软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(1):34−61 [doi: 10.13328/j.cnki.jos.007446] [CSTR: 32375.14.jos.007446]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

智能合约可升级技术综述*

郭 涛[1,3], 尚凤军[1], 刘曼宇[4], 刘期烈[2,3]

[1](重庆邮电大学 计算机科学与技术学院, 重庆 400065)

[2](重庆邮电大学 通信与信息工程学院, 重庆 400065)

[3](重庆邮电大学 大数据智能计算重点实验室, 重庆 400065)

[4](河北工程大学 信息与电气工程学院, 河北 邯郸 056009)

通信作者: 刘期烈, E-mail: liuql@cqupt.edu.cn

摘 要: 智能合约作为一种自动执行的计算机交易协议, 广泛用于实现区块链网络中的各种业务逻辑. 然而区块链严格不变性使得智能合约维护存在困扰, 关于智能合约的可升级性讨论成为热点研究问题. 致力于以可升级智能合约为研究对象, 系统性地梳理可升级智能合约的国内外发展现状, 并介绍 7 种主流的可升级智能合约模式. 将从可升级智能合约、应用需求、升级框架与安全监管这 4 个角度进行总结, 涵盖可升级智能合约的设计、实现、测试、部署及运维多个阶段, 总结可升级智能合约的研究进展与未来挑战, 以期为区块链应用发展提供参考.

关键词: 区块链; 智能合约; 可升级技术; 升级框架; 安全监管

中图法分类号: TP311

中文引用格式: 郭涛, 尚凤军, 刘曼宇, 刘期烈. 智能合约可升级技术综述. 软件学报, 2026, 37(1): 34–61. http://www.jos.org.cn/1000-9825/7446.htm

英文引用格式: Guo T, Shang FJ, Liu MY, Liu QL. Overview on Upgradeable Smart Contract Technologies. Ruan Jian Xue Bao/ Journal of Software, 2026, 37(1): 34–61 (in Chinese). http://www.jos.org.cn/1000-9825/7446.htm

Overview on Upgradeable Smart Contract Technologies

GUO Tao[1,3], SHANG Feng-Jun[1], LIU Man-Yu[4], LIU Qi-Lie[2,3]

[1](School of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China)

[2](School of Communications and Information Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China)

[3](Key Laboratory of Big Data Intelligent Computing, Chongqing University of Posts and Telecommunications, Chongqing 400065, China)

[4](School of Information and Electrical Engineering, Hebei University of Engineering, Handan 056009, China)

Abstract: Smart contracts, as automatically executed computer transaction protocols, are widely applied in blockchain networks to implement various types of business logic. However, the strict immutability of blockchain poses significant challenges for smart contract maintenance, making upgradeability a prominent research topic. This study focuses on upgradeable smart contracts, systematically reviewing their development status both domestically and internationally, and introducing seven mainstream upgradeable contract models. The research is summarized from four key perspectives: upgradeable smart contracts, application requirements, upgrade frameworks, and security oversight. It covers multiple stages, including design, implementation, testing, deployment, and maintenance. The goal is to provide insights and references for the further development of blockchain applications.

Key words: blockchain; smart contract; upgradeable technology; upgrade framework; security oversight

---

* 基金项目: 国家留学基金委员会留学基金 (202407840001); 重庆市自然科学基金创新发展联合基金 (CSTB2024NSCQ-LZX0134); 重庆邮电大学博士研究生创新人才项目 (BYJS202312)
收稿时间: 2024-10-31; 修改时间: 2025-01-11, 2025-02-28; 采用时间: 2025-04-11; jos 在线出版时间: 2025-08-20
CNKI 网络首发时间: 2025-08-20

智能合约的发展与区块链技术的演进密切相关. 1994 年, Szabo[1] 首次提出智能合约的概念: “智能合约是一种执行合约条款的计算机化交易协议.” 1996 年, Grigg[2]提出李嘉图合约, 更明确地定义由计算机编写的智能合约的意图和用途. 2008 年, Nakamoto[3]提出比特币的设计构想, 标志着区块链 1.0 时代的开始, 使得智能合约有望在安全且去中心化的环境中有效部署. 2013 年, Buterin[4]提出一个用户可自由创造实例的区块链——以太坊, 该项目在 2015 年上线, 是第 1 个支持可编程的智能合约平台. 以太坊虚拟机实现代表区块链 2.0 时代, 推动去中心化应用的发展. 智能合约作为去中心化应用的基础模块, 与传统程序相比, 具有显著的去中心化特性[5]: 能够在区块链上自动执行合约条款, 不依赖任何中介机构或第三方服务, 执行过程和结果对所有人公开透明, 其代码和交易记录在区块链上任何人都可以审查. 有效防止数据篡改和欺诈行为, 确保合约内容和执行过程的完整性与可信度[6].

2016–2018 年间, 多个开源区块链平台如 EOS[7]、NEO[8]、Fabric[9]、Libra[10]、Zilliqa[11]的陆续诞生, 标志着区块链技术进入 3.0 时代[12]. 区块链 3.0 时代强调可扩展性、隐私性和效率, 以支持更复杂的交易协议和多样化的并行应用. 在金融领域, Hyperledger Fabric、Corda 和 Tron 等平台以支持企业级智能合约自主开发[13], 用作市场预测. 在政府部门中也可用于自治管理[14], 如电子政务 eGov-DAO、数字版权管理、智能交通等应用. 可升级智能合约是未来智能合约发展的新趋势, 它使得合约在部署后仍具备动态升级的能力, 从而提高其可维护性和适应性. 2022 年, Compound Finance[15]和 Open Sea[16]项目中, 通过可升级智能合约技术实现功能的灵活升级.

