制策略中访问者实体部分的属性映射表集合的大小设置为 1，即访问者实体部分的属性映射表集合只含有一个属性映射表，即$\mathcal{P}_S = \{\mathbb{A}_i^S : i \in [1]\}$。为方便后续表述，本文做如下定义：

1) 符号$N_S$表示访问者实体的属性数量；
2) 符号$N_O$表示资源实体的属性数量；
3) 符号$N_P^S$表示访问控制策略中访问者实体部分的属性数量；
4) 符号$N_P^O$表示访问控制策略中资源实体部分的属性数量。

对于权限维护相关数据的存储、修改及删除，由于访问者实体与资源实体的数据存储结构相同，所以本实验只对访问者实体部分进行测试分析，实验结论同样适用于资源实体。

（1）合约部署 gas 消耗

将合约$\mathbb{C}_{\mathcal{S}}$、$\mathbb{C}_{\mathcal{O}}$、$\mathbb{C}_{\mathcal{P}}$、$\mathbb{C}_{\mathcal{V}}$以及$\mathbb{C}_{ecdsa}$部署至链上的 gas 消耗如表 3-3 所示。小节（3.2）已介绍，$\mathbb{C}_{\mathcal{S}}$与$\mathbb{C}_{\mathcal{O}}$的数据存储结构是相似的，所以部署$\mathbb{C}_{\mathcal{S}}$与部署$\mathbb{C}_{\mathcal{O}}$所消耗的 gas 基本相同，由于$\mathbb{C}_{\mathcal{P}}$的数据存储结构相较于$\mathbb{C}_{\mathcal{S}}$与$\mathbb{C}_{\mathcal{O}}$更为复杂，$\mathbb{C}_{\mathcal{P}}$的数据存储结构可以视为由$\mathbb{C}_{\mathcal{S}}$或$\mathbb{C}_{\mathcal{O}}$的数据存储结构构成的数组，所以部署$\mathbb{C}_{\mathcal{P}}$的 gas 消耗要高于部署$\mathbb{C}_{\mathcal{S}}$或$\mathbb{C}_{\mathcal{O}}$，另外，访问控制逻辑$\mathcal{L}$在$\mathbb{C}_{\mathcal{V}}$中实现，$\mathbb{C}_{\mathcal{V}}$还需要实现对$\mathbb{C}_{\mathcal{S}}$、$\mathbb{C}_{\mathcal{O}}$以及$\mathbb{C}_{\mathcal{P}}$的调用，所以相较于$\mathbb{C}_{\mathcal{S}}$与$\mathbb{C}_{\mathcal{O}}$，$\mathbb{C}_{\mathcal{V}}$的执行逻辑更为复杂，部署合约$\mathbb{C}_{\mathcal{V}}$也需要消耗更多的 gas，而$\mathbb{C}_{ecdsa}$需要实现更为复杂的椭圆曲线计算过程，所以消耗 gas 最多。

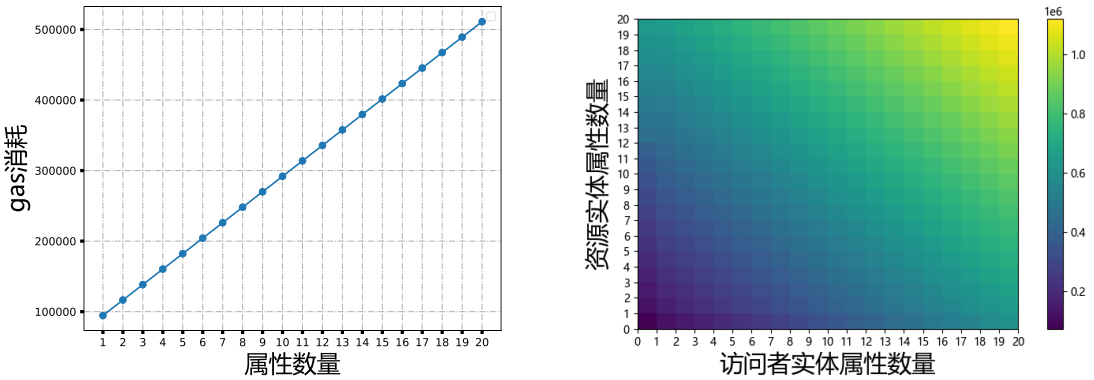
表 3-3 合约部署的 gas 消耗

Table 3-3 Contracted deployment gas consumption

| 合约名称 | 合约部署 gas 消耗 |
|---|---|
| $\mathbb{C}_{\mathcal{V}}$ | 996116 |
| $\mathbb{C}_{\mathcal{S}}$ | 534731 |
| $\mathbb{C}_{\mathcal{O}}$ | 534719 |
| $\mathbb{C}_{\mathcal{P}}$ | 1057110 |
| $\mathbb{C}_{ecdsa}$ | 2003055 |

（2）数据存储 gas 消耗

本小节对将访问者实体存储至$\mathbb{C}_{\mathcal{S}}$的 gas 消耗$g_{\text{add}}^{\mathcal{S}}$与访问者实体的属性数量$N_{\mathcal{S}}$之间的关系进行了实验，图 3-6（a）展示了实验结果，从图中可以看出，$g_{\text{add}}^{\mathcal{S}}$与$N_{\mathcal{S}}$呈线性关系，这是由于 IoTAC 使用哈希方式存储属性信息，即将属性键值对$A=(A_0,A_1)$以$\big(\mathcal{H}(A_0),\mathcal{H}(A_1)\big)$的形式存储至$\mathbb{C}_{\mathcal{S}}$，$\big(\mathcal{H}(A_0),\mathcal{H}(A_1)\big)$的长度为定值，所以为线性关系，使用最小二乘法计算对$g_{\text{add}}^{\mathcal{S}}$与$N_{\mathcal{S}}$的关系进行估计，得到公式（3-20）。




(a) 访问者实体信息存储　　(b) 访问控制策略存储

图 3-6 链上存储实验结果

Figure 3-6 On-chain storage experimental results

本小节对将访问控制策略存储至$\mathbb{C}_{\mathcal{P}}$的 gas 消耗$g_{add}^{\mathcal{P}}$与访问者实体部分的属性数量$N_{\mathcal{P}}^{\mathcal{S}}$以及资源实体部分的属性数量$N_{\mathcal{P}}^{\mathcal{O}}$之间的关系进行了实验，图 3-6（b）展示了实验结果，依据在上一实验中得到的线性关系，同样使用最小二乘法对$g_{add}^{\mathcal{P}}$与$N_{\mathcal{P}}^{\mathcal{S}}$及$N_{\mathcal{P}}^{\mathcal{O}}$的关系进行估计，得到公式（3-21）。

另外，对于能够存储至链上合约中的访问者实体、资源实体与访问控制策略的属性数量最大值取决于区块链网络中 gasLimit 参数的具体大小。

（3）数据修改 gas 消耗

对于修改已存储至链上合约中的信息，本小节对修改$\mathbb{C}_{\mathcal{S}}$中指定访问者实体的属性信息的 gas 消耗与被修改的属性值的数量$n$之间的关系进行了两次实验，在第一次实验中，每次调用$\mathbb{C}_{\mathcal{S}}$只修改一条属性信息；在第二次实验中，以批量的方式修改$\mathbb{C}_{\mathcal{S}}$中的属性信息，两者在调用$\mathbb{C}_{\mathcal{S}}$时的参数传递方式是不同的，前者传递的参数为一个键值对$A$，后者传递的参数为两个具有动态长度的数组，分别存储键字段与值字段，可以表示为$\{A_0^i: i \in \{0,1,\dots\}\}$与$\{A_1^i: i \in \{0,1,\dots\}\}$。

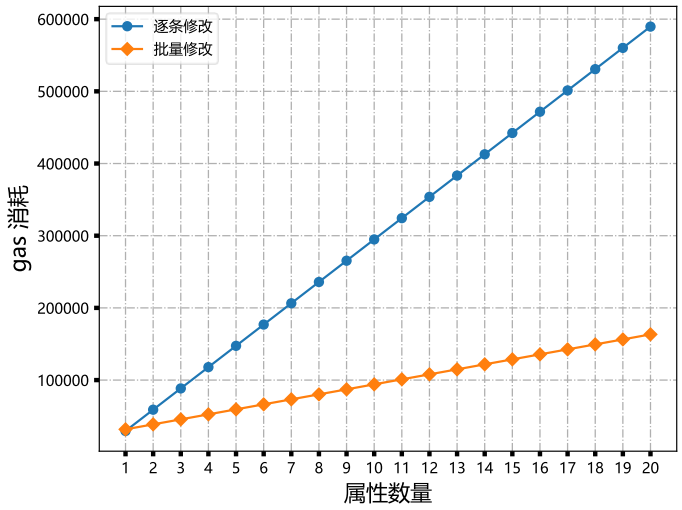



图 3-7 链上合约信息修改实验结果

Figure 3-7 Experimental results of on-chain contract information modification

图 3-7 展示了实验结果，实验结果得出，第一种方式修改一条属性值需要消耗 29484 gas，第二种方式修改一条属性值需要消耗 31735 gas，但如果修改多条属性值，第二种方式要明显优于第一种方式，这是因为对于修改多条属性值，逐条修改方式需要多次初始化合约执行上下文，而批量修改方式只需一次，但如果只修改一条属性值，批量修改方式较逐条修改方式则有更高的执行复杂度。这两种方式的最小二乘估计结果如公式（3-22）与公式（3-23）所示。

$$g_{add}^{\mathcal{S}}(N_{\mathcal{S}}) = 21924.13 \cdot N_{\mathcal{S}} + 72625.3 \tag{3-20}$$

$$g_{add}^{\mathcal{P}}\left(N_{\mathcal{P}}^{\mathcal{S}}, N_{\mathcal{P}}^{\mathcal{O}}\right) = 27578.31 \cdot N_{\mathcal{P}}^{\mathcal{S}} + 26668.99 \cdot N_{\mathcal{P}}^{\mathcal{O}} + 74493 \tag{3-21}$$

$$g_{setInOne}^{\mathcal{S}}(n) = 29484 \cdot n \tag{3-22}$$

$$g_{setInBatch}^{\mathcal{S}}(n) = 6923.9 \cdot n + 24824.95 \tag{3-23}$$

（4）数据删除 gas 消耗

对于删除已存储至链上合约中的信息，在 Solidity 语言中，删除某一个映射表需要提供该映射表所有的键，并将该键对应的值置为 0，但映射表不会维护所有的键，所以需要另外维护一个具有动态长度的数组，用以保存映射表的键字段，但这种设计会增加链上存储资源的占用，所以在 IoTAC 中，为节约链上存储资源，链上合约不会存储访问者实体与资源实体的属性名称，因此在 IoTAC 中无法真正删除一个实体。在 IoTAC 的底层设计中，对于一个实体是否存在是通过一个标志位映射表$\mathcal{M}_{id}: id \longrightarrow \{0,1\}$来判断的，值字段为 1 表示标识$id$对应的实体存在，反之，不存在，所以在 IoTAC 中删除一个实体，并不是真正将其删除，而是将其对应的标志位置为 0，如图 3-8 所示，经测试，删除实体需要消耗 14850 gas。

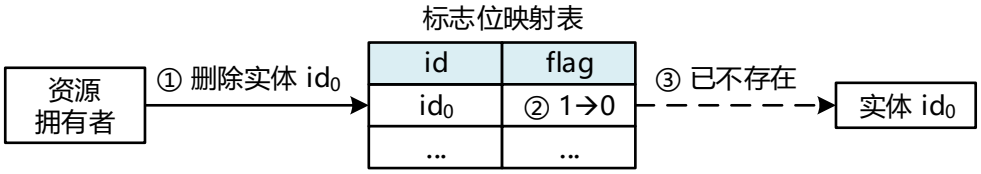



图 3-8 实体信息删除示意图

Figure 3-8 Entity information deletion schematic

（5）总结

从上述实验结果可以看出，IoTAC 对于链上存储空间的占用与实际数据$D$的大小相关，在 IoTAC 中，对于数据存储、修改及删除的 gas 消耗可以表示为$O(|D|)$，如果实际数据过大，所消耗的 gas 可能超出以太坊网络的 gasLimit，此时操作将会无法执行。综上，IoTAC 的运行严重依赖于链上资源，当实际数据的复杂度过高时，IoTAC 将无法运行，即 IoTAC 的可扩展性受限。

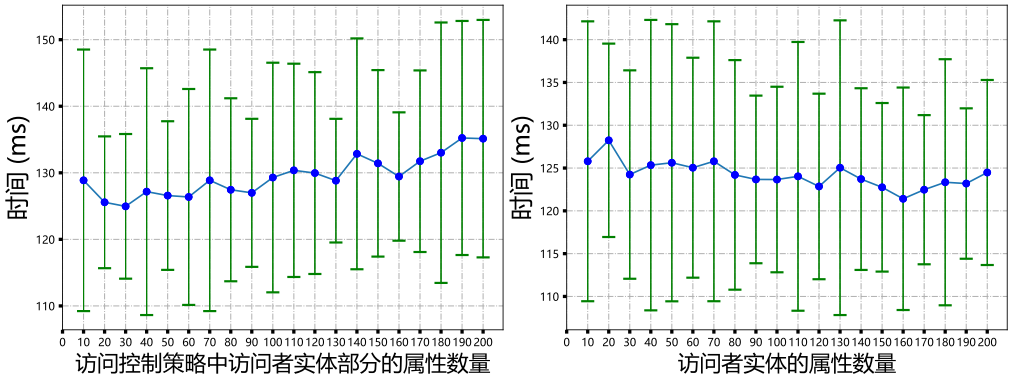
### 3.4.3 访问控制验证相关实验

（1）属性数量对访问控制验证时间的影响

对于访问控制验证，从算法 3-1 中可以看出，合约$\mathbb{C}_{\mathcal{V}}$中的访问控制验证过程可以划分为验证签名、获取信息以及验证访问者实体与资源实体是否满足访问控制策略三部分，本小节对第三部分进行实验，由于验证访问者实体是否满足访问控制策略与验证资源实体是否满足访问控制策略是等价的，所以本实验只对访问者实体验证进行实验，本实验分为以下两部分：

1) 将$N_S$固定为 200，$N_P^S$从 10 逐渐递增至 200，每次增加 10；
2) 将$N_P^S$固定为 10，$N_S$从 10 逐渐递增至 200，每次增加 10。

另外，考虑网络延时与服务节点的实行运行状况，每次实验重复 100 次，计算每次实验的均值与方差，实验结果如图 3-9 所示[①]，从图 3-9（a）中可以看出，访问控制验证时间会随着访问控制策略中访问者实体的属性数量的增加而缓慢增加，但所用时间在 100ms 至 160ms 之间，能够满足实际使用需求，其实，$N_{\mathcal{P}}^{\mathcal{S}}$ 等于算法 3-2 中第 7 行至第 11 行的循环次数，$N_{\mathcal{P}}^{\mathcal{S}}$越大，遍历次数越多，故需花费更多的时间，另外，$\mathbb{C}_{\mathcal{V}}$调用$\mathbb{C}_{\mathcal{S}}$以获取访问者实体的属性信息，这个过程也需要花费一定的时间。从图 3-9（b）中可以看出，访问控制验证时间与$N_{\mathcal{S}}$是没有关系的，所用时间在 100ms 至 150ms 之间，这是因为属性信息是以映射表的方式存储至链上的，检索的时间复杂度与映射表大小无关，始终为$O(1)$，实验结果也同时验证了 Solidity 语言对于映射表的底层实现是合理的。




(a) (b)