尽管智能合约具有诸多好处, 但为修复漏洞和进行潜在的功能改进, 升级合约代码仍然是必要的[17]. 由于上市时间的压力、发布的紧迫性、开发工具的不成熟[18]、高质量的培训资源的缺乏, 以及以太坊路线图的不明确和生态系统的快速发展, 合约开发者难免会在智能合约中引入一定程度的技术债务[19], 例如代码中的漏洞、临时性解决方法和非最优实现. 此外, 区块链的严格不变性也会造成智能合约功能的固化[20], 这意味着后续业务功能扩展面临挑战. 在过去的几年里, 已经发生几起备受瞩目的智能合约攻击事件. 以 2016 年 DAO 黑客攻击为例. 攻击者利用重入漏洞窃取价值超过 5 000 万美元的以太币[21]. 此类安全事件的主要攻击目标通常是持有大量以太币的智能合约, 考虑到人为失误、技术改进和新发现的漏洞, 这些智能合约一直存在升级的需求.

对于以太坊开发人员来说, 智能合约升级并不是一个全新的概念. 升级既可以允许在不改变合约地址的情况下迅速修复安全漏洞, 防止潜在攻击和资金损失, 也可以通过逐步添加新功能实现系统持续迭代. 最早关于合约升级的讨论可以追溯到 2016 年 Johnson[22]在 GitHub 上的提案, 自此合约升级便成为以太坊发展历程中的重要话题: 2017 年开始探索代理合约模式以解决智能合约的不可变性问题[23]. 2018 年, OpenZeppelin 推出代理合约库, 为可升级智能合约提供标准化工具[24]. 2019 年, EIP-1822 提案进一步标准化代理合约机制[25]. 2020–2022 年[26], 随着以太坊 2.0 和 Layer2 网络扩展方案的推广[27], 可升级智能合约的性能和灵活性得到显著提升. 2023 年, OpenZeppelin 发布 OpenZeppelin Contracts 5.0[28], 添加对 Foundry 的升级支持, 并集成了 Defender 2.0, 发布了升级插件和合约指南的更新, 使得可升级智能合约的开发和管理更加便捷和安全. 2024 年, 基于可信执行环境的完全可编辑智能合约升级平台得到关注[29]. 目前, 智能合约可升级方法主要在非学术的技术博客中讨论[30−39]. 例如, 以太坊 Stack Exchange 中包含近 10 万个有关于以太坊智能合约可升级性的讨论帖子, 其中很多开发者都有升级合约的需求. 主要原因可以归纳为: 功能扩展、性能优化和漏洞修复[40]等.

基于区块链的可升级智能合约技术在商业化创新驱动下取得较大进展, 但学术研究方面相对滞后, 截至 2024 年 7 月, 以知网为中文数据源, Web of Science、IEEE Xplore、DBLP、Google Scholar 等英文数据源, 如表 1 检索主题包含“可升级智能合约/Upgradable Smart Contracts”相关的学术论文仅有中文 2 篇和英文 40 篇. 其中体现应用升级需求类相关的论文占比最高, 达到 44% (26/59), 其中 16 篇论文发表在高水平学术期刊; 其次是智能合约升级规范相关的论文占比 37% (22/59), 其中 13 篇论文发表在高水平学术期刊; 介绍相关可升级智能合约的论文较少, 占比 19% (11/59), 但 6 篇论文发表在高水平学术期刊.

根据统计, 目前关于可升级智能合约的现实需求主要集中在以下 3 个方面.

1) 介绍类说明: 大多为综述类介绍智能合约的升级方式, 主要工作涉及代理模式的大规模探索性研究[41].

2) 需求类分析: 分析可升级合约时调用优化、部署或后期的维护问题, 主要工作围绕需求分析和框架应用展开.

3) 规范类讨论: 关注智能合约升级的安全性和规范性问题, 提出合约升级的安全开发流程.

表 1 主要文献来源

领域	CCF/SCI等级	发文数量	发表源简称
会议	A类	4	USENIX Security、IWWWC、ASE
	B类	4	DSN、LCTES、ICPC、SNAER
	C类	1	FC 2022
期刊	Q1	18	Empir. Softw. Eng.、Inform. Software Tech.、IEEE Trans. Softw. Eng.、ACM Trans. Softw. Eng. Methodol.、Inform. Syst. Front.、Int. J. Prod. Res.、J. Netw. Comput. Appl.、IEEE Internet Things J.、IEEE Commun. Mag.、Autom. Constr.、J. Cleaner Prod.、Blockchain: Research and Applications
	Q2	1	Electronics
	Q3	5	Software and Systems Modeling、Information、Frontiers in Blockchain、The Journal of The British Blockchain Association
	Q4	2	KSII Transactions on Internet and Information Systems、Secur. Commun. Netw.

可升级智能合约需求类和规范类的讨论占据主导地位. 这表明大多数开发人员仍然停留在应用与设计阶段, 对可升级智能合约的理解尚不深入. 图 1 展示与本文研究主题相关的文献发表时间信息, 而可升级智能合约的研究主要集中在欧美国家, 国内处于起步阶段. 造成这种分布不均的主要原因包括以下几点.

1) 对合约缺陷的前期规避更受重视, 然而这往往会忽视智能合约后期的可维护性.

2) 智能合约在国内应用起步较晚, 健全的代码规范和开发经验需要时间积累. 所以目前关于可升级智能合约的论文数量还相对较少.

图 2 进一步总结过去 4 年中相关文献综述[18,40−47]对于可升级智能合约的使用情况. 如图 2(a) 所示, 目前智能合约的主要类型包括但不限于非同质化代币 (NFT) 市场合约、治理合约、去中心化金融 (DeFi) 合约、代理合约和代币合约. 其中代币合约占据 40% 左右, 而代理合约占据 20%[41].

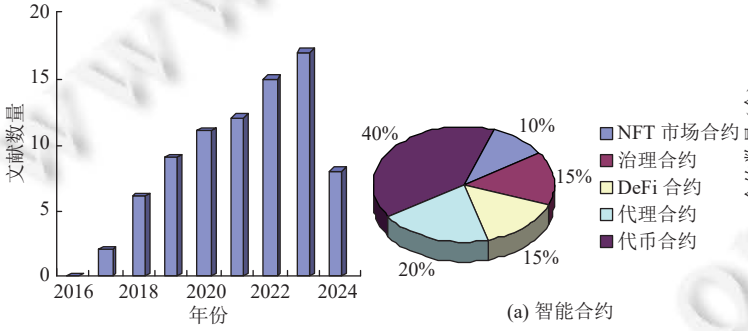

图 1 可升级智能合约时空研究分布

图 2 可升级智能合约各类分析

此外, 如图 2(b) 所示收集并展示可升级智能合约的发展趋势. 从 2019 年起, 透明代理是当时最流行的可升级方案, 市场占有率约为 60%, 此时钻石合约提案尚未发布, 几乎没有应用[42]. 2020 年, 受 DeFi 项目的兴起推动透明代理使用量的显著增加[43]. 同时, 最小代理和通用可升级代理标准 (UUPS) 模式开始受到开发者关注. 2021 年, 透明代理仍是主流升级方案, NFT 和区块链游戏需求推动最小代理的增长, 而 UUPS 也因存储效率较高得到广泛使用, 钻石合约和信标代理逐渐在大规模复杂部署项目中使用[44]. 2022 年, 透明代理增长趋缓, 主要在需要稳定性的项目中继续使用. 最小代理被大量应用于 DApp 和 NFT 市场, UUPS 模式逐渐成为热门选择之一, 钻石在模块化需求高的项目中应用增加, 信标代理则因其灵活性在 NFT 和 DeFi 项目中被频繁采用[41]. 2023 年, 透明代理被更高效的 UUPS 模式逐步取代, 数量增速放缓[45], UUPS 广泛用于 DeFi 和 NFT 项目, 最小代理随着轻量化应用的进一步发展. 区块链游戏和中心化自治组织 (DAO) 项目对钻石合约的需求持续增长, 而信标代理由于跨链项目需求也逐渐增加. 2024 年, 透明代理的应用量稳定在 19 000–21 000 之间, 仍用于对升级稳定性要求较高的传统项目[47].

UUPS 模式进一步扩展到新的区块链和协议中, 其应用量已超过透明代理. 最小代理广泛应用于需要批量实例的轻量级合约中, 而钻石合约和信标代理在复杂协议中的应用逐步成熟.

上述活动反映出智能合约设计模式的演变和社区对灵活性需求的日益增长. 这一显著增长不仅突显 UUPS 模式在智能合约开发中的广泛普及, 也展示开发者对于能够灵活升级和维护智能合约的强烈需求.

如表 2 所示提供一些可升级智能合约的相关数据集.

表 2　可升级智能合约分析的相关数据集

文献	类型	地址	备注
[18]	官方文档	https://ethereum.org/en/developers/docs/smart- contracts/upgrading/	以太坊社区引入的升级模式
[36]	开源仓库	https://github.com/tintinweb/smart-contract-sanctuary	包含各种网络的智能合约源
[41]	开源仓库	https://github.com/USCHunt-Anon/USCHunt	原型工具、数据集和分析结果
[43]	官方文档	https://etherscan.io/contractsVerified	过经验证的智能合约的源代码
[43]	开源仓库	https://github.com/safe-global/safe-contracts	安全合约升级部署及其地址的集合
[45]	开源仓库	https://www.selectdataset.com/dataset/547f3b7b06015118aec0d672a2ca74b4	智能合约升级版本谱系数据集
[45]	源代码	https://cloud.google.com/blog/products/data-analytics/ethereum-bigquery-publicdataset-smart-contract-analytics	BigQuery中公开的以太坊数据集
[45]	官方文档	https://info.etherscan.com/what-is-proxy-contract/	代理合约实现相关的可识别的字节码模式

本文致力于按时间顺序梳理和讨论区块链发展过程中的智能合约可升级方法, 针对相关文献梳理, 从可升级智能合约的模型构造、优化方法、升级框架与安全监督这 4 个角度进行总结, 涵盖可升级智能合约的设计、实现、测试、部署及运维多个阶段. 引言部分概述可升级智能合约的研究进展和分布. 第 1 节整理可升级智能合约的概念模型和基本工作流程. 第 2 节介绍相关优化方法、升级需求. 第 3 节介绍相关设计框架、工具与应用工具. 第 4 节讨论升级规范和安全监管问题. 第 5 节对本文内容进行总结, 提出未来研究方向.

1　可升级智能合约

可升级智能合约与可编辑区块链 (redactable blockchain)[48]有本质区别. 智能合约本身是不可变的, 这使得在合约部署后无法直接对其业务逻辑进行更新. 为解决业务逻辑地址更新的问题, 最初引入“合约迁移 (contract migration)”机制.

合约迁移是指在区块链系统 (如以太坊) 中将已部署的智能合约从一个地址或版本迁移到另一个地址或版本的过程, 通常涉及旧合约的停用和新合约的部署. 为了确保数据和状态的平滑过渡, 在进行迁移之前, 需要明确原合约的优化需求或漏洞, 是否有必要升级合约的数据结构 (如更改存储方式). 例如在 2015–2016 年, 最早的合约迁移方式是数据迁移 (data migration) 模式. 如图 3 所示, 数据迁移模式主要包括以下两个步骤.