图 3-9 IoTAC 访问控制验证时间与属性数量的关系

Figure 3-9 Access control verification time versus number of attributes in IoTAC

（2）IoTAC 整体时间性能测试

本小节对于树莓派节点与云节点（北京）之间的网络延时、一次完整的访问控制验证的所用时间（其中，$N_{\mathcal{S}}=20$，$N_{\mathcal{O}}=20$，$N_{\mathcal{P}}^{\mathcal{O}}=20$，$N_{\mathcal{P}}^{\mathcal{S}}=20$）以及只完成访问者实体部分验证的所用时间（其中，$N_{\mathcal{S}}=20$，$N_{\mathcal{O}}=0$，$N_{\mathcal{P}}^{\mathcal{O}}=0$，$N_{\mathcal{P}}^{\mathcal{S}}=20$）进行了实验，每次实验重复 100 次，计算每次实验的均值与方差，实验结果如图 3-10 所示，实验结果的具体数据如表 3-4 所示。

表 3-4 IoTAC 时间性能测试结果

Table 3-4 IoTAC Time Performance Test Results

| 编号 | 测试项 | 均值 | 方差 |
|---|---|---|---|
| 1 | 树莓派节点至云节点（北京）的网络延时 | 33.3ms | 12.1ms |
| 2 | 验证时间（$N_{\mathcal{S}}=20,N_{\mathcal{O}}=20,N_{\mathcal{P}}^{\mathcal{O}}=20,N_{\mathcal{P}}^{\mathcal{S}}=20$） | 145.2ms | 23.2ms |
| 3 | 验证时间（$N_{\mathcal{S}}=20,N_{\mathcal{O}}=0,N_{\mathcal{P}}^{\mathcal{O}}=0,N_{\mathcal{P}}^{\mathcal{S}}=20$） | 140.1ms | 18.9ms |

① 在图 3-9 中，蓝色实心点表示每次测量结果的均值，绿色实线为误差线，其长度等于每次测试结果的标准差的 2 倍，关于误差线（或误差棒，Error Bar）的详细解释可参阅：https://en.wikipedia.org/wiki/Error_bar。

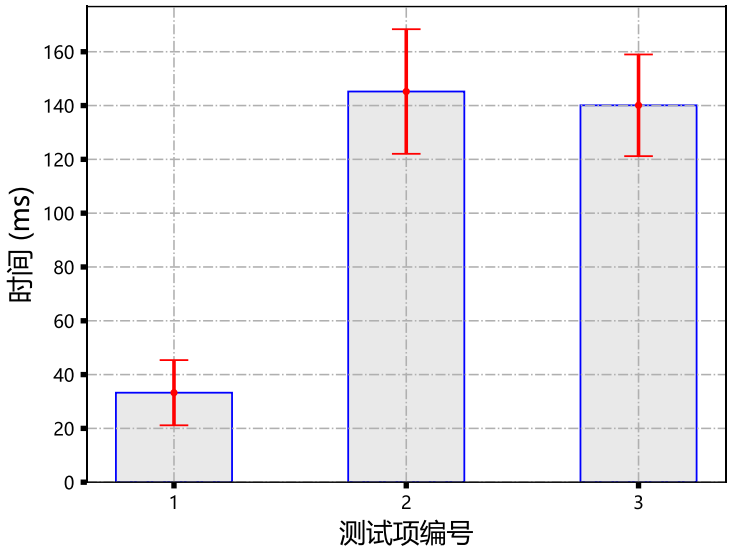



图 3-10 IoTAC 时间性能测试结果

Figure 3-10 IoTAC time performance test results

从各项实验结果可以看出，在网络延时的均值为 33.3ms、方差为 12.1ms 的情况下，IoTAC 完成一次访问控制验证所用时间的均值为 145.2ms，方差为 23.2ms。另外，测试项 2 与测试项 3 的用时相差很小，即实际的属性验证时间很短，网络延时与合约之间的相互调用占用了大部分时间。

（5）总结

从上述实验结果可以看出，对于一次完整的访问控制验证，IoTAC 约需 150ms，另外，在 IoTAC 的链上合约中，访问控制验证为常量方法，即不会改变以太坊的世界状态，所以访问控制逻辑的执行没有 gas 消耗。综上，IoTAC 的访问控制验证速率能够满足实际应用场景。

### 3.4.4 安全性分析

参照文献[71]，本小节将访问控制模型的安全风险划分为外部风险与内部风险，其中，外部风险是指外部恶意攻击者（或非授权访问者实体）能否欺骗访问控制系统通过访问控制验证；内部风险是指已授权的访问者实体能否被恶意攻击者利用，从访问系统内部向访问控制系统发起攻击。

对于外部风险，恶意攻击者能够通过访问控制验证需要满足两个要求，第一，能够将自身的属性信息存储至链上合约中；第二，添加的属性信息要符合对应的访问控制策略。IoTAC 是由资源拥有者自主管理的访问控制模型，对于链上合约的修改只能由资源拥有者来完成，外界实体无法越过资源拥有者自行修改链上合约，另外，对于恶意攻击者能否冒充资源拥有者修改链上合约，这取决于以太坊网络的安全性，而以太坊网络是一个巨大的漏洞悬赏池，其安全性广受考验，除

非资源拥有者操作不当导致账户密钥泄露，否则恶意攻击者不可能冒充资源拥有者的身份，

对于内部风险，如果恶意攻击者打算利用某一个已授权访问者实体向访问控制系统发起攻击，那么恶意攻击者必须同时拥有该已授权访问者实体的身份标识与签名私钥，此时，恶意攻击者还需要知晓该访问者实体能够访问哪些物联网设备（即设备标识），才能向物联网网关发起一次完整的访问请求，但恶意攻击者无法从链上合约中获知，因为链上合约中只存储相关数据的哈希值，所以恶意攻击者无法向物联网网关发起一次完整的访问请求。

另外，IoTAC 为加强安全性，已授权访问者的权限具有时效性，超过一定时间后，访问者需要重新申请相关权限。

# 4 基于链上计算与存储的物联网访问控制的可扩展性优化研究

# 4 Research on Scalability Optimization of IoT Access Control Based on On-chain Computing and Storage

在 IoTAC 中，访问控制逻辑的计算与相关数据的存储皆由链上合约负责完成，物联网网关只负责接收来自访问者实体的请求与执行链上合约返回的访问控制验证结果，IoTAC 的局限性在于其严重依赖于链上资源，小节（3-1）已介绍，公链网络的链上资源具有昂贵性，并且相较于链下资源，链上资源的性能是受限的，这是公链网络的不可能三角问题[75]，这导致 IoTAC 的可扩展性较差，此处的扩展性是就模型的计算能力及存储能力而言的。

针对 IoTAC 的上述局限性，本小节基于链下存储、链上证明的区块链应用设计理念[76,77]，在 IoTAC 的基础上进行拓展，构建了基于双层区块链架构的物联网访问控制模型，并将其命名为 IoTAC-L2。IoTAC-L2 使用小节（3.1.2）介绍的根哈希存储方式，将 IoTAC 中由链上合约负责存储的数据委托至链下存储服务存储，链上合约只存储链下数据的证明信息（默克尔帕特里夏树的根哈希值），这种设计方式能够在保证安全性与可靠性的前提下，极大地减少系统对链上资源的占用。

## 4.1 基于双层区块链架构的物联网访问控制模型设计（Two-layer Blockchain Architecture-based IoT Access Control Model Design）

IoTAC-L2 的实现范式与 IoTAC 相同，即公式（3-8）至公式（3-12），但两者的实现方式不同，IoTAC-L2 实现了一种双层区块链架构以完成相关计算与存储，图 4-1 展示了 IoTAC-L2 双层区块链架构的工作原理。

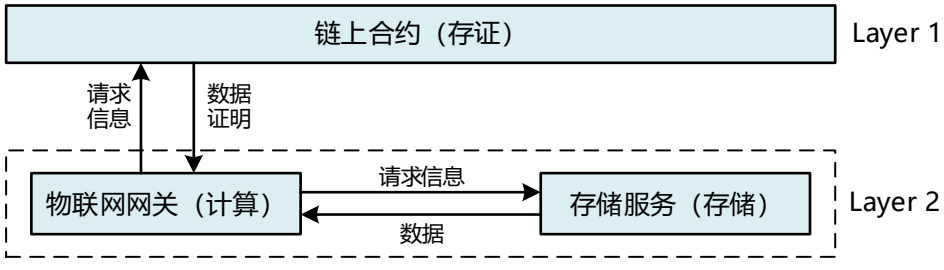



图 4-1 IoTAC-L2 双层区块链架构工作原理

Figure 4-1 IoTAC-L2 two-layer blockchain architecture working principle

在 IoTAC-L2 的双层区块链架构中，第一层（Layer 1）为链上合约，负责存储相应的证明信息，第二层（Layer 2）由物联网网关与存储服务构成，分别负责

访问控制模型的计算与存储，当物联网网关获取一条数据时，会分别向存储服务与链上合约发出请求，存储服务将相应的数据返回至物联网网关，链上合约将此数据的证明信息返回至物联网网关，物联网网关使用证明信息验证数据是否正确，这种机制能够保证存储服务无法欺骗物联网网关。

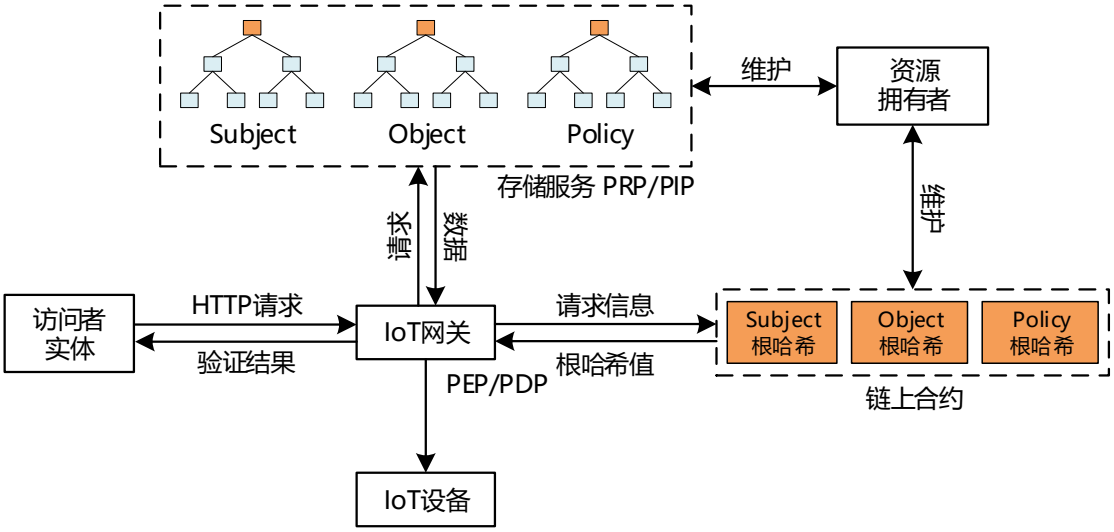



图 4-2 IoTAC-L2 系统架构图

Figure 4-2 IoTAC-L2 system architecture diagram

图 4-2 展示了 IoTAC-L2 的系统架构，在 IoTAC-L2 中，由物联网网关作为 ABAC 范式中的策略执行点（PEP）与策略决策点（PDP），物联网网关负责接收来自访问者实体的请求、访问控制逻辑的计算以及访问控制验证结果的执行；存储服务作为 ABAC 范式中的策略信息点（PIP）与策略获取点（PRP），负责存储实体属性信息与访问控制策略；链上合约负责存储链下数据的证明信息。

对于链下存储部分，由于默克尔帕特里夏树不但能够存储映射表，而且能够提供数据的默克尔证明（详见小节（2.3.2）），所以 IoTAC-L2 使用默克尔帕特里夏树存储访问者实体、资源实体以及访问控制策略。

为方便下文表述，本文作如下定义：

1) 符号$\mathbb{T}_{\mathcal{S}}$、$\mathbb{T}_{\mathcal{O}}$和$\mathbb{T}_{\mathcal{P}}$分别表示存储访问者实体、资源实体以及访问控制策略的默克尔帕特里夏树；

2) 符号$\mathbb{A}$表示键值对集合。

其中，$\mathbb{T}_{\mathcal{S}}$、$\mathbb{T}_{\mathcal{O}}$与$\mathbb{T}_{\mathcal{P}}$相当于 IoTAC 中的$\mathbb{C}_{\mathcal{S}}$、$\mathbb{C}_{\mathcal{O}}$与$\mathbb{C}_{\mathcal{P}}$，但 IoTAC-L2 不再将实体以映射表的形式存储，而是将实体以键值对集合的形式存储，为方便下文表述，本文使用符号$\mathbb{A}$表示键值对集合，综上，$\mathbb{T}_{\mathcal{S}}$、$\mathbb{T}_{\mathcal{O}}$和$\mathbb{T}_{\mathcal{P}}$可以表示为：

$$\mathbb{T}_{\mathcal{S}}: id_s \longrightarrow \mathbb{A}_s, \qquad s \in \mathcal{S} \tag{4-1}$$

$$\mathbb{T}_{\mathcal{O}}: id_o \longrightarrow \mathbb{A}_o, \qquad o \in \mathcal{O} \tag{4-2}$$

$$\mathbb{T}_{\mathcal{P}}: id_a \longrightarrow \left(\{\mathbb{A}_i^{\mathcal{S}}\}_{i\in[m]}, \{\mathbb{A}_i^{\mathcal{O}}\}_{j\in[n]}\right), \qquad a \in \mathcal{A} \tag{4-3}$$

对于链上证明部分，IoTAC-L2 通过部署一个链上合约$\mathbb{C}_{\mathcal{H}}$来维护$\mathbb{T}_{\mathcal{S}}$、$\mathbb{T}_{\mathcal{O}}$以及$\mathbb{T}_{\mathcal{P}}$的根哈希值$h_{\mathcal{S}}$、$h_{\mathcal{O}}$和$h_{\mathcal{P}}$，IoTAC-L2 将 IoTAC 中需要存储至链上合约中的所有数据转化为三个固定长度的哈希值，即无论实际数据量有多大，最终存储至链上合约中的数据的长度是固定的，空间复杂度为$O(1)$，相比于 IoTAC，这种设计能够释放链上存储空间，从而使访问控制系统具有高可扩展性。当物联网网关向存储服务请求指定数据时，存储服务将该数据及其默克尔证明一同返回至物联网网关，物联网网关获取到该数据后，将使用链上合约中该数据对应的默克尔帕特里夏树根哈希值对存储服务提供的默克尔证明进行验证，如果验证通过，则说明数据正确，反之，则说明数据错误，图 4-3 展示了数据完整性验证过程。

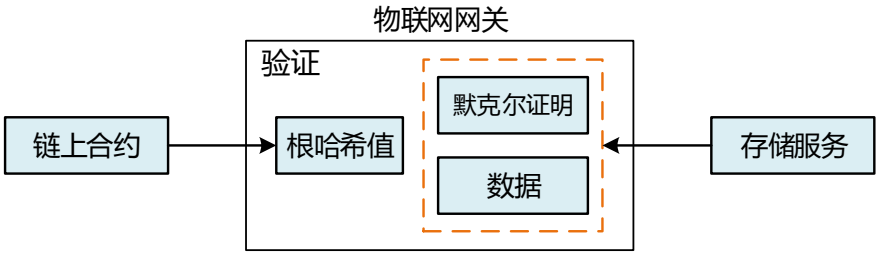



图 4-3 数据完整性验证示意图

Figure 4-3 Data Integrity Check Schematic

为方便下文表述，本文使用函数$\mathcal{V}_{\pi}:(d,\pi,h)\longrightarrow\{0,1\}$表示默克尔证明的验证函数，其中，$d$为待进行默克尔证明的数据，$\pi$为默克尔证明，$h$为根哈希值，当返回结果为1时，表示验证通过，反之，表示验证不通过。