(1) 数据恢复: 从迁移源区块链中读取相关区块数据, 并根据合约的数据结构使用相应的方法恢复数据.

(2) 数据写入: 收集完需要迁移的数据后, 在目标区块链上部署并初始化新合约.

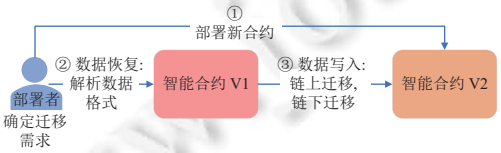

图 3　合约迁移流程

合约迁移后, 新版本的合约将被部署, 并且需要将所有状态和余额迁移至新合约实例. 数据迁移方法的优点在于其简单直接, 可以避免旧逻辑中的漏洞延续到新逻辑中. 新部署的合约还能够进行全面优化以摆脱旧合约的技术债务. 然而数据迁移会导致合约生成新的地址, 这一过程中高度依赖人工操作, 如果在迁移期间状态更新出现错误, 会破坏原有功能并引入新的安全漏洞. 此外, 新版本与旧合约之间的关联性丧失会削弱用户体验, 并降低用户

对系统的信任. 因此数据迁移方法在实践中面临较大的挑战, 通常并非首选.

考虑到新旧合约之间的关联性丧失问题, 如图 4, 一些项目采用同时运行多版本并存 (versioning)[49]的模式, 让用户选择是否升级. 多版本并存适用于长期运营、用户基础庞大的区块链项目, 它提供了更高的兼容性和安全性, 这在 2017–2018 年 DeFi 生态逐渐成熟时更常见, 特别是在去中心化交易所和借贷协议 (如 Compound、Aave) 中. 但多个版本的链上治理、身份管理和数据同步将变得复杂, 可能导致不同版本的状态不一致, 开发团队需要花费额外精力管理多个代码库的修复漏洞、发布升级. 此外, 由于用户可以自由选择合约版本, 会造成流动性分散问题. 例如 Uniswap V2 和 V3 版本由于机制不同, 一部分用户仍选择留在 V2, 导致资金分散[50].

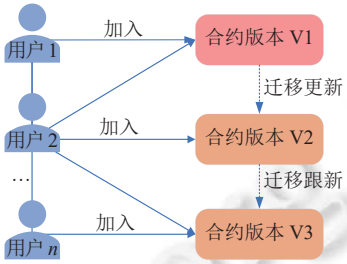

图 4 多版本并存模式

显然这需要更加全面的解决方案: 如何在不更改其地址的情况下更新其核心逻辑? 以及如何最小化操作开销? 如表 3 所示, 从这些问题中形成了代理模式 (proxy pattern).

表 3 基于代理的可升级智能合约方法小结

模式	部署成本	升级成本	调用成本	扩展性	安全性	优点	缺点
数据迁移	高	高	高	低	中	根据需求可动态调整	人工操作敏感、Gas费用昂贵和版本关联缺失
多版本并存	高	高	低	高	中	兼容性强, 历史数据保留, 支持审计与回溯分析	流动性分散, 管理难度大, 维护成本高, 数据同步问题
代理模式	低	低	中	中	低	业务逻辑和数据存储分离	delegatecall()调用漏洞、存储布局不匹配和权限管理问题
最小代理	低	低	低	低	中	多个代理合约共享相同的实现代码, 节省存储空间和Gas费用	合约后续不具备可升级性, 存在实现合约单点故障和存储冲突问题
透明代理	高	高	中	低	中	利用插槽方法解决逻辑冲突问题	实现合约Gas调用消耗、权限判定复杂问题和升级权限集中风险
变形合约	中	中	中	高	低	在不改变合约地址的情况下进行升级, 保持用户体验的连续性	地址重用风险、数据在迁移风险、用户信任问题
UUPS	低	低	低	中	中	不需要代理合约参与, 减少Gas消耗	升级逻辑漏洞和权限管理风险
钻石模式	高	低	中	高	中	模块化设计允许在多个切面以及一次交易中添加或替换多个功能	切面设计相对复杂, 权限边界定义和逻辑不一致问题
信标代理	高	中	低	高	高	通过更新一个指向最新实现合约的指针来实现升级, 消除对非结构化存储的需求	单点故障和中心化风险问题

1.1 代理模式

代理本质上是充当两个对象之间的中介[51], 而代理模式提供一种用于更新合约逻辑而不改变其地址状态的升级方法——数据分离 (data separation). 如图 5 所示, 将代理模式合约拆分为两部分.

(1) 代理合约 (proxy contract): 处理合约的数据存储和用户请求, 并将这些请求转发给逻辑合约.

(2) 逻辑合约 (logic contract): 包含具体的业务逻辑, 可随时替换以实现合约的升级.

数据分离的核心思想是通过代理合约进行逻辑调用, 将逻辑合约的地址存储在代理合约中. 当用户调用合约

的方法 (如执行 method_call() 操作) 时, 代理合约使用 delegatecall() 函数将调用转发给逻辑合约. 逻辑合约则通过操作代理合约的状态数据 (通过 getter() 和 setter() 方法) 来完成数据读取和修改. 另一方面, 代理模式引入额外的代理地址, 使得逻辑和存储合约地址之间的持续调用增加额外 Gas 消耗. 此外, 逻辑合约调用 delegatecall() 操作代理合约的存储, 如果两者存储布局不匹配, 变量错位会引发关键数据 (如地址或权限) 的覆盖问题. 一旦权限管理被攻破, 攻击者可以将逻辑合约地址更新为恶意合约地址, 利用 delegatecall() 在代理合约上下文中执行恶意代码, 窃取资金或破坏合约状态. 因此, 根据不同的需求, 衍生出最小代理、透明代理、可升级代理、钻石合约和信标代理等模型.

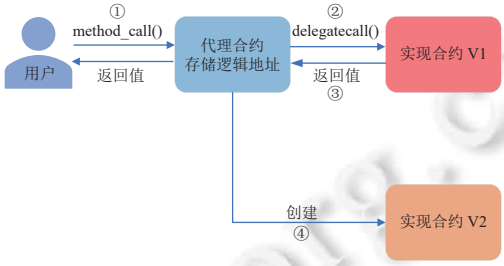

图 5 代理模型

1.2 最小代理

2018 年, EIP-1167 提案[52]规定了最小代理 (minimal proxy) 的概念, 它允许开发者创建一个代理合约, 该代理合约可以将调用转发到另一个实现合约 (implementation contract, 也称为基础合约). 图 6 展示最小代理的模型设计, 支持多个代理合约共享相同的实现代码, 但每个代理拥有独立的存储状态, 从而降低部署成本和存储需求.

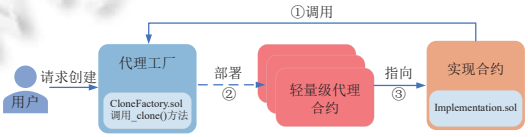

图 6 最小代理

1) 代理工厂 (proxy factory): 用户将调用工厂的一个函数, 而工厂将克隆一份实施合约的副本, 但拥有自己的存储空间. 这意味着每个克隆都有相同的逻辑, 但存储状态独立.

2) 代理 (proxy): 代理合约是执行合约的克隆, 但具有独立的存储.

3) 实现合约 (implementation contract): 有时被称为逻辑合约或执行合约, 强调该合约包含实际的业务逻辑, 但由代理合约调用.

开发者首先需要编写包含业务逻辑的实现合约 Implementation.sol, 然后将其部署到区块链上. 如图 7, 当用户编写并部署一个代理合约时, 将触发 clone() 函数, 代理工厂将输出代理的地址, 而在代理合约中设置指向实现合约的地址, 代理合约使用 delegatecall() 函数将调用转发至实现合约.

最小代理模式通过简化代理合约的结构, 显著降低部署和运行时的 Gas 成本. 代理合约本身不包含业务逻辑, 仅存储状态变量, 充当一个静态的前端接口, 它通过极简字节码部署多个合约实例, 利用 delegatecall() 函数将调用请求转发给后端的实现合约实现业务逻辑. 多个代理合约可以共享同一实现合约, 从而进一步节省部署成本.

尽管最小代理支持实现合约地址的动态更新, 但通常不包含复杂的升级机制和权限验证功能. 升级时无需修改代理合约, 仅需更新内部引用的实现合约地址. 确保用户和外部系统始终通过统一代理合约接口与智能合约交互, 从而保证合约地址的一致性与稳定性.

然而, 当多个最小代理共享实现合约时, 整个系统的复杂性显著增加. 实现合约在升级时, 新增状态变量时必须附加至存储布局的末尾, 避免调整原有变量顺序, 以防存储错位导致关键数据被覆盖或权限被意外篡改, 否则会

引发存储冲突[53], 例如管理员地址被覆盖为随机地址, 导致权限被篡改或丢失. 此外, 以太坊中函数调用是根据数据有效负载的前 4 个字节函数选择器 (函数名称及其签名的哈希值) 标识, 4 个字节长度的低碰撞性这意味着两个函数之间可能会发生冲突: 两个具有不同名称的不同函数最终可能会具有相同的选择器. 当代理和执行合约为不同的目的使用相同的存储槽时, 代理合约的调用意图因函数选择器冲突而发生偏移时, 可能意外触发实现合约中的其他函数, 导致代理合约被错误升级至随机地址. 这种情况下, 系统的安全性取决于对实现合约地址的严格管理和权限控制.

```
contract CloneFactory {
    function createClone(address target) internal returns (address result) {
        bytes20 targetBytes = bytes20(target);
        assembly {
            let clone := mload(0x40)
            mstore(
                clone,
                0x3d602d80600a3d3981f3363d3d373d3d3d363d730000000000000000000000000000
            )
            mstore(add(clone, 0x14), targetBytes)
            mstore(
                add(clone, 0x28),
                0x5af43d82803e903d91602b57fd5bf3000000000000000000000000000000000000
            )
            result := create(0, clone, 0x37)
        }
    }
}
```

图 7 clone() 函数

1.3 透明代理

2019 年, EIP-1967 提案[54]规定了透明代理 (transparent proxy), 旨在避免函数冲突, 并确保只有管理员 (admin) 才能调用升级函数. 这意味当普通用户调用代理合约时, 始终会执行实现合约中的函数, 而不会意外触发代理合约的管理函数. 并且利用代理存储槽 (proxy storage slots) 的方法, 明确规定代理存储地址的位置, 确保代理不会向最终用户暴露可能与实现合约冲突的函数. 函数调用对于用户来说是透明的. 图 8 展示透明代理的设计结构.

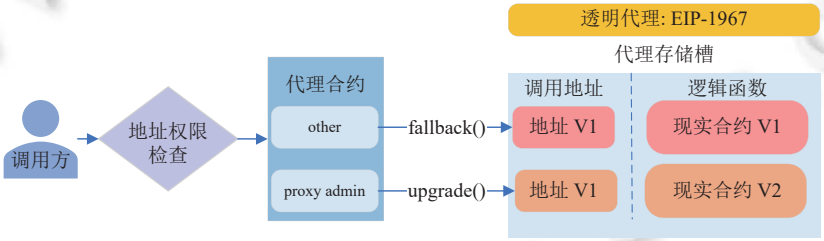

图 8 透明代理

1) 透明代理 (transparent upgradeable proxy) 主要实现 3 个核心功能.