另外，存储服务与链上合约是相互隔离的，资源拥有者实体在对存储服务中的数据进行修改后，还需修改链上合约中对应的根哈希值，但这个修改操作绝不是由存储服务来执行，而是由资源拥有者来执行，如果存储服务能够修改链上合约，那么存储服务则能够将错误数据返回至物联网网关并且通过数据验证。对于数据的权限设计，IoTAC-L2 与 IoTAC 是相同的，链上合约与存储服务中的数据由资源拥有者负责维护，任何访问者实体都可以获取已存储的数据，并对数据的正确性进行验证，但对于数据的修改只能由资源拥有者来完成，存储服务应当提供相应的安全保障以完成上述权限约束。另外，为统一数据格式与保证匿名性，存储的数据为实际数据的哈希值，主要包括实体标识、属性名称以及属性值等信息，使用哈希函数 Keccak-256。

## 4.2 访问控制算法设计（Access Control Algorithm Design）

为方便下文表述，本文使用函数$search$表示搜索键值对集合$\mathbb{A}$中键字段为$A_0'$的键值对的值字段$A_1$，$search$的数学定义如下：

$$search:(\mathbb{A},A_0')=\begin{cases}A_1 & if\ \exists A\in\mathbb{A}:A_0=A_0'\\ () & otherwise\end{cases} \tag{4-4}$$

其中，IoTAC-L2 的访问控制逻辑$\mathcal{L}$由物联网网关负责执行，算法 4-1 展示了 IoTAC-L2 访问控制逻辑$\mathcal{L}$的具体算法流程。

算法 4-1 IoTAC-L2 的访问控制逻辑$\mathcal{L}$

**输入**：请求$R = (id_s, id_o, id_a, (pk_s, sig_s))$;

**输出**：权限验证结果$r \in \{0,1\}$，其中，如果$r = 1$表示访问控制验证通过，反之，表示不通过;

// 从$\mathbb{T}_S$中获取访问者实体$s$的属性键值对集合$\mathbb{A}_s$与默克尔证明$\pi_s$

$(\mathbb{A}_s, \pi_s) \leftarrow f_{\boldsymbol{D}}((\mathbb{T}_S, id_s), "search", ())$;

// 从合约$\mathbb{C}_{\mathcal{H}}$中获取$\mathbb{T}_S$的根哈希值$h_S$

$h_S \leftarrow f_{\boldsymbol{D}}((\mathbb{C}_{\mathcal{H}}, \mathbb{T}_S), "search", ())$;

// 验证$\mathbb{A}_s$是否正确

**if** $\mathcal{V}_\pi\big((id_s, \mathbb{A}_s), \pi_s, h_S\big)! = 1$

    **return** 0;

// 验证签名$sig_s$是否正确

**if** $\Big(\mathcal{H}(pk_s) = search(\mathbb{A}_s, "pk") \wedge \mathcal{V}_{sig}(pk_s, sig_s)\Big) = 0$

    **return** 0;

// 从$\mathbb{T}_{\mathcal{P}}$中获取操作$a$对应的访问控制策略$\Big(\{\mathbb{A}_i^S\}_{i\in[m]}, \{\mathbb{A}_i^O\}_{j\in[n]}\Big)$与默克尔证明$\pi_a$

$\left(\left(\{\mathbb{A}_i^S\}_{i\in[m]}, \{\mathbb{A}_i^O\}_{j\in[n]}\right), \pi_a\right) \leftarrow f_{\boldsymbol{D}}((\mathbb{T}_{\mathcal{P}}, id_a), "search", ())$;

// 从合约$\mathbb{C}_{\mathcal{H}}$中获取$\mathbb{T}_{\mathcal{P}}$的根哈希值$h_{\mathcal{P}}$

$h_{\mathcal{P}} \leftarrow f_{\boldsymbol{D}}((\mathbb{C}_{\mathcal{H}}, \mathbb{T}_{\mathcal{P}}), "search", ())$;

// 验证访问控制策略$\Big(\{\mathbb{A}_i^S\}_{i\in[m]}, \{\mathbb{A}_i^O\}_{j\in[n]}\Big)$是否正确

**if** $\mathcal{V}_\pi\left(\left(id_a, \left(\{\mathbb{A}_i^S\}_{i\in[m]}, \{\mathbb{A}_i^O\}_{j\in[n]}\right)\right), \pi_a, h_{\mathcal{P}}\right)! = 1$

    **return** 0;

// 从$\mathbb{T}_O$中获取资源实体$o$的属性键值对集合$\mathbb{A}_o$与默克尔证明$\pi_o$

$(\mathbb{A}_o, \pi_o) \leftarrow f_{\boldsymbol{D}}((\mathbb{T}_O, id_o), "search", ())$;

// 从合约$\mathbb{C}_{\mathcal{H}}$中获取$\mathbb{T}_O$的根哈希值$h_O$

$h_O \leftarrow f_{\boldsymbol{D}}((\mathbb{C}_{\mathcal{H}}, \mathbb{T}_O), "search", ())$;

// 验证$\mathbb{A}_o$是否正确

**if** $\mathcal{V}_\pi\big((id_o, \mathbb{A}_o), \pi_o, h_O\big)! = 1$

    **return** 0;

// 验证是否满足访问控制策略

**if** $\mathcal{V}_{\mathbb{A}}\left(\{\mathbb{A}_i^S\}_{i\in[m]}, \mathbb{A}_s\right) \wedge \mathcal{V}_{\mathbb{A}}\left(\{\mathbb{A}_i^O\}_{j\in[n]}, \mathbb{A}_o\right)! = 1$

    **return** 0;

**return 1**;  // 验证通过

在算法 4-1 中，$\mathcal{V}_{\mathbb{A}}: \big(\{\mathbb{A}_i\}_{i\in[m]}, \mathbb{A}_x\big) \longrightarrow \{0,1\}$表示访问控制策略属性键值对集合验证算法，用于验证访问者实体的属性信息是否满足访问控制策略中的访问者

实体部分，或验证资源实体的属性信息是否满足访问控制策略中的资源实体部分，$\mathcal{V}_{\mathbb{A}}$的数学定义如下：

$$\mathcal{V}_{\mathbb{A}}\left(\{\mathbb{A}_i\}_{i\in[m]},\mathbb{A}_x\right)=\begin{cases}1 & if\ \exists\mathbb{A}_k\in\{\mathbb{A}_i\}_{i\in[m]}:\mathbb{A}_k\subseteq\mathbb{A}_x\\ 0 & otherwise\end{cases} \tag{4-5}$$

其中，算法 4-2 展示了$\mathcal{V}_{\mathbb{A}}$的具体算法流程，该流程与算法 3-2 类似。

算法 4-2 访问控制策略属性键值对集合验证算法$\mathcal{V}_{\mathbb{A}}$

**输入**：由属性键值对集合组成的集合$\{\mathbb{A}_i\}_{i\in[m]}$，键值对集合$\mathbb{A}_x$；

**输出**：验证结果$r\in\{0,1\}$，其中，如果$r=1$表示验证通过，反之，表示不通过;

```
flag ← 0;      // 将验证结果flag初始化为 0
// 遍历属性键值对集合{M_i}_{i∈[m]}
foreach i in [m]
    flag' ← 1;      // 用于记录中间验证结果
    foreach A in A_i
        // 满足A_i ⊆ A_x
        if A_1! = search(A_x, A_0)
            flag ← 0;
            break;
    // 满足A_i ⊆ A_x
    if flag' = 1
        flag ← flag';
        break;
return flag;      // 返回结果
```

## 4.3 实验评估及安全分析（Experimental Evaluation and Safety Analysis）

### 4.3.1 实验环境

表 4-1 展示了本实验涉及的服务节点的硬件参数及操作系统信息，图 4-4 展示了实验场景架构，在实验中，使用树莓派 4B 作为物联网网关，云节点（上海）作为存储服务节点，云节点（北京）作为区块链网络节点，由本机向树莓派节点发出 HTTP 请求，树莓派节点根据请求中的实体信息从云节点（上海）中获取对应的属性、策略信息，并从云节点（北京）获取相应的根哈希对数据的完整性进行验证，如果验证通过，则进行访问控制验证并执行验证结果。

表 4-1 IoTAC-L2 硬件参数与操作系统信息

Table 4-1 Hardware parameters and operating system information of IoTAC-L2

| 相关节点 | 硬件参数 | 操作系统 |
|---|---|---|
| 笔记本电脑<br>（本机，徐州） | CPU：i5-1135G7（4 核）<br>内存：16GB | WSL2<br>Ubuntu 20.04.3 LTS 64 位 |

| | | |
|---|---|---|
| 云节点<br>（北京） | CPU：1 核<br>内存：2GB | Ubuntu Server 20.04 LTS 64 位 |
| 云节点<br>（上海） | 处理器：1 核<br>内存：2GB | Ubuntu Server 20.04 LTS 64 位 |
| 树莓派 4B<br>（徐州） | CPU：Broadcom BCM2711（4 核）<br>内存：4GB | Ubuntu Server 20.04 LTS 64 位 |

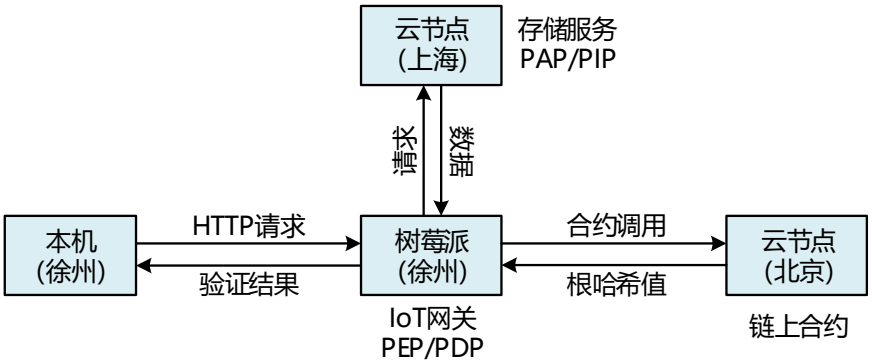



图 4-4 IoTAC-L2 实验场景架构图

Figure 4-4 Experimental scenario architecture diagram of IoTAC-L2

本实验所涉及的相关编译器与开源库版本信息如表 4-2 所示，IoTAC-L2 的软件部分由智能合约、物联网网关服务以及存储服务三部分构成，智能合约部分使用 Solidity 语言编写，使用开源库 Truffle 与 Web3.js 进行部署与交互；物联网网关服务使用 JavaScript 编写，负责监听来自访问者实体的 HTTP 请求并调用存储服务与链上合约获取相应数据，最终完成访问控制逻辑的计算以及访问控制验证结果的执行；存储服务使用 TypeScript 编写，运行在云节点（上海）中，负责将数据以默克尔帕特里夏树的形式存储，并使用 LevelDB 进行持久化，并支持对数据的增加、删除以及修改等操作。

表 4-2 IoTAC-L2 编译器与开源库版本信息

Table 4-2 Compiler and open source library version information of IoTAC-L2

| 相关编译器与开源库 | 版本 |
|---|---|
| Golang | go1.17.6 linux/amd64 |
| Solidity | v0.8.11 |
| tsc | v4.5.5 |
| Node | v16.13.2 |
| Geth | v1.9.25 |
| level | v7.0.1 |
| Truffle | v5.4.32 |
| Web3.js | v1.5.3 |

### 4.3.2 权限维护相关实验

本小节对 IoTAC-L2 中访问者实体的权限维护进行相关实验，主要涉及与权限维护相关的合约的部署、与权限维护相关数据（包括属性及策略信息）的存储、修改以及删除，在 IoTAC-L2 中，对于与权限维护相关的数据的存储、修改以及删除只是修改合约中固定长度的默克尔帕特里夏树根哈希值（256 bit），所以本实验只测试对于链上合约中长度为 256 bit 的哈希值的修改。

（1）合约部署的 gas 消耗

经实验测试，部署合约$\mathbb{C}_{\mathcal{H}}$需要消耗 313817 gas。

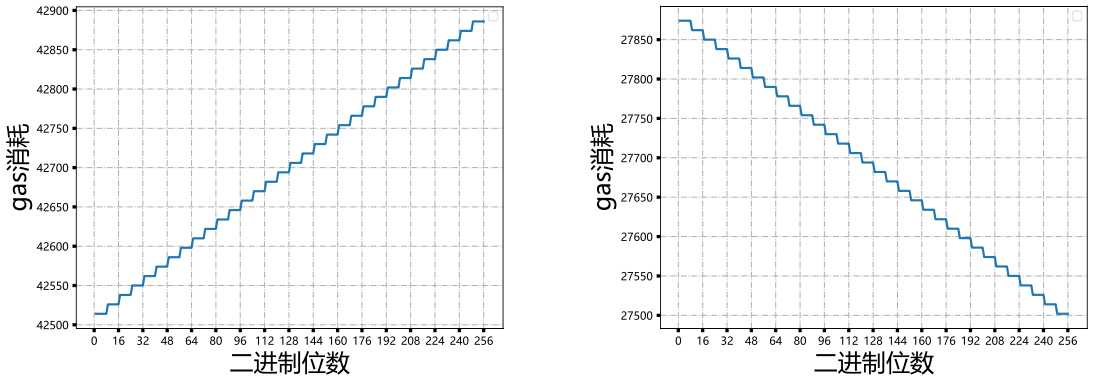
（2）修改链上长度为 256 bit 的哈希值的 gas 消耗

IoTAC-L2 将哈希值以 bytes32 类型存储至链上合约中，哈希值长度为 256 bit，本小节对哈希值的修改进行了如下两个测试实验：

1) 将链上合约中的初始哈希值设置为$h_0 = 0$，每次将其倒数第$i$个二进制位置1，即第$i$次测试的哈希值为$h_i = h_0 + 2^{i-1}$，$i \in [256]$，每次测试将哈希值从$h_0$更改为$h_i$，测试 256 次；

2) 将链上合约中的初始哈希值设置为$h_0 = 2^{256} - 1$，每次将其倒数第$i$个二进制位置0，即第$i$次测试的哈希值为$h_i = h_0 - 2^{i-1}$，$i \in [256]$，每次测试将哈希值从$h_0$更改为$h_i$，测试 256 次。

图 4-5 展示了两次实验的结果，从图中可以看出，修改一条哈希值的 gas 消耗是以字节为单位的（在图中，每 8 个二进制位发生一次跳变），修改全 1 字节越多，gas 消耗会越少，而修改全 0 字节越多，gas 消耗也会越多，在以太坊虚拟机的底层实现中将二进制位从 1 置为 0 要比从 0 置为 1 消耗较少的 gas。




(a) 将二进制位从 0 置 1　　(b) 将二进制位从 1 置 0

图 4-5 gas 消耗与二进制序列长度的关系

Figure 4-5 Gas consumption versus binary sequence length

（3）总结

从上述实验结果可以看出，IoTAC-L2 对于链上存储空间的占用与实际数据长度$L$无关，即 IoTAC-L2 的链上空间复杂度可以表示为$O(1)$，无论实际数据的

大小如何，对于实际数据的修改、删除以及维护等操作只是修改链上合约中的根哈希值，相较于 IoTAC，gas 消耗始终在一个较小的范围内（小于 42900 gas）。综上，IoTAC-L2 在 IoTAC 的基础上实现了较好的可扩展性。

### 4.3.3 访问控制验证相关实验

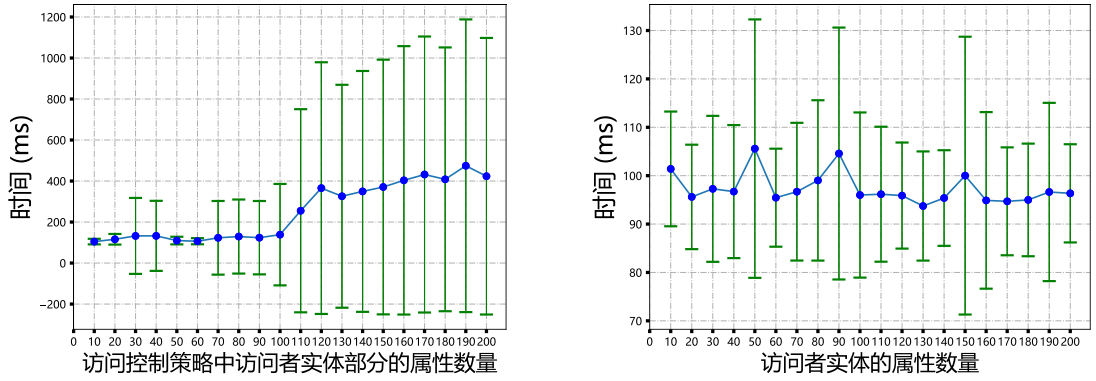
（1）属性数量对访问控制验证时间的影响

本小节对访问控制验证所需的时间与属性数量的关系进行了实验，测试方法与小节（3-4）相同，本实验分为以下两部分：

1) 将$N_S$固定为 200，$N_P^S$从 10 逐渐递增至 200，每次增加 10；

2) 将$N_P^S$固定为 10，$N_S$从 10 逐渐递增至 200，每次增加 10。

其中，考虑网络延时与服务节点的实行运行状况，每次测试重复 100 次，计算每次测试的均值与方差。从图 4-6（a）展示了实验 1）的实验结果，可以看出访问控制时间的均值与方差皆会随着属性数量的增加而增加，这是由于当属性数量增加时，网络传输的数据量以及访问控制验证的计算量与内存占用皆会显著增加，导致网络延时以及服务节点的实时状态出现波动；图 4-6（b）展示了实验 2）的实验结果，可以看出访问控制验证时间与$N_S$没有明显关系，这是由于物联网网关服务会将属性键值对集合转化为映射表。另外有一点需要说明，实验结果是有局限性的，实验结果与实时网络延迟、机器实时运行状况以及代码设计等皆有关系。




(a) (b)

图 4-6 IoTAC-L2 访问控制验证时间与属性数量的关系

Figure 4-6 Access control verification time versus number of attributes in IoTAC-L2

（2）IoTAC-L2 整体时间性能测试

本小节对 IoTAC-L2 中一次完整的访问控制验证所涉及的各个过程进行了时间测试，其中，$N_S = 20$，$N_O = 20$，$N_P^O = 20$，$N_P^S = 20$，测试项如下：

1) 完成一次完整的访问控制验证所需的时间；

2) 树莓派节点至云节点（北京）的网络延时；

3) 树莓派节点至云节点（上海）的网络延时；

4) 树莓派节点向云节点（上海）请求访问控制验证所需的所有数据；

5) 树莓派节点向云节点（北京）请求访问控制验证所需的所有根哈希值；

6) 完成访问控制验证所需的所有数据的默克尔验证。

每次测试重复 100 次，计算每次测试的均值与方差，实验结果的具体数据如表 4-3 所示，统计图如图 4-7 所示，从实验结果中可以看出，在树莓派节点与云节点（北京）以及云节点（上海）之间的网络延时均值分别为 33.3ms 与 23.9ms，方差分别为 12.1ms 与 7.4ms 的情况下，IoTAC-L2 完成一次访问控制验证的所用时间的均值为 106.2ms，方差为 17.2ms，大部分时间用于请求相关数据与根哈希值，默克尔验证与访问控制逻辑的执行用时较少。

表 4-3 IoTAC-L2 时间性能测试结果

Table 4-3 IoTAC-L2 Time Performance Test Results

| 编号 | 测试项 | 均值 | 方差 |
|---|---|---|---|
| 1 | 访问控制验证 | 106.2ms | 17.2ms |
| 2 | 访问者实体身份验证 | 101.4ms | 12.0ms |
| 3 | 树莓派节点至云节点（北京）网络延时 | 33.3ms | 12.1ms |
| 4 | 树莓派节点至云节点（上海）的网络延 | 23.9ms | 7.4ms |
| 5 | 树莓派节点向云节点（上海）请求数据 | 51.16ms | 8.8ms |
| 6 | 树莓派节点向云节点（北京）请求根哈希值 | 48.3ms | 8.3ms |
| 7 | 所有数据的默克尔验证 | 6.5ms | 4.9ms |

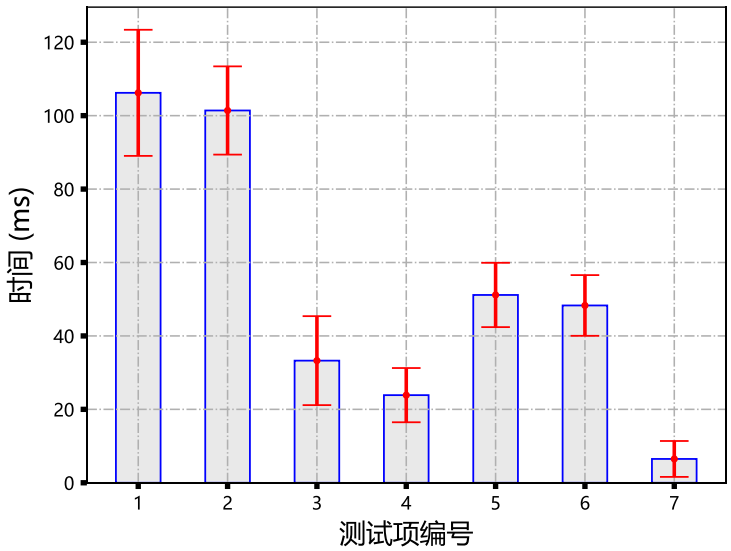



图 4-7 IoTAC-L2 时间性能测试结果

Figure 4-7 IoTAC-L2 time performance test results

（3）比较

文献[18]与文献[6]分别实现了基于以太坊的物联网访问控制模型与物联网架构，两者对于一次请求的处理时间皆为秒级，而 IoTAC-L2 对于一次请求的处理时间为毫秒级。

（4）总结

从上述实验结果可以看出，当$N_S = 20$，$N_O = 20$，$N_P^O = 20$，$N_P^S = 20$时，对于一次完整的访问控制验证，IoTAC-L2 约需 100ms，但当属性、策略数据较大时，平均约需 400ms，这是由于实验所用云节点的性能及带宽受限导致的，从上述实验结果也可以看出，IoTAC-L2 完成一次访问控制验证的时间主要用于向云节点请求相关数据，访问控制逻辑的实际执行时间很短，所以这一点可以通过提高云节点的性能加以改善，并且在实际应用场景中，可以对访问控制策略进行细粒度划分以降低每个策略的数据量，所以 IoTAC-L2 能够满足实际应用场景的需求。

另外，相较于 IoTAC，IoTAC-L2 虽然降低了对链上计算与存储资源的占用，但是 IoTAC-L2 在链下需要完成更多的计算（主要包括对默克尔帕特里夏树的维护、生成以及验证默克尔证明以及访问控制逻辑的计算等）以及在网络中传输更多的数据，IoTAC-L2 在本质上是通过使用更多的链下资源以减少对链上资源的占用以及 gas 消耗。

### 4.3.4 安全性分析

与小节（3.4.4）相同，本小节从外部风险与内部风险两个方向对 IoTAC-L2 的安全性进行分析。

对于外部风险，与 IoTAC 类似，恶意攻击者能够通过访问控制验证需要满足以下两个要求，第一，恶意访问者需要将自身的属性信息添加至存储服务中，IoTAC-L2 同样是由资源拥有者自主管理的访问控制模型，只有资源拥有者能够对存储服务中的数据进行修改；第二，恶意访问者需要修改链上合约中访问者实体属性信息对应的默克尔帕特里夏树根哈希值，链上合约的安全性由以太坊保障，除非资源拥有者操作不当导致账户密钥泄露，否则恶意攻击者难以完成。

对于内部风险，如果恶意攻击者已知某个已授权访问者实体的签名私钥，恶意攻击者能够访问指定的物联网设备的前提是获知该物联网设备及操作对应的具体标识信息，但 IoTAC-L2 为保证匿名性，存储服务只存储实际数据的哈希值，所以恶意攻击者无法从存储服务中获知相关信息，仍然无法访问该物联网设备。

另外，为保证安全性，在 IoTAC-L2 中，访问者的权限同样具有时效性。

# 5 基于区块链与简短非交互零知识证明的物联网访问控制隐私保护研究

# 5 Research on IoT Access Control Privacy Protection Based on Blockchain and zk-SNARK

在 IoTAC-L2 中，虽然资源拥有者能够自行管理隐私信息，但资源拥有者能够从物联网网关中获知访问者实体的具体访问信息，本章希望在 IoTAC-L2 的基础上构建一个遵循“Privacy By Design”设计原则[78]，能够为访问者实体的身份隐私提供保障的访问控制模型，该访问控制模型应当符合如下描述：

1) 访问控制系统能够验证访问者身份的有效性；
2) 访问者无法欺骗访问控制系统；
3) 访问控制系统无法获知关于访问者身份的任何信息。

基于上述目的，本章在 IoTAC-L2 的基础上引入 zk-SNARK，zk-SNARK 能够在保证计算的正确性的同时，实现计算的隐私性及可验证性，由于 zk-SNARK 以概率性可检查证明（Probabilistically Checkable Proof，PCP）定理为基础，能够将复杂的 NP 问题转化为检验有限数量的随机值，所以 zk-SNARK 能够提高访问控制的验证速率。本章构建了一个实现了 ABAC 范式，并且能够隐匿访问者实体身份信息的访问控制模型，将其命名为 IoTAC-ZK。

## 5.1 基于区块链与简短非交互零知识证明的物联网访问控制模型设计（IoT Access Control Model Design Based on Blockchain and zk-SNARK）

### 5.1.1 实现思路及分析

本小节将对如何将 IoTAC-L2 拓展至 IoTAC-ZK，以及 IoTAC-ZK 的具体实现思路进行介绍。首先对 IoTAC-L2 的访问控制逻辑进行更进一步的分析，IoTAC-L2 的访问控制逻辑$\mathcal{L}$可以划分为以下三个步骤：

1) 验证访问者实体$s$身份的有效性，即验证访问者实体$s$提供的签名$sig_s$是否有效；
2) 验证访问者实体$s$是否符合所请求的操作$a$对应的访问控制策略$P_a$；
3) 验证访问者实体$s$所请求的资源实体$o$是否符合所请求的操作$a$对应的访问控制策略$P_a$。

IoTAC-ZK 的目标是访问者能够在不泄露自身任何身份信息的前提下，向访问控制系统证明自身身份的合法性，所以 IoTAC-ZK 的关注点是 IoTAC-L2 访问控制

逻辑的第一步与第二步，即在这两步中实现 zk-SNARK，图 5-1 展示了 IoTAC-ZK 的实现思路。另外，IoTAC-ZK 使用 zk-SNARK 的开源实现 ZoKrates[90]。

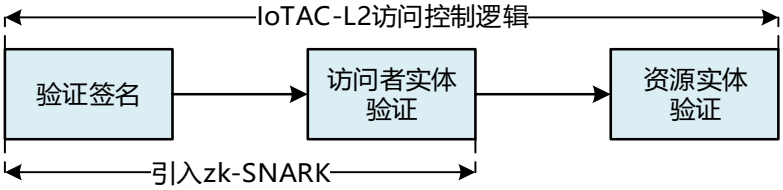



图 5-1 IoTAC-ZK 实现思路示意图

Figure 5-1 IoTAC-ZK implementation idea schematic

对于第一步，ZoKrates 支持爱德华兹曲线数字签名算法（Edwards-curve Digital Signature Algorithm，EdDSA），但是访问者实体还需要给出签名公钥，并且证明该公钥的有效性，IoTAC-ZK 的实现思路是访问控制系统将访问者实体的公钥存储为一个默克尔树，不选择默克尔帕特里夏树有以下两个原因：

1) 相较于默克尔帕特里夏树，基于默克尔树的默克尔证明在 ZoKrates 中更易实现；

2) 访问控制系统不需要提供公钥的搜索功能。

图 5-2 展示了访问者实体身份有效性验证的实现思路，访问者实体需要将自身的公钥、公钥的默克尔证明以及签名作为输入，访问控制系统需要将公钥默克尔树的根哈希值作为输入。

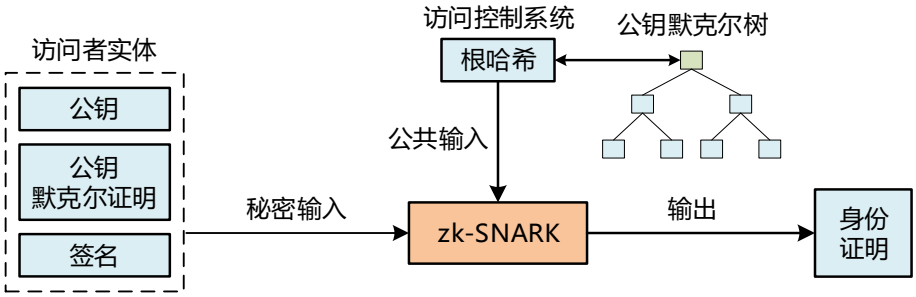



图 5-2 访问者实体身份有效性验证的实现思路

Figure 5-2 Visitor entity identity validation implementation ideas

对于第二步，验证访问控制策略中访问者实体部分$\{\mathbb{A}_i^s : i \in [m]\}$是否存在一个属性键值对集合$\mathbb{A}_k^s$为访问者实体$s$的属性键值对集合$\mathbb{A}_s$的子集，即$\mathbb{A}_k^s$中每一个属性键值对都属于$\mathbb{A}_s$，这个过程可以视为一系列的关系证明（Proof of Membership），可以表示为：

$$\exists \mathbb{A}_k^s \in \{\mathbb{A}_i^s : i \in [m]\} : \forall A_a^s \in \mathbb{A}_k^s : A_a^s \in \mathbb{A}_s \tag{5-1}$$

其中，关系证明有默克尔证明与布隆过滤器两种实现思路，详细介绍如下：

1）对于默克尔证明，默克尔证明不会出现误判情况，但验证默克尔证明需要进行多次哈希计算，并且需对访问控制策略中访问者实体部分的每个属性键值对完成默克尔证明的验证，另外，由于默克尔证明的大小与默克尔树的高度成正

比，空间复杂度可以表示为$O(log_2 N)$，其中，$N$为默克尔树中的元素数量，所以第二步如果使用默克尔证明，时间复杂度与空间复杂度皆较高。

2）对于布隆过滤器，检查布隆过滤器的时间复杂度极低，因为布隆过滤器能够通过简单的位运算进行校验，并且布隆过滤器的大小是常数，与插入元素的数量无关，即空间复杂度为$O(1)$，虽然布隆过滤器存在误判的概率，但可以通过调整构造参数将误报率降至极小。