- 构造函数: 初始化 admin 和实现合约地址.
- fallback(): 回调函数, 将调用委托给实现合约, 并禁止 admin 调用.
- upgrade(): 升级函数, 改变实现合约地址, 仅允许由 admin 调用.

2) 管理员合约 (proxy admin): 负责管理 admin 地址和升级功能.

3) EIP-1967 Proxy: 规定代理存储地址的特定位置, 并确保数据位置一致, 以防止存储冲突.

4) 实现合约 (implementation): 管理核心业务代码.

透明代理首先检查调用方的地址, 依据其权限将调用委托给底层逻辑代理. 如果调用方是 other 地址, 则代理合约只接受调用实现合约, 通过 delegatecall() 将调用转发给当前存储的实现合约地址. 实现合约在代理合约的上下文中执行, 并返回结果给代理合约. 代理合约再将结果返回给用户; 如果调用方是代理的管理员, 则代理不会接

受其他委派调用并执行管理员特定的逻辑, 如升级合约、更改权限等. 例如当存放当前实现合约的地址需要被更新时, 透明代理发出 Upgraded 事件, 以便外部代理合约监听到实现合约的变更. 计算方式为 bytes32(uint256(keccak256(“eip1967.proxy.implementation”))–1). 存储插槽也可以通过 AdminChanged 事件更新管理员地址. 计算方式为 bytes32(uint256(keccak256(“eip1967.proxy.admin”))–1).

如图 9, 为避免实现合约的版本迭代而造成的存储冲突, 代理合约和实现合约升级过程中共享相同的存储结构. 两者通过直接继承相同的变量定义来确保存储布局完全一致, 使得实现合约的版本迭代可以与之前的版本无缝兼容. 此外, 如图 10 使用恒定变量哈希的方法 (如 keccak256) 来定义存储槽的位置, 将每个关键变量存储在特定的槽中. 这样即使实现合约版本升级, 新增或调整变量也不会影响已有存储的正确性, 从而保证向后兼容性. 伪随机存储槽方法, 使新地址具有足够的随机性. 即便实现合约中的某个变量不慎占用同一槽位, 其影响也可以忽略不计, 从而进一步降低潜在冲突的风险.

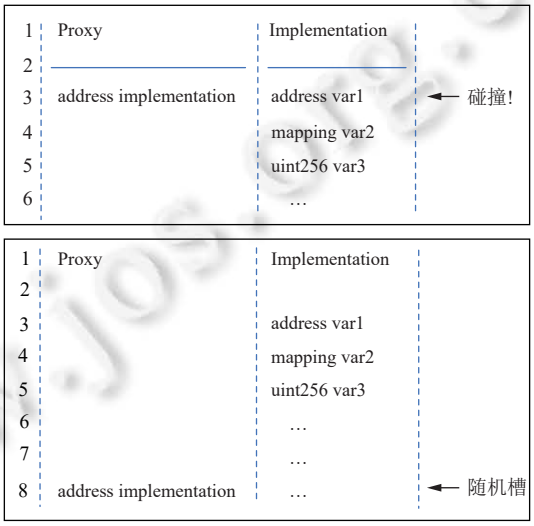

图 9　透明代理存储布局示意图

```
4   contract Robo{
5       uint public value;
6       function setValue(uint _value)public {}
7   }
8   contract RoboProxy{
9       function _delegate(address implementation)internal {}
10      fallback() external  {}
11      upgradeTo(address newImpI)external {
12          //Get the storage slot where implementation is localed and assign value
13      }
14  }
```

图 10　继承存储函数

透明代理模式在提升用户体验和升级便捷性的同时, 需要在性能和安全性之间进行权衡. 首先是 Gas 消耗过高问题, 透明代理的设计需要将升级逻辑存储在代理合约中, 这意味着调用类似 upgradeTo(address newImplementation) 的函数时, 代理必须更新实现合约地址. 由于每次用户调用均需通过代理层转发到实现合约, 代理合约承担额外的处理步骤, 从而增加 Gas 消耗, 使得透明代理的部署和运行成本高于普通合约. 其次是权限判定的复杂性, 普通用户的调用应优先转发至实现合约, 而管理员调用应直接执行代理合约的升级逻辑. 为区分这两种情况, 代理合约需要额外的逻辑判断用户权限. 此外, 由于透明代理对权限的判断依赖管理员地址的存储加载, 在伊斯坦布尔 (Istanbul) 分叉之后, 由于 Gas 计算规则的变化, 每次存储加载的成本进一步增加, 尤其是在读取管理员地址

时消耗显著, 这进一步加剧 Gas 消耗问题. 而权限管理逻辑的复杂性也增加代码实现的难度, 一旦开发不慎, 可能引发权限漏洞. 最后是升级权限集中风险, 透明代理模式的升级逻辑由管理员集中管理, 这种设计虽然便于操作, 但也引入安全风险. 如果管理员权限被泄露或遭到攻击, 攻击者可能通过恶意升级替换实现合约地址, 从而接管整个系统.

1.4 变形合约

2019 年 2 月 28 日, 以太坊从 7280 000 区块开始执行君士坦丁堡 (Constantinople) 硬分叉, 并引入新的操作码 CREATE2, 该操作码为合约部署提供一种通过指定部署参数生成固定地址的机制. 它允许在不依赖部署者地址的情况下, 通过哈希计算确定合约的部署地址. 这一特性间接支持一种创新的合约升级方式, 即变形合约 (metamorphic contract)[55], 变形合约利用 CREATE2 操作码, 在同一地址上通过销毁原有合约并重新部署新合约的方式, 实现合约逻辑的替换. 与传统升级模式相比, 变形合约无需额外的代理合约或存储迁移, 因而在存储需求较少或合约地址不变场景中更具效率. 如图 11 所示, 合约“变形”过程可通过以下 3 个步骤实现.

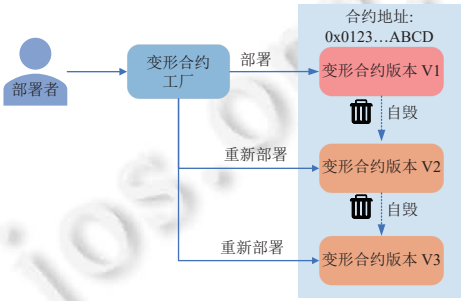

图 11 变形合约

1) 通过工厂合约部署具有自毁功能的合约.

2) 触发该合约的 SELFDESTRUCT 操作码自毁, 将原合约地址上的代码和状态清除.

3) 工厂合约结合 CREATE2 操作码, 在相同地址重新部署一个新的合约.

在这一过程中, SELFDESTRUCT 操作码的作用是销毁原合约及其状态, 清理合约地址空间. 随后, 通过 CREATE2 操作码, 使用固定的、不确定的初始化代码创建一个新合约, 确保新合约的地址在部署前即可预计算得出. 这样, 新部署的合约既能与链上尚未存在的地址交互, 又不会改变其原有地址. 与透明代理的地址管理方式不同, 这种升级方式更适用于无需持久状态或状态易于重建的合约, 例如时间敏感或一次性用途的代币合约或无需复杂余额管理的身份合约.

变形合约利用 CREATE2 操作码的确定性特性, 在销毁旧合约后可以在相同地址重新部署新实现合约. 然而, 若开发者未完全清除旧合约的数据状态, 这种地址重用会导致新逻辑与残留数据的冲突或兼容性问题. 攻击者可以利用合约销毁后出现的地址空档期, 部署恶意合约伪装成预期的实现合约, 占用预定地址. 这种“地址抢占”攻击使得重部署失败会导致功能中断, 影响应用的正常运行. 此外, 由于变形合约销毁旧合约时会丢失原有的存储状态, 部署者需要额外的存储管理机制 (例如, 外部状态存储合约) 来保留数据. 如果状态数据在迁移过程中丢失或被篡改, 则导致数据不一致或系统功能瘫痪. 变形合约的设计允许在不更改地址的情况下更换逻辑, 但这种灵活性可能降低透明度, 使得用户对系统的信任度可能下降.

1.5 通用可升级代理标准

2019 年, EIP-1822 提案[25]规定通用可升级代理标准 (universal upgradeable proxy standard, UUPS), 该标准与所有合约普遍兼容, 通过在代理合约中唯一存储位置来完成实现合约的地址存储. 实现合约本身包含升级功能, 不需要代理合约直接参与升级操作. 而是通过让它继承一个包括升级逻辑的通用标准接口, 使实现合约符合 UUPS 标准, 如继承 OpenZeppelin 的 UUPS Upgradeable 接口, 继承该接口的实现合约不仅包含业务逻辑, 还包括更新存储

在代理存储空间中特定插槽中实现地址所需的代码. 这种方式简化部署者在管理代理和实现合约方面的操作流程. 如图 12, UUPS 代理需要部署 3 个合约.

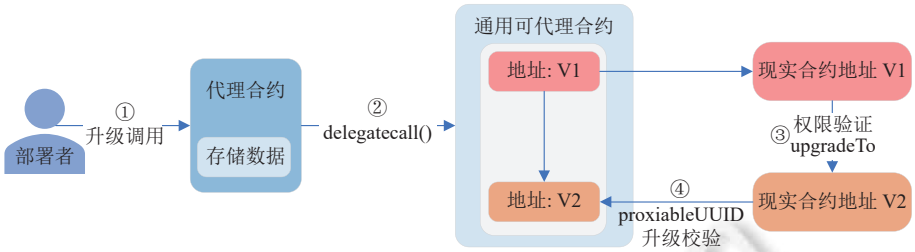

图 12 UUPS

1) 代理合约 (proxy contract): 存储数据的合约. 代理合约接收并处理部署者所有传入的升级调用, 通过 delegatecall() 将这些调用委托给实现合约.

2) 实现合约 (implementation contract): 包含代理合约使用逻辑的合约. 实现合约包含实际的业务逻辑和功能.

3) 可代理合约 (proxiable contract): 在实现合约中继承此功能以提供升级能力. 该合约提供升级所需的标准接口, 同时确保升级操作符合安全性要求. 通常称为 upgradeTo 的功能, 它允许授权实体 (例如管理员或治理机制) 升级实现合约. 此函数将代理合约的指针更新为新实现合约的地址. 每次升级实现合约时, 需要部署一个新的实现合约, 并通过 upgradeTo 函数进行切换.

首先部署代理合约, 并设置初始的实现合约地址. 可代理合约编写包含业务逻辑的实现合约, 并确保其实现 upgradeTo 和 proxiableUUID 函数. 所有对合约功能的调用都通过代理合约进行, 代理合约将调用委托给当前的实现合约. 当需要升级时, 通过调用实现合约的 upgradeTo 函数, 更新代理合约中存储的实现合约地址.