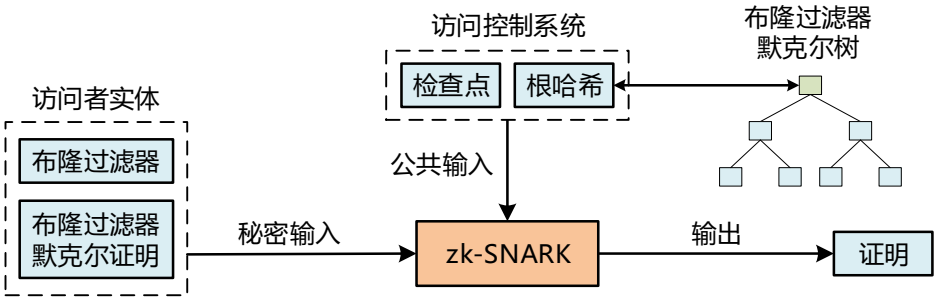



图 5-3 访问者实体属性验证的实现思路

Figure 5-3 Visitor entity attribute verification implementation ideas

综上，IoTAC-ZK 对于第二步使用布隆过滤器，IoTAC-ZK 也需将所有访问者实体的布隆过滤器以默克尔树的形式存储，图 5-3 展示了访问者实体属性验证的实现思路，访问者实体需要将自身的布隆过滤器、布隆过滤器的默克尔证明作为输入，访问控制系统需要将布隆过滤器的检查点以及布隆过滤器默克尔树的根哈希值作为输入。

为方便下文表述，本文将上述两个步骤中的 zk-SNARK 模型定义为$\mathcal{Z}$，对于$\mathcal{Z}$的定义详见公式（2-79）至（2-81）。

### 5.1.2 布隆过滤器设计

布隆过滤器（Bloom Filter）于 1970 年由 Burton Howard Bloom 提出，布隆过滤器实际上是一个很长的二进制向量和一系列随机映射函数，布隆过滤器可以用于检索一个元素是否在一个集合中，布隆过滤器的优点是空间效率和查询时间都远远超过一般的算法，缺点是有一定的误识别率以及删除困难[89]。

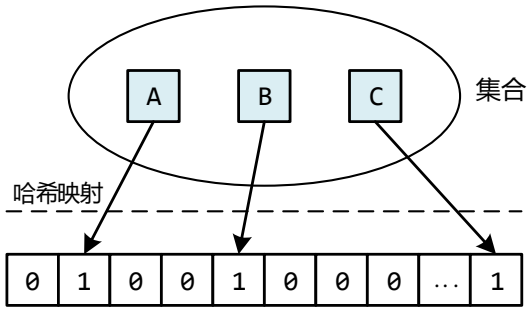



图 5-4 布隆过滤器示意图

Figure 5-4 Bloom filter schematic

图5-4展示了布隆过滤器的工作原理，布隆过滤器通常使用二进制序列表示，当一个元素被加入集合中时，通过$K$个哈希函数将这个元素映射成二进制序列中的$K$个点（检查点），并将这些点置为1（图中$K=1$），当在集合中检索某个元素时，只需验证检查点是否皆为1就能够（大约）知晓集合是否包含此元素：如果检查点中有任何一个点为0，则此元素一定不在集合中；如果检查点皆为1，则被检元素很可能在集合中。

布隆过滤器的误报率$\varepsilon$可以表示为公式（5-2），其中，$M$为布隆过滤器的长度（即二进制序列的长度），$K$为哈希函数的数量，$n$为插入元素的数量。在IoTAC-ZK中，$M=2^{10}=1024$，$K=8$，布隆过滤器可以容纳的哈希值范围是$[0,2^{10}-1]$，图 5-5 展示了布隆过滤器误报率的仿真结果，从图中可以看出，当$M=1024$，$K=8$，$n\leq 20$时，误报率$\varepsilon<2\times10^{-7}$，此时布隆过滤器能够满足实际需求。由于哈希函数可以视为一个伪随机数预言机，IoTAC-ZK 使用哈希函数 SHA256 计算访问者实体的属性键值对$A=(A_0,A_1)$生成的哈希值$h_A$，计算方法如公式（5-3）所示，按照一定规则使用哈希函数 SHA256 的输出结果$h_A$生成$K$个检查点。

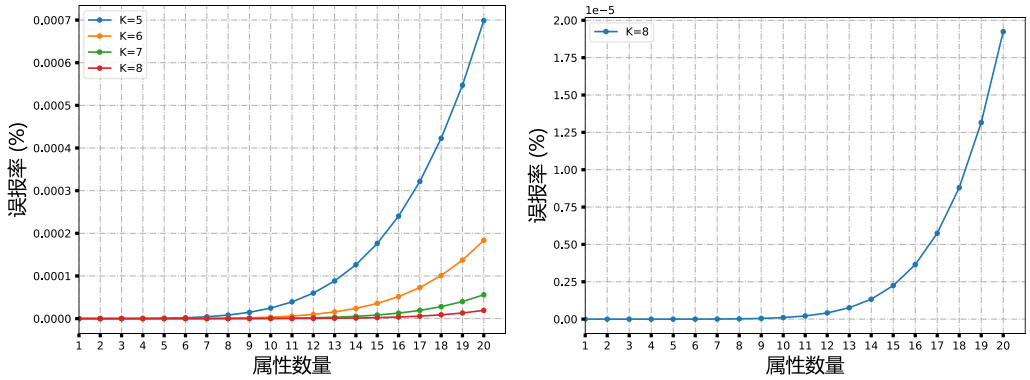
$$\varepsilon=\left(1-\left[1-\frac{1}{M}\right]^{K\cdot n}\right)^{K} \tag{5-2}$$

$$h_A=\mathrm{SHA256}(A_0\mid A_1) \tag{5-3}$$

其中，检查点的计算规则如下：

$$cp_k=\sum_{i=0}^{\lfloor log_2 M\rfloor-1} h_A[k+K\cdot i]\cdot 2^i \tag{5-4}$$

其中，$cp_k$表示第$k$个检查点，$k\in\{0,1,\ldots,K-1\}$，$h_A[x]$表示$h_A$第$x$个二进制位，$x\in\{0,1,\ldots,255\}$。




(a) 误报率与插入元素数量的关系　　(b) $K=8$ 时误报率与插入元素数量的关系

图 5-5 布隆过滤器误报率模拟图

Figure 5-5 Bloom filter false alarm rate simulation chart

### 5.1.3 模型设计

在 IoTAC-ZK 中，存储服务将每个访问者实体对应的公钥$PK$与布隆过滤器$filter$依据公式（5-5）存储为一个默克尔树。

$$leaf = (PK \mid filter) \tag{5-5}$$

其中，$leaf$表示默克尔树的叶子节点，“|”运算符表示将公钥$PK$与布隆过滤器$filter$进行拼接。另外，本文使用符号$\mathbb{T}_{\mathcal{S}}$表示上述默克尔树。

本文将 IoTAC-ZK 的实现范式定义为如下：

$$R = (\pi, id_o, id_a), \qquad o \in \mathcal{O} \wedge a \in \mathcal{A} \tag{5-6}$$

$$\boldsymbol{D} = \left((\mathcal{M}_{\mathcal{S}}, \mathcal{M}_{\mathcal{O}}, \mathcal{M}_{\mathcal{A}}, \mathbb{T}_{\mathcal{S}}), f_{\boldsymbol{D}}\right) \tag{5-7}$$

$$f_{\boldsymbol{D}}: (k, o, \sigma) \longrightarrow r, \qquad a \in \mathcal{A} \tag{5-8}$$

$$\mathcal{L} = (\mathcal{L}_{\mathcal{O}}^{a}, \mathcal{Z}), \qquad a \in \mathcal{A} \tag{5-9}$$

$$\mathcal{L}_{\mathcal{O}}^{a}: (id_o, id_a, \boldsymbol{D}) \longrightarrow \{0,1\}, \qquad o \in \mathcal{O} \wedge a \in \mathcal{A} \tag{5-10}$$

$$\Upsilon: (\mathcal{L}, \boldsymbol{D}, R) \longrightarrow \{0,1\} \tag{5-11}$$

相较于 IoTAC-L2，IoTAC-ZK 的访问请求$R$由公式（3-8）转变为公式（5-6）；数据集合$\boldsymbol{D}$在公式（3-10）的基础上加入了默克尔树$\mathbb{T}_{\mathcal{S}}$；访问控制逻辑$\mathcal{L}$中加入了基于 zk-SNARK 的访问者实体证明与验证模型$\mathcal{Z}$。

遵循“低耦合、高内聚”的模块化程序设计理念，IoTAC-ZK 将上述零知识证明过程拆解为验证身份（验证默克尔证明与签名）与检查布隆过滤器两个过程，通过链上合约$\{\mathbb{C}_{\mathcal{Z}}^{a}\}_{a\in\mathcal{A}}$实现，资源拥有者需事先对每一个访问控制策略$\mathcal{P}_a$进行可信初始化，生成对应的证明、验证密钥对$(key_P^a, key_V^a)$与 zk-SNARK 验证合约$\mathbb{C}_{\mathcal{Z}}^{a}$，然后将$key_P^a$公开给访问者，访问者使用$key_P^a$生成身份证明。另外，对于访问控制策略的每次修改，资源拥有者皆需重新部署对应的$\mathbb{C}_{\mathcal{Z}}^{a}$，资源拥有者需要维护映射表$\mathcal{M}_{\mathcal{Z}}: id_a \longrightarrow (\mathbb{C}_{\mathcal{Z}}^{a}, key_V^a)$。

访问者实体向物联网网关发出访问请求$R = (\pi, id_o, id_a)$，物联网网关调用链上合约$\mathcal{M}_{\mathcal{Z}}(id_a)$对访问者实体提供的身份证明$\pi$进行验证，物联网网关自身负责完成访问控制逻辑$\mathcal{L}_{\mathcal{O}}^{a}(id_o, id_a, \boldsymbol{D})$的计算，最终的访问控制验证结果可以表示为$\mathcal{M}_{\mathcal{Z}}(id_a) \wedge \mathcal{L}_{\mathcal{O}}^{a}(id_o, id_a, \boldsymbol{D})$。

图 5-6 展示了 IoTAC-ZK 的系统架构，相较于 IoTAC-L2，在 IoTAC-ZK 中，存储服务中增加了由访问者实体的公钥与布隆过滤器相关信息构成的默克尔树，并且链上合约$\mathbb{C}_{\mathbb{T}}$负责维护该默克尔树的根哈希值；链上合约中增加了 zk-SNARK 验证合约集合$\{\mathbb{C}_{\mathcal{Z}}^{a}\}_{a\in\mathcal{A}}$，用于完成零知识证明相关验证逻辑的计算。在 IoTAC-ZK 中，由物联网网关与链上合约同时作为 ABAC 访问控制范式中的策略决策点（PEP），物联网网关负责完成对资源实体的验证，链上合约负责完成对访问者实体的验证。

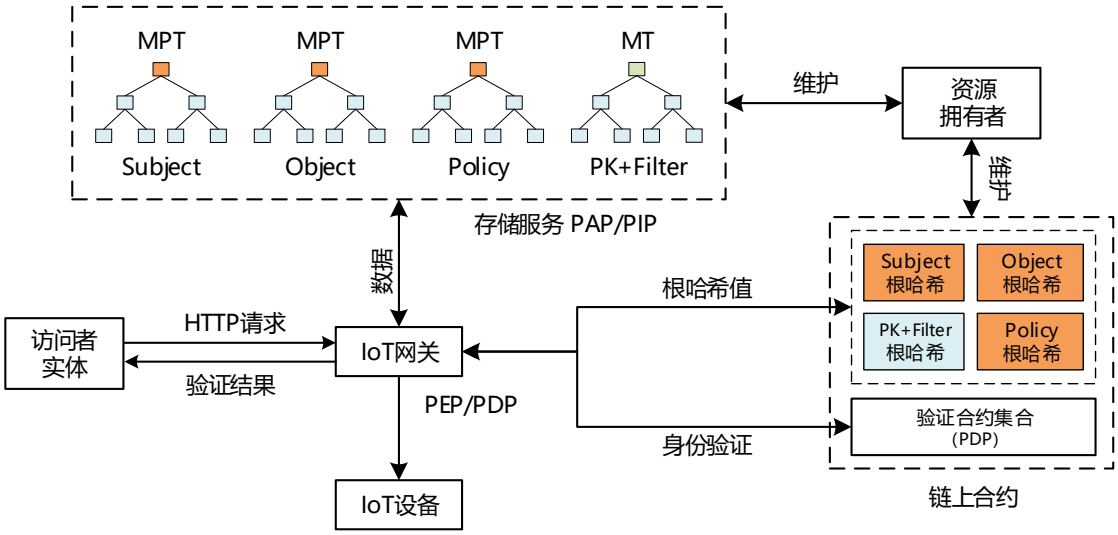



图 5-6 IoTAC-ZK 系统结构图

Figure 5-6 IoTAC-ZK system architecture schematic

### 5.1.4 隐私性分析

在 IoTAC-ZK 中，资源拥有者无法从请求$R$（见公式（5-6））中推断出任何关于访问者的任何身份信息，在$R$中，与访问者身份有关的信息为访问者提供的身份证明$\pi$，访问者使用证明密钥$key_P^a$与身份信息$\vec{w}$生成身份证明$\text{Proof}(key_P^a, \vec{w}) = \pi$（详见小节 2.4.6 中的公式（2-80）），在 zk-SNARK 中，Proof 为单向函数，资源拥有者无法根据身份证明$\pi$逆推出访问者的身份信息$\vec{w}$，所以资源拥有者无法获知访问者的身份信息。

## 5.2 访问控制算法设计（Access Control Algorithm Design）

IoTAC-ZK 的访问控制逻辑$\mathcal{L}$由物联网网关与验证合约共同负责执行，为方便下文表述，本文使用函数$\mathcal{V}_Z: (\mathbb{C}_Z^a, key_V^a, \pi) \longrightarrow \{0,1\}$表示使用合约$\mathbb{C}_Z^a$对请求中的证明$\pi$进行 zk-SNARK 验证，算法 5-1 展示了访问控制逻辑$\mathcal{L}$的具体流程。

算法 5-1 IoTAC-ZK 的访问控制逻辑$\mathcal{L}$

输入：请求$R = (\pi, id_o, id_a)$;

输出：权限验证结果$r \in \{0,1\}$，其中，如果$r = 1$表示访问控制验证通过，反之，表示不通过;

1. // 从$\mathbb{T}_{\mathcal{P}}$中获取操作$a$对应的访问控制策略$\left(\{\mathbb{A}_i^S\}_{i\in[m]}, \{\mathbb{A}_i^O\}_{j\in[n]}\right)$与默克尔证明$\pi_a$
2. $\left(\left(\{\mathbb{A}_i^S\}_{i\in[m]}, \{\mathbb{A}_i^O\}_{j\in[n]}\right), \pi_a\right) \leftarrow f_D(\mathbb{T}_{\mathcal{P}}, "search", (id_a))$;
3. // 从合约$\mathbb{C}_{\mathcal{H}}$中获取$\mathbb{T}_{\mathcal{P}}$的根哈希值$h_{\mathcal{P}}$
4. $h_{\mathcal{P}} \leftarrow f_D((\mathbb{C}_V, \mathbb{T}_{\mathcal{P}}), "search", ())$;
5. // 验证访问控制策略$\left(\{\mathbb{A}_i^S\}_{i\in[m]}, \{\mathbb{A}_i^O\}_{j\in[n]}\right)$是否正确

6. **if** $\mathcal{V}_{\pi}\left(\left(id_a,\left(\{\mathbb{A}_i^{\mathcal{S}}\}_{i\in[m]},\{\mathbb{A}_i^{\mathcal{O}}\}_{j\in[n]}\right)\right),\pi_a,h_{\mathcal{P}}\right)!=1$
7. **return** 0;
8. // 获取操作$a$对应的验证合约$\mathbb{C}_Z^a$与验证密钥$key_V^a$
9. $(\mathbb{C}_Z^a,key_V^a)\leftarrow\mathcal{M}_Z(id_a)$
10. // 验证访问者实体的身份证明是否正确
11. **if** $\mathcal{V}_Z\left(\mathbb{C}_Z^a,key_V^a,\pi\right)!=1$
12. **return** 0;
13. // 从$\mathbb{T}_{\mathcal{O}}$中获取资源实体$o$的属性键值对集合$\mathbb{A}_o$与默克尔证明$\pi_o$
14. $(\mathbb{A}_o,\pi_o)\leftarrow f_{\boldsymbol{D}}((\mathbb{T}_{\mathcal{O}},id_o),"search",())$;
15. // 从合约$\mathbb{C}_{\mathcal{H}}$中获取$\mathbb{T}_{\mathcal{O}}$的根哈希值$h_{\mathcal{O}}$
16. $h_{\mathcal{O}}\leftarrow f_{\boldsymbol{D}}((\mathbb{C}_{\mathcal{V}},\mathbb{T}_{\mathcal{O}}),"search",())$;
17. // 验证$\mathbb{A}_o$是否正确
18. **if** $\mathcal{V}_{\pi}\left((id_o,\mathbb{A}_o),\pi_o,h_{\mathcal{O}}\right)!=1$
19. **return** 0;
20. // 验证资源实体$o$是否满足访问控制策略
21. **if** $\mathcal{V}_{\mathbb{A}}\left(\{\mathbb{A}_i^{\mathcal{O}}\}_{j\in[n]},\mathbb{A}_o\right)!=1$
22. **return** 0;
23. **return** 1; // 验证通过

## 5.3 实验评估及安全分析（Experimental Evaluation and Safety Analysis）

### 5.3.1 实验环境

表 5-1 IoTAC-ZK 硬件参数与操作系统信息

Table 5-1 Hardware parameters and operating system information of IoTAC-ZK

| 相关节点 | 硬件参数 | 操作系统 |
|---|---|---|
| 笔记本电脑<br>（本机，徐州） | CPU：i5-1135G7（4 核）<br>内存：16GB | WSL2<br>Ubuntu 20.04.3 LTS 64 位 |
| 云节点<br>（北京） | CPU：1 核<br>内存：2GB | Ubuntu Server 20.04 LTS 64 位 |
| 云节点<br>（上海） | 处理器：1 核<br>内存：2GB | Ubuntu Server 20.04 LTS 64 位 |
| 树莓派 4B<br>（徐州） | CPU：Broadcom BCM2711（4 核）<br>内存：4GB | Ubuntu Server 20.04 LTS 64 位 |

表 5-1 展示了本实验涉及的服务节点的硬件参数及操作系统信息，图 5-7 展示了本实验的场景架构，其中，树莓派 4B 节点作为物联网网关，云节点（北京）作为区块链网络节点，云节点（上海）作为存储服务节点，由本机向树莓派节点发出 HTTP 请求，树莓派节点对请求进行解析，并向云节点（北京）发出请求，

根据操作标识调用相应的验证合约验证访问者提供的身份证明，然后，树莓派节点向两个云节点发出请求获取相关信息，完成对资源实体的访问控制验证。

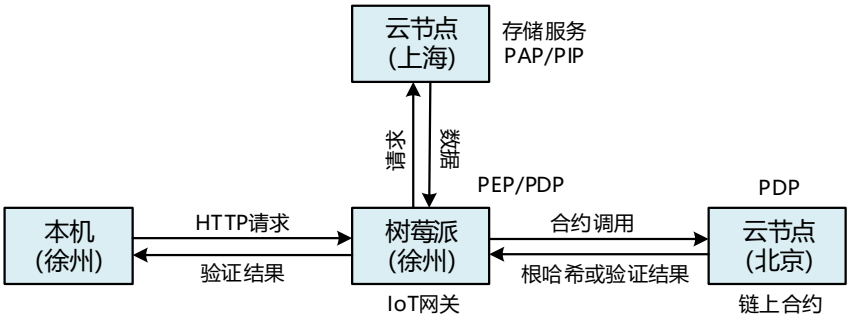



图 5-7 IoTAC-ZK 实验场景架构图

Figure 5-7 Experimental scenario architecture diagram of IoTAC-ZK

本实验涉及的相关编译器与开源库版本如表 5-2 所示，IoTAC-ZK 的软件部分与 IoTAC-L2 相同，由智能合约、物联网网关服务以及存储服务三部分构成，IoTAC-ZK 使用开源库 ZoKrates 实现访问者实体的身份验证功能，包括可信初始化（生成证明、验证密钥）以及生成验证合约等。

表 5-2 IoTAC-ZK 相关编译器与开源库版本

Table 5-2 Compiler and open source library version information of IoTAC-ZK

| 相关编译器与开源库 | 版本 |
|---|---|
| Golang | go1.17.6 linux/amd64 |
| Solidity | v0.8.11 |
| tsc | v4.5.5 |
| rustc | 1.58.1 (db9d1b20b 2022-01-20) |
| ZoKrates | v0.7.11 |
| Node | v16.13.2 |
| Geth | v1.9.25 |
| level | v7.0.1 |
| Truffle | v5.4.32 |
| Web3.js | v1.5.3 |

### 5.3.2 zk-SNARK 功能实现相关实验

本小节分别对使用 ZoKrates 实现默克尔证明的验证、布隆过滤器的检查、EdDSA 数字签名的验证以及访问者实体身份的验证所需要的时间进行了相关测试，ZoKrates 实现相关功能的零知识证明需要完成以下五个步骤[90]：

1) 编译。将事先用高级语言编写的待进行零知识证明的问题编码为电路，这一步也称为代码扁平化，具体原理详见小节（2.4.1）；

2) 可信初始化。将电路依次转化为秩 1 约束系统与二次计算程序，并构建

公共参考字符串（CRS）模型，生成证明密钥与验证密钥，具体原理详见小节（2.4.2）至小节（2.4.6）；

3) 计算见证值。根据证明方输入的问题的解，生成电路的一个完整计算状态（或秩 1 约束系统的输入），包括输入、输出以及所有中间变量，具体原理详见小节（2.4.1）与小节（2.4.2）；

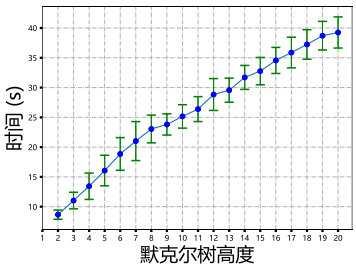
4) 计算证明。使用第三步生成的见证值及第二步生成的证明密钥生成待证明问题的证明，具体原理详见小节（2.4.5）与小节（2.4.6）；

5) 验证。使用第二步生成的验证密钥对第四步生成的证明进行验证，具体原理详见小节（2.4.5）与小节（2.4.6）。

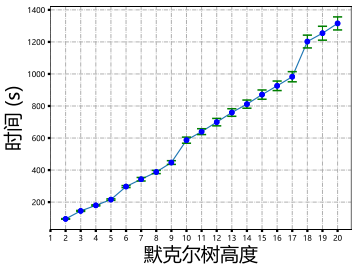
另外，考虑机器实时运行情况的波动，本实验使用本机在不同的时间段重复进行了 10 次测试，计算所有测试结果的均值与方差。

（1）默克尔证明的验证功能

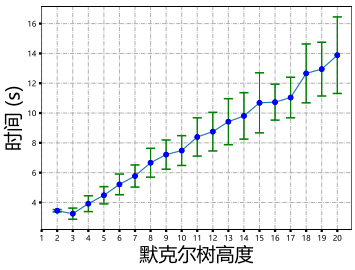
本小节对 ZoKrates 实现默克尔证明验证功能的各阶段时间消耗与默克尔哈希树高度的关系进行了实验，默克尔哈希树的高度从 2 逐渐递增至 20，每次增加 1，当深度为 20 时，默克尔树能够存储$2^{20} = 1048576$个访问者实体的身份信息，已经能够满足实际应用场景。



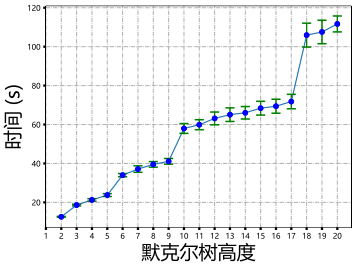

(a) 编译



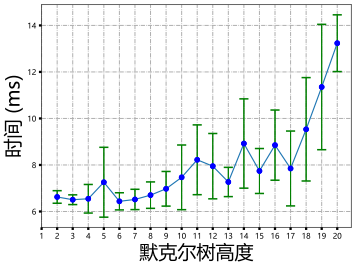

(b) 可信初始化




(c) 计算见证值




(d) 计算证明




(e) 验证

图 5-8 默克尔证明时间消耗

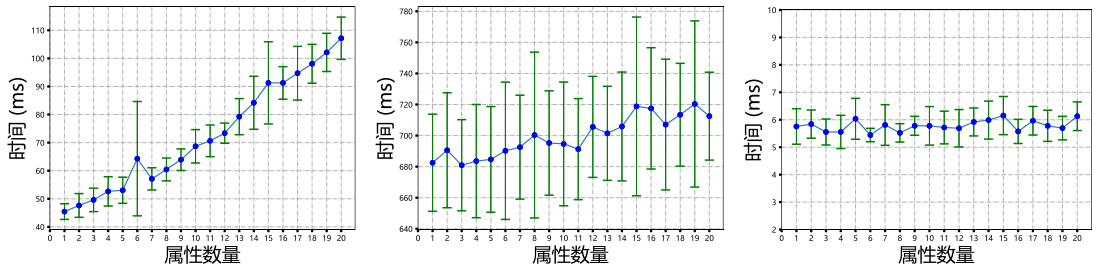
Figure 5-8 Merkle proof time consumption

图 5-8 展示了实验结果及误差线，从图中可以看出，编译、可信初始化、计算验证值以及计算证明的所用时间与默克尔哈希树的高度近似呈线性关系，这是因为验证过程中哈希值的计算次数等于默克尔哈希树的高度，在 zk-SNARK 中，计算哈希值是一个时间复杂度较高的运算过程。虽然前四个阶段用时较长，但是

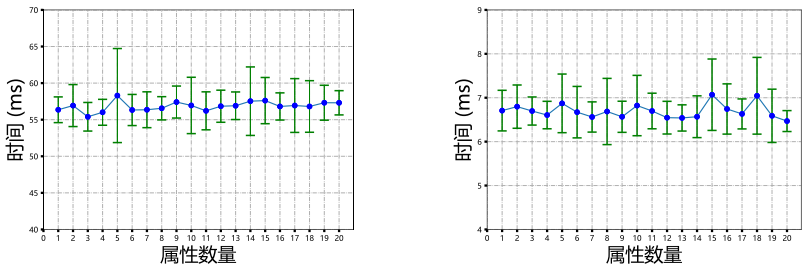
对于验证过程只需要约 5ms 至 15ms，这是因为在 zk-SNARK 中，验证证明的计算复杂度与待证明问题的复杂度无关，验证证明是一个时间复杂度较低的运算过程。

（2）布隆过滤器的检查功能

本小节对使用 ZoKrates 实现布隆过滤器检查功能的各阶段时间消耗与访问控制策略中访问者实体部分的属性数量$N_S^{\mathcal{P}}$的关系进行了相关实验，在 IoTAC-ZK 中，布隆过滤器的哈希函数数量$K=8$，检查点数量$N_{cp}=K\cdot N_S^{\mathcal{P}}=8N_S^{\mathcal{P}}$，属性数量从 1 逐渐递增至 20，每次增加 1。




(a) 编译　(b) 可信初始化　(c) 计算见证值




(d) 计算证明　(e) 验证

图 5-9 布隆过滤器时间消耗

Figure 5-9 Bloom filter time consumption

图 5-9 展示了实验结果及误差线，从图中可以看出，编译及可信初始化阶段的所用时间随着属性数量的增加而增加，这是由于检查点数量等于证明过程中需要校验的二进制位的数量，检查点数量越高，将布隆过滤器的检查逻辑转化为电路以及可信初始化的计算复杂度就越高。对于计算见证值、计算证明以及验证，所用时间基本是稳定的，与属性数量没有明显关系，这是由于检查布隆过滤器某个检查点只需进行位运算即可完成，计算复杂度较低，故对于前 4 个阶段，检查布隆过滤器所需的时间要远小于验证默克尔证明。

编译及可信初始化所消耗的时间会随着属性数量的增加而增加，是因为编译与可信初始化需要对布隆过滤器的检查逻辑进行较为复杂的转化，而对于计算见证值、计算证明以及验证只是在可信初始化的输出的基础上进行较为简单的计算，从各个阶段所消耗的时间也能够看出，ZoKrates 构建布隆过滤器验证功能的计算量主要集中于编译与可信初始化。另外，对于验证时间，检查布隆过滤器与验证

默克尔证明相差不大，这也能够证明，在 zk-SNARK 中对于证明的验证是一个时间复杂度很低并且与待证明问题的复杂度无关的运算过程。

（3）EdDSA 数字签名的验证功能

对于验证签名，实验结果的具体数据如表 5-3 所示，统计图如图 5-10 所示，从实验结果可以看出，ZoKrates 构建 EdDSA 数字签名的时间消耗较少。

表 5-3 EdDSA 时间消耗

Table 5-3 EdDSA time consumption

| 编号 | 测试项 | 均值 | 方差 |
|---|---|---|---|
| 1 | 编译 | 1.7s | 27.7ms |
| 2 | 可信初始化 | 17.8s | 370.1ms |
| 3 | 计算见证值 | 541.6ms | 16.6ms |
| 4 | 生成证明 | 648.9ms | 14.0ms |
| 5 | 验证证明 | 4.3ms | 0.3ms |

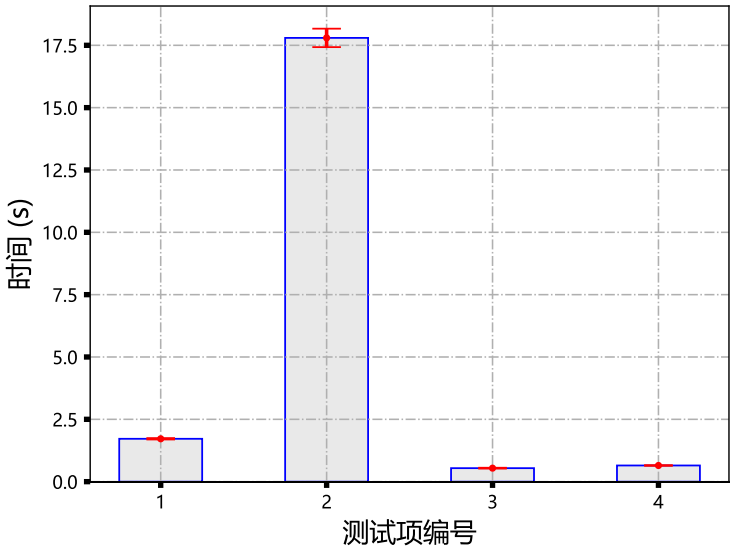



图 5-10 EdDSA 时间消耗统计图

Figure 5-10 EdDSA time consumption statistics chart

（4）访问者实体身份验证功能

对于访问者实体的身份验证，在本实验中，默克尔树的高度为 10，访问控制策略访问者实体部分的属性数量$N_p^{\delta} = 20$。实验结果的具体数据如表 5-4 所示，统计图如图 5-11 所示，从实验结果可以看出，编译及可信初始化用时较长，所用时间的均值分别为 47.4s 与 725.8s，方差分别为 1.8s 与 7.3s，所以资源拥有者在部署访问控制策略时，约需要 13 分钟生成该策略的 zk-SNARK 相关功能，但此过程仅需进行一次；计算见证值与生成证明所用时间的均值分别为 23.7s 与 94.9s，方差分别为 2.2s 与 2.9s，所以访问者约需要 2 分钟生成自身的身份证明，

但此过程同样只需进行一次；验证时间极短，所需时间的均值为 8.0ms，方差为 1.0ms。

表 5-4 访问者实体身份验证时间消耗

Figure 5-4 Visitor entity authentication time consumption

| 编号 | 测试项 | 均值 | 方差 |
|---|---|---|---|
| 1 | 编译 | 47.4s | 1.8s |
| 2 | 可信初始化 | 725.8s | 7.3s |
| 3 | 计算见证值 | 23.7s | 2.2s |
| 4 | 计算证明 | 94.9s | 2.9s |
| 5 | 验证证明 | 8.0ms | 1.0ms |

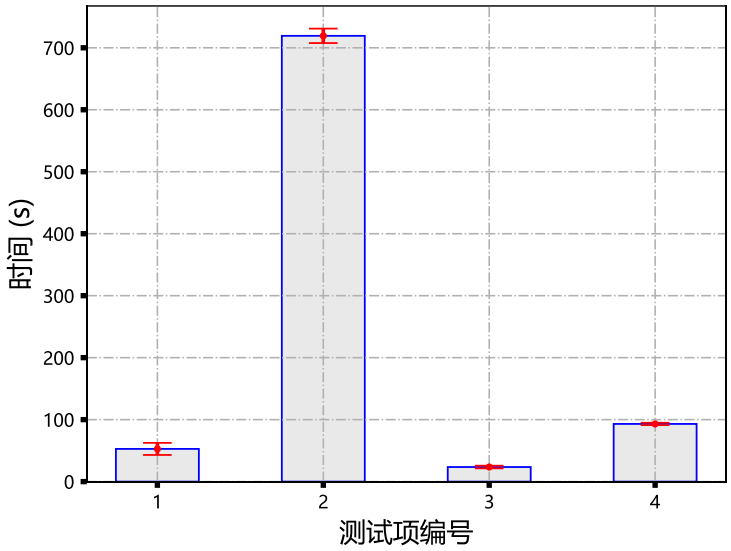



图 5-11 访问者实体身份验证时间消耗统计图

Figure 5-11 Visitor entity authentication time consumption statistics chart

（5）总结

从上述实验结果可以看出，对于使用 ZoKrates 构建访问者实体身份验证的 zk-SNARK 功能，由于哈希计算的复杂性，构建默克尔证明的验证功能用时较长，而对于 EdDSA 签名以及布隆过滤器，用时较短。但在 ABAC 范式中，如公式（2-5）所示，访问控制系统需要使用内部数据（即属性、策略信息）对访问请求进行验证，即验证访问请求中的信息与系统内部数据是否具有某种集合关系，虽然布隆过滤器也能够完成集合关系的验证，但布隆过滤器的误报率会随着插入元素数量的增加的升高，并且布隆过滤器不具有删除功能，从而影响可扩展性，所以使用默克尔证明是必要的，幸运的是，构建访问者实体身份验证的 zk-SNARK 功能只需在访问控制系统初始化时进行一次，所以本节提出的访问者实体隐私保护思路是可以应用于实际场景的。

### 5.3.3 访问者实体身份验证相关实验

在本实验中，将默克尔树的高度设置为10，此时访问控制系统可以存储$2^{10}=1024$个访问者实体的身份信息，访问控制策略访问者实体部分的属性数量$N_p^S$设置为20。

（1）合约部署的 gas 消耗

本小节使用 ZoKrates 生成访问者实体身份验证的零知识证明智能合约，并将合约部署至链上，经实验，将该合约部署至链上需要消耗 1418687 gas。

（2）IoTAC-ZK 时间性能测试

对于验证合约完成一次完整的访问者实体身份验证所需要的时间，本实验重复 100 次，计算所有结果的均值与方差，在最终结果中，均值为 64.6ms，方差为 14.5ms。

表 5-5 访问控制模型性能比较

Figure 5-5 Access control model performance comparison

| 模型 | 均值 | 方差 |
|---|---|---|
| IoTAC-ZK | 64.6ms | 14.5ms |
| IoTAC-L2 | 101.4ms | 12.0ms |
| IoTAC | 140.1s | 18.9ms |

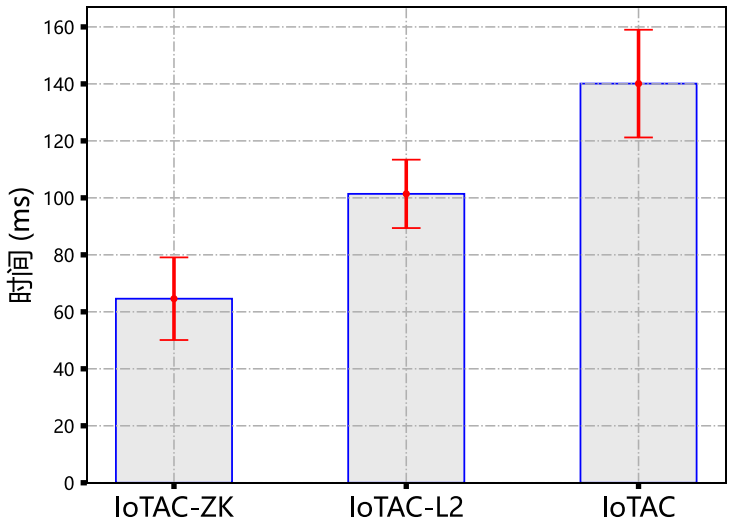



图 5-12 访问控制模型性能比较统计图

Figure 5-12 Access control model performance comparison statistics chart

（3）比较

文献[23]实现了基于以太坊与 zk-SNARK 的物联网访问控制模型 BZBAC，对于一次请求的处理时间，BZBAC 为秒级，而 IoTAC-ZK 为毫秒级，但 BZBAC 的初始化时间较短，所用时间为秒级，而 IoTAC-ZK 为分钟级。

（4）总结

本小节对 IoTAC-ZK、IoTAC-L2 以及 IoTAC 完成一次完整的访问者实体身份验证[①]所需要的时间进行比较，具体数据如表 5-5 所示，统计图如图 5-12 所示，可以看出，IoTAC-ZK 不仅能够满足实际应用场景，而且身份验证速率明显优于 IoTAC-L2 以及 IoTAC。

### 5.3.4 安全性分析

与 IoTAC 及 IoTAC-L2 相同，本小节从外部风险与内部风险两个方向对 IoTAC-ZK 的安全性进行分析。

对于外部风险，恶意攻击者能够通过访问控制验证需要满足两个要求，第一，恶意攻击者能够生成某个访问控制策略的证明$\pi$，为生成证明$\pi$，恶意攻击者需要将自身的身份信息（签名公钥与布隆过滤器）添加至存储服务中，IoTAC-ZK 是由资源拥有者自主管理的访问控制模型，只有资源拥有者能够对相关信息进行更改，所以恶意攻击者无法将身份信息添加至存储服务中；第二，恶意攻击者需要修改链上合约中访问者实体属性信息对应的默克尔树根哈希值，除非资源拥有者操作不当导致账户密钥泄露，否则恶意攻击者难以完成。

对于内部风险，如果恶意攻击者能够窃取到某个访问者实体的证明$\pi$，此时恶意攻击者还需要知晓相应物联网设备以及操作的具体标识，而 IoTAC-ZK 同样使用哈希存储方式存储相关数据，恶意攻击者无法获知相关标识，从而恶意攻击者无法向物联网网关发起一次完整的请求。

另外，IoTAC-ZK 应当每隔一段时间更新一次验证合约，以使访问者实体的身份证明具有时效性，进而加强系统的安全性。

①此处“访问者实体身份验证”是指只验证访问者实体的身份信息是否满足访问控制策略。

# 6 总结与展望

# 6 Conclusion and Future Work

## 6.1 工作总结（Conclusion）

本文重点围绕物联网访问控制这一主题，对于基于区块链与零知识证明的物联网访问控制进行了研究。现今，物联网技术快速发展，安全与隐私问题也随之而来，访问控制是物联网安全与隐私的重要保障。区块链与零知识证明给物联网访问控制的发展带来了新的契机，区块链能够在没有可信第三方的条件下有效地解决物联网的信任问题，而简短非交互零知识证明能够在实现隐私保护的同时提升验证效率。

本文基于上述动机开展研究，本文主要完成的工作包括以下几个方面：

（1）本文提出了基于智能合约的物联网访问控制模型（IoTAC），旨在解决传统物联网访问控制模型需要将核心功能委托至可信第三方的问题。IoTAC 基于以太坊平台实现了基于属性的访问控制范式，使用区块链智能合约技术实现访问控制逻辑以及属性、策略等信息的存储，由于区块链网络的去中心化特性，IoTAC 不再需要可信第三方的支持。在 IoTAC 中，物联网网关负责接收访问者实体的请求，并将请求信息交由链上合约完成访问控制验证，最后由物联网网关执行验证结果。本文从权限维护与访问控制验证两个角度，对 IoTAC 的性能进行了广泛测试，测试结果表明，对于验证速率，IoTAC 能够满足实际应用场景，但对于权限维护，由于 IoTAC 严重依赖链上资源及链上资源的昂贵性，IoTAC 的可扩展性受限。

（2）本文提出了基于双层区块链架构的物联网访问控制模型（IoTAC-L2），旨在解决 IoTAC 的可扩展性问题。IoTAC-L2 在 IoTAC 的基础上实现了高可扩展性，IoTAC-L2 将属性及策略信息以默克尔帕特里夏树的形式存储至链下，链上合约只存储默克尔帕特里夏树的根哈希值，通过默克尔证明保证链下数据的完整性。由于根哈希值长度为较小的定值，与链下数据的实际大小无关，并且访问控制逻辑的计算由物联网网关完成，所以 IoTAC-L2 能够释放链上资源，提高访问控制模型的可扩展性。本文从权限维护与访问控制验证两个角度，对 IoTAC-L2 的性能进行了广泛测试，测试结果表明，对于验证速率及权限维护，IoTAC-L2 皆能够满足实际应用场景，但在 IoTAC-L2 中，资源拥有者能够从物联网网关中获知访问者实体的访问信息。

（3）本文提出了基于区块链与简短非交互零知识证明的物联网访问控制模型（IoTAC-ZK），旨在解决 IoTAC-L2 的隐私问题。在 IoTAC-ZK 中，访问者实

体能够在无法欺骗访问控制系统的前提下，向访问控制系统证明自身身份的合法性，并且访问控制系统无法获知关于访问者实体身份的任何信息。IoTAC-ZK 将访问者实体的身份验证划分为身份有效性检查与属性检查两个步骤，IoTAC-ZK 通过默克尔证明实现身份有效性检查，通过布隆过滤器实现属性检查，最终使用 ZoKrates 开源工具库实现上述步骤的简短非交互零知识证明。本文对 IoTAC-ZK 的身份验证性能进行了广泛测试，测试结果表明，IoTAC-ZK 不仅能够满足实际应用场景，而且身份验证速率明显优于 IoTAC-L2 以及 IoTAC。

## 6.2 工作展望（Future Work）

本文针对基于区块链与零知识证明的物联网访问控制进行了系统研究，虽然取得了一定成果，但由于时间、条件及经验等限制，很多内容并没有深入研究与探讨，现整理如下：

（1）在 IoTAC-L2 中，在物联网设备数量极大或物联网数据量极大的情况下，链下存储服务可能无法将所有数据组织成一个默克尔帕特里夏树，可能需要组织成多个默克尔帕特里夏树，此时链上合约也需要维护多个默克尔帕特里夏树根哈希值，在这种场景下需要更加复杂的分布式设计。

（2）在 IoTAC-ZK 中，使用默克尔证明完成访问者实体身份的存在性检查，但是构建默克尔证明的非交互零知识证明功能是一个非常耗时的过程，另外，资源拥有者每次更新访问控制策略后，都需要重新部署该策略对应的验证合约，希望在未来能够找到一种更好的设计思路。

# 参考文献


[1] Gilder G. Microcosm: the quantum revolution in economics and technology[M]. Simon and Schuster, 1990.

[2] Statista. Internet of Things (IoT) Connected Devices Installed base Worldwide from 2015 to 2025[EB/OL]. 2016[2022-03-19]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[3] Roman R, Najera P, Lopez J. Securing the internet of things[J]. Computer, 2011, 44(9): 51-58.

[4] Alaba F A, Othman M, Hashem I A T, et al. Internet of Things security: A survey[J]. Journal of Network and Computer Applications, 2017, 88: 10-28.

[5] Xu K, Qu Y, Yang K. A tutorial on the internet of things: from a heterogeneous network integration perspective[J]. IEEE network, 2016, 30(2): 102-108.

[6] Delgado-Mohatar O, Tolosana R, Fierrez J, et al. Blockchain in the internet of things: Architectures and implementation[C]. 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC). IEEE, 2020: 1072-1077.

[7] Ali M S, Dolui K, Antonelli F. IoT data privacy via blockchains and IPFS[C]. Proceedings of the seventh international conference on the internet of things. 2017: 1-7.

[8] Ouaddah A, Mousannif H, Abou Elkalam A, et al. Access control in the Internet of Things: Big challenges and new opportunities[J]. Computer Networks, 2017, 112: 237-262.

[9] Ouaddah A, Elkalam A A, Ouahman A A. Towards a novel privacy-preserving access control model based on blockchain technology in IoT[M]. Europe and MENA cooperation advances in information and communication technologies. Springer, Cham, 2017: 523-533.

[10] Pavithran D, Shaalan K, Al-Karaki J N, et al. Towards building a blockchain framework for IoT[J]. Cluster Computing, 2020, 23(3): 2089-2103.

[11] Chen T, Lu A, Kunpittaya J, et al. A Review of Zero Knowledge Proofs[DB/OL]. (2022-01-25)[2022-04-07]. https://timroughgarden.github.io/fob21/reports/r4.pdf.

[12] Pinno O J A, Gregio A R A, De Bona L C E. Controlchain: Blockchain as a central enabler for access control authorizations in the iot[C]. GLOBECOM 2017-2017 IEEE Global Communications Conference. IEEE, 2017: 1-6.

[13] Abd El-Aziz A A, Kannan A. A comprehensive presentation to XACML[C]. Third International Conference on Computational Intelligence and Information Technology (CIIT 2013), 2013: 155-161.

[14] Hu V C, Ferraiolo D, Kuhn R, et al. Guide to attribute based access control (abac) definition and considerations (draft)[J]. NIST special publication, 2013, 800(162): 1-54.

[15] Jones M, Hardt D. The oauth 2.0 authorization framework: Bearer token usage[R]. RFC 6750, October, 2012.

[16] Siriwardena P. User Managed Access (UMA)[M]. Advanced API Security. Apress, Berkeley, CA, 2014: 155-170.

[17] Dewni Weeraman. User Managed Access — UMA 2.0[DB/OL]. (2017-08-06)[2022-05-15]. https://dewni-matheesha.medium.com/user-managed-access-uma-2-0-bcecb1d535b3.

[18] Sun S, Du R, Chen S, et al. Blockchain-Based IoT Access Control System: Towards Security, Lightweight, and Cross-Domain[J]. IEEE Access, 2021, 9: 36868-36878.

[19] Yutaka M, Zhang Y, Sasabe M, et al. Using ethereum blockchain for distributed attribute-based access control in the internet of things[C]. 2019 IEEE Global Communications Conference (GLOBECOM). IEEE, 2019: 1-6.

[20] Shurman M, Obeidat A A R, Al-Shurman S A D. Blockchain and smart contract for IoT[C]. 2020 11th International Conference on Information and Communication Systems (ICICS). IEEE, 2020: 361-366.

[21] Lu Y. The blockchain: State-of-the-art and research challenges[J]. Journal of Industrial Information Integration, 2019, 15: 80-90.

[22] Wilson D, Ateniese G. From pretty good to great: Enhancing PGP using bitcoin and the blockchain[C]. International conference on network and system security. Springer, Cham, 2015: 368-375.

[23] Song L, Ju X, Zhu Z, et al. An access control model for the Internet of Things based on zero-knowledge token and blockchain[J]. EURASIP Journal on Wireless Communications and Networking, 2021, 2021(1): 1-20.

[24] Groth J. On the size of pairing-based non-interactive arguments[C]. Annual international conference on the theory and applications of cryptographic techniques. Springer, Berlin, Heidelberg, 2016: 305-326.

[25] Pop C D, Antal M, Cioara T, et al. Blockchain and demand response: Zero-knowledge proofs for energy transactions privacy[J]. Sensors, 2020, 20(19): 5678.

[26] Nord J H, Koohang A, Paliszkiewicz J. The Internet of Things: Review and theoretical framework[J]. Expert Systems with Applications, 2019, 133: 97-108.

[27] Chahid Y, Benabdellah M, Azizi A. Internet of things security[C]. 2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS). IEEE, 2017: 1-6.

[28] (印度) 拉杰·卡马尔(Raj Kamal)著. 物联网导论[M]. 机械工业出版社, 2019.

[29] Wikipedia. IEEE 802.15.4[DB/OL]. (2022-04-26)[2022-05-16]. https://en.wikipedia.org/wiki/IEEE_802.15.4.

[30] Lampson B W. Protection[J]. ACM SIGOPS Operating Systems Review, 1974, 8(1): 18-24.

[31] Wikipedia. Computer access control[DB/OL]. (2021-10-18)[2022-03-14]. https://en.wikipedia.org/wiki/Computer_access_control.

[32] Smith R E. A contemporary look at Saltzer and Schroeder's 1975 design principles[J]. IEEE Security & Privacy, 2012, 10(6): 20-25.

[33] Gusmeroli S, Piccione S, Rotondi D. A capability-based security approach to manage access control in the internet of things[J]. Mathematical and Computer Modelling, 2013, 58(5-6): 1189-1205.

[34] Wikipedia. Access-control list[DB/OL]. (2022-03-16)[2022-03-19]. https://en.wikipedia.org/wiki/Access-control_list.

[35] Qiu L, Zhang Y, Wang F, et al. Trusted computer system evaluation criteria[C]. National Computer Security Center. 1985.

[36] Loscocco P A, Smalley S D, Muckelbauer P A, et al. The inevitability of failure: The flawed assumption of security in modern computing environments[C]. In Proceedings of the 21st national information systems security conference. 1998.

[37] Ferraiolo D, Kuhn D R, Chandramouli R. Role-based access control[M]. Artech house, 2003.

[38] Sandhu R S, Coyne E J, Feinstein H L, et al. Role-based access control models[J]. Computer, 1996, 29(2): 38-47.

[39] Jin X, Krishnan R, Sandhu R. A unified attribute-based access control model covering DAC, MAC and RBAC[C]. IFIP Annual Conference on Data and Applications Security and Privacy. Springer, Berlin, Heidelberg, 2012: 41-55.

[40] Hu V C, Ferraiolo D, Kuhn R, et al. Guide to attribute based access control (abac) definition and considerations (draft)[J]. NIST special publication, 2013, 800(162): 1-54.

[41] Ferraiolo D, Chandramouli R, Kuhn R, et al. Extensible access control markup language (XACML) and next generation access control (NGAC)[C]. Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control. 2016: 13-24.

[42] Vollbrecht J, Calhoun P, Farrell S, et al. AAA authorization framework[R]. rfc 2904, August, 2000.

[43] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system[J]. Decentralized Business

Review, 2008: 21260.

[44] Merkle R C. A digital signature based on a conventional encryption function[C]. Conf erence on the theory and application of cryptographic techniques. Springer, Berlin, Heid elberg, 1987: 369-378.

[45] Becker G. Merkle signature schemes, merkle trees and their cryptanalysis[J]. Ruhr-Uni versity Bochum, Tech. Rep, 2008, 12: 19.

[46] Wood G. Ethereum: A secure decentralised generalised transaction ledger[J]. Ethereum project yellow paper, 2014, 151(2014): 1-32.

[47] Knizhnik K. Patricia tries: A better index for prefix searches[J]. Dr. Dobb's Journal, 2 008.

[48] Buterin V. Dagger: A Memory-Hard to Compute, Memory-Easy to Verify Scrypt Alter native. 2013b[J]. URL {http://vitalik.ca/ethereum/dagger.html}, 2017.

[49] Pritzker P, Gallagher P D. Sha-3 standard: permutation-based hash and extendable-outp ut functions[J]. Information Tech Laboratory National Institute of Standards and Techno logy, 2014: 1-35.

[50] Wikipedia. Flooding (computer networking)[DB/OL]. (2021-02-14)[2022-03-14]. https://e n.wikipedia.org/wiki/Flooding_(computer_networking).

[51] Smith C. ETHEREUM ACCOUNTS[DB/OL]. (2022-01-04)[2022-03-19]. https://ethereu m.org/en/developers/docs/accounts/

[52] Antonopoulos A M, Wood G. Mastering ethereum: building smart contracts and dapps [M]. O'reilly Media, 2018.

[53] Johnson D, Menezes A, Vanstone S. The elliptic curve digital signature algorithm (EC DSA)[J]. International journal of information security, 2001, 1(1): 36-63.

[54] Pote S, Sule V, Lande B K. Arithmetic of Koblitz Curve Secp256k1 Used in Bitcoin Cryptocurrency Based on One Variable Polynomial Division[C]. 2nd International Confe rence on Advances in Science & Technology (ICAST). 2019.

[55] eth.wiki. RLP[DB/OL]. (2020-11-06)[2022-03-14]. https://eth.wiki/fundamentals/rlp.

[56] Szabo N. Formalizing and securing relationships on public networks[J]. First monday, 1997.

[57] Sipser M. Introduction to the Theory of Computation[J]. ACM Sigact News, 1996, 27 (1): 27-29.

[58] Paul Wackerow. ETHEREUM VIRTUAL MACHINE (EVM)[DB/OL]. (2022-01-19)[202 2-03-14]. https://ethereum.org/en/developers/docs/evm/.

[59] Bernays P. Alonzo Church. An unsolvable problem of elementary number theory. Ame

rican journal of mathematics, vol. 58 (1936), pp. 345–363[J]. The Journal of Symbolic Logic, 1936, 1(2): 73-74.

[60] Rick Park. Solidity: Is there any way to calculate the elapsed time for smart contract? [EB/OL]. (2019-11-10)[2022-03-17]. https://ethereum.stackexchange.com/questions/77358/solidity-is-there-any-way-to-calculate-the-elapsed-time-for-smart-contract.

[61] Ethereum. Solidity[DB/OL]. (2022-03-07)[2022-03-17]. https://docs.soliditylang.org/en/v0.8.13/.

[62] ChainSafe. Ethereum JavaScript API[DB/OL]. (2020-07-24)[2022-03-17]. https://web3js.readthedocs.io/en/v1.2.11/index.html.

[63] Goldwasser S, Micali S, Rackoff C. The knowledge complexity of interactive proof systems[J]. SIAM Journal on computing, 1989, 18(1): 186-208.

[64] Blum M, Feldman P, Micali S. Non-interactive zero-knowledge and its applications[M]. Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali. 2019: 329-349.

[65] Blum M, De Santis A, Micali S, et al. Noninteractive zero-knowledge[J]. SIAM Journal on Computing, 1991, 20(6): 1084-1118.

[66] Wikipedia. Common reference string model[DB/OL]. (2019-01-30)[2022-03-17]. https://en.wikipedia.org/wiki/Common_reference_string_model.

[67] Bitansky N, Canetti R, Chiesa A, et al. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again[C]. Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. 2012: 326-349.

[68] Arora S, Safra S. Probabilistic checking of proofs: A new characterization of NP[J]. Journal of the ACM (JACM), 1998, 45(1): 70-122.

[69] Wikipedia. Probabilistically checkable proof [DB/OL]. (2022-03-18)[2022-05-17]. https://en.wikipedia.org/wiki/Probabilistically_checkable_proof.

[70] Sasson E B, Chiesa A, Garman C, et al. Zerocash: Decentralized anonymous payments from bitcoin[C]. 2014 IEEE Symposium on Security and Privacy. IEEE, 2014: 459-474.

[71] 苏瑞国,阳建,秦继伟,武晓雄,贾振红.基于物联网区块链的轻量级共识算法研究[J].计算机工程:1-11[2022-04-06].DOI:10.19678/j.issn.1000-3428.0063845.

[72] Conoscenti M, Vetro A, De Martin J C. Blockchain for the Internet of Things: A systematic literature review[C]. 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA). IEEE, 2016: 1-6.

[73] Delgado-Mohatar O, Fierrez J, Tolosana R, et al. Biometric template storage with bloc

kchain: A first Look into cost and performance tradeoffs[C]. Proceedings of the IEEE/ CVF Conference on Computer Vision and Pattern Recognition Workshops. 2019: 0-0.

[74] Delgado-Mohatar O, Tolosana R, Fierrez J, et al. Blockchain in the Internet of Thing s: Architectures and Implementation[C]. 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC). IEEE, 2020: 1072-1077.

[75] Buterin V. Why sharding is great: demystifying the technical properties[DB/OL]. (2021 -04-07)[2022-04-06]. https://vitalik.ca/general/2021/04/07/sharding.html.

[76] Zyskind G, Nathan O, Pentland A. Enigma: Decentralized computation platform with guaranteed privacy[J]. arXiv preprint arXiv:1506.03471, 2015.

[77] Eberhardt J, Heiss J. Off-chaining models and approaches to off-chain computations[C]. Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distribut ed Ledgers. 2018: 7-12.

[78] Schaar P. Privacy by design[J]. Identity in the Information Society, 2010, 3(2): 267-27 4.

[79] Reitwiessner C. zkSNARKs in a nutshell[J]. Ethereum blog, 2016, 6: 1-15.

[80] Steven Yue. 浅谈零知识证明之三：zkSNARK 证明体系的实现[DB/OL]. (2020-03-15)[2 022-03-19]. http://blog.higashi.tech/2020/03/15/zkpub_03.html.

[81] Ben-Sasson E, Chiesa A, Tromer E, et al. Succinct {Non-Interactive} Zero Knowledge for a von Neumann Architecture[C]. 23rd USENIX Security Symposium (USENIX Se curity 14). 2014: 781-796.

[82] Parno B, Howell J, Gentry C, et al. Pinocchio: Nearly practical verifiable computation [C]. 2013 IEEE Symposium on Security and Privacy. IEEE, 2013: 238-252.

[83] Ben-Sasson E, Chiesa A, Genkin D, et al. SNARKs for C: Verifying program executi ons succinctly and in zero knowledge[C]. Annual cryptology conference. Springer, Berli n, Heidelberg, 2013: 90-108.

[84] Vitalik Buterin. Quadratic Arithmetic Programs: from Zero to Hero[DB/OL]. (2016-12-12)[2022-03-18] https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-z ero-to-hero-f6d558cea649.

[85] Gennaro R, Gentry C, Parno B, et al. Quadratic span programs and succinct NIZKs without PCPs[C]. Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2013: 626-645.

[86] Meffert D. Bilinear pairings in cryptography[J]. Radboud Universiteit Nijmegen, Comp uting Science Department, the Netherlands, 2009: 22-82.

[87] Koblitz N, Menezes A. Pairing-based cryptography at high security levels[C]. IMA Int

ernational Conference on Cryptography and Coding. Springer, Berlin, Heidelberg, 2005: 13-36.

[88] Hoffstein J, Pipher J, Silverman J H, et al. An introduction to mathematical cryptography[M]. New York: springer, 2008.

[89] Bloom B H. Space/time trade-offs in hash coding with allowable errors[J]. Communications of the ACM, 1970, 13(7): 422-426.

[90] Eberhardt J, Tai S. Zokrates-scalable privacy-preserving off-chain computations[C]. IEEE International Conference on Internet of Things, 2018: 1084-1091.

# 作者简历

## 一、基本情况

姓名：张驰 性别：男 民族：汉 出生年月：1996-08-30 籍贯：山东省莱芜市

2015-09—2019-07 中国矿业大学计算机科学与技术学院学士；

2019-09—2022-06 中国矿业大学计算机科学与技术学院学士攻读硕士学位

## 二、学术论文

1. Yang X, Yu X, Zhang C, et al. MineGPS: Battery-Free Localization Base Station for Coal Mine Environment[J]. IEEE Communications Letters, 2021, 25(8): 2579-2583.

## 三、获奖情况

1. 2019-2020 学年 硕士二等学业奖学金
2. 2020-2021 学年 硕士二等学业奖学金
3. 2021-2022 学年 硕士二等学业奖学金

## 四、研究项目

1. 横向课题：深井提升系统机电液一体化智能监控系统
2. 横向课题：暗井提升机全状态远程协同监控系统与应用

# 学位论文原创性声明

本人郑重声明：所呈交的学位论文《基于区块链与零知识证明的物联网访问控制研究》，是本人在导师指导下，在中国矿业大学攻读学位期间进行的研究工作所取得的成果。据我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

年　月　日

# 学位论文数据集

| 关键词* | 密级* | 中图分类号* | UDC | 论文资助 |
|---|---|---|---|---|
| | | | | |

| 学位授予单位名称* | 学位授予单位代码* | 学位类别* | 学位级别* |
|---|---|---|---|
| 中国矿业大学 | 10290 | | |

| 论文题名* | 并列题名* | 论文语种* |
|---|---|---|
| | | |

| 作者姓名* | | 学号* | |
|---|---|---|---|
| 培养单位名称* | 培养单位代码* | 培养单位地址 | 邮编 |
| 中国矿业大学 | 10290 | 江苏省徐州市 | 221116 |
| 学科专业* | 研究方向* | 学制* | 学位授予年* |
| | | | |

| 论文提交日期* | |
|---|---|

| 导师姓名* | | 职称* | |
|---|---|---|---|

| 评阅人 | 答辩委员会主席* | 答辩委员会成员 |
|---|---|---|
| | | |

电子版论文提交格式　文本（　）　图像（　）　视频（　）　音频（　）　多媒体（　）
其他（　）

推荐格式：application/msword; application/pdf

| 电子版论文出版（发布）者 | 电子版论文出版（发布）地 | 权限声明 |
|---|---|---|
| | | |

| 论文总页数* | |
|---|---|

注：共 33 项，其中带*为必填数据，共 22 项。