UUPS 模式通过将升级逻辑放在实现合约中而不是代理合约中, 消除对 proxy admin 合约的需要, 只需在实现合约内进行权限检查逻辑, 从而减少每次调用时的 Gas 消耗. UUPS 模式的另一个优势在于灵活性, 由于升级逻辑位于实现合约中, 因此它本身也可以升级, 提供更大的灵活性. UUPS 模式还允许通过停止继承可代理合约、删除 authorizeUpgrade 等逻辑来实现永久性锁定不可升级状态, 这是透明代理模式无法做到的.

在 UUPS 模式中, 所有的升级逻辑都集中在实现合约中, 每次升级实现合约时, 都可能改变授权升级逻辑. 这意味着如果实现合约内升级函数被意外删除或修改, 可能导致合约永久锁死. 此外, 升级权限管理不当被攻击者利用或存在漏洞, 整个系统的安全性可能会受到威胁[33]. 每次升级实现合约时, 都可能改变授权升级逻辑, 从而增加安全风险. 为避免此类问题, 开发者需要在实现合约中自定义实现 authorizeUpgrade 函数, 以严格限制升级权限. 即便如此, 用户权限管理会增加代码实现的复杂性, 并需要仔细的权限管理和审计管理以防止未经授权的升级.

1.6 钻石模式

2020 年, 以太坊的 EIP-2535[56]提案定义了一种称为钻石合约 (diamond contract) 的模块化智能合约开发架构, 以解决智能合约的可拓展性挑战. 钻石模式允许在多个切面中添加或替换功能, 同时支持在一次交易中完成多个功能的升级, 这一过程称为“钻石切割 (diamond cut)”. 如图 13 所示, 单个钻石和核心面孔可以构建一个分散的、可升级的治理框架[57].

1) 钻石合约 (diamond contract): 合约本身不包含业务逻辑, 而是通过代理模式调用其他合约的逻辑.

2) 切面 (facet): 切面包含实际业务逻辑的合约, 可以被钻石合约调用. 每个切面合约实现特定的功能, 并且可以独立升级或替换, 而不会影响其他切面合约的运行.

3) 放大镜 (loupe): 特殊的切面合约, 提供查询功能, 外部用户可以查询钻石合约所包含的所有切面合约以及其提供的函数选择器 (function selector).

4) 钻石切割 (diamond cut): 管理钻石合约功能的一组函数, 支持添加、替换或删除切面合约及其函数选择器

等功能.

5) 函数选择器 (function selector): 每个函数都由一个函数选择器标识, 这是一个 4 字节的标识符, 用于在调用时唯一标识并确定要执行的具体函数.

6) 非结构化存储 (unstructured storage): 钻石合约使用非结构化存储来保存状态变量, 非结构化存储的布局更加灵活, 但在合约升级时需要特别注意存储的兼容性, 以避免因布局不匹配而导致潜在问题.

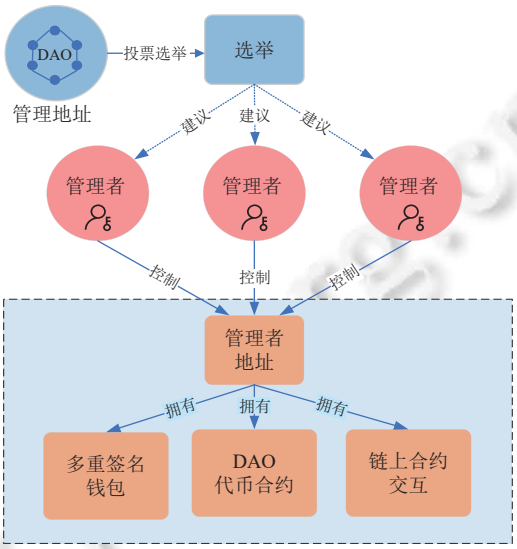

图 13 钻石合约结构

升级过程中, 基于钻石模式的 DAO 使用单一的管理地址进行智能合约治理, 该地址由 DAO 拥有, 并由社区共识进行调整. 社区成员可以通过提案和投票系统来决定是否对智能合约进行升级. 一旦提案获得通过, 就可以执行钻石切割事件, 包括添加新的切面、替换现有切面或移除不再需要的切面[54]. 虽然社区参与投票过程, 但投票结果并不具有自动执行性. 相反, 结果只是对行政行为的建议. 此外, 管理员有权修改 DAO 的财政资金 (treasury), 包括多重签名钱包 (multisig treasury) 和代币合约 (token contract), 例如包括铸造 (mint)、销毁 (burn) 或重新分配 DAO 代币, 而不需要 DAO 会员的许可.

钻石图案允许功能性的集体升级. 对 DAO 实施变更的过程包括以下 4 个步骤, 如图 14 所示.

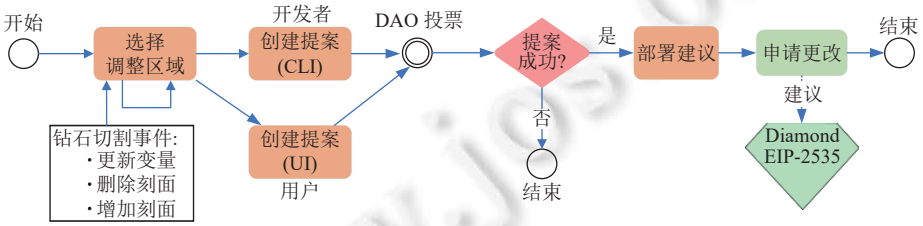

图 14 钻石升级流程图

1) 共识确定: 首先, 社区必须就升级达成共识. 包括调整刻面内的特定变量值或从钻石中添加和移除切面.

2) 创建提案: 至少有一个 DAO 成员在 DAO 中创建一个提案, 通过命令行接口 (CLI) 或者用户界面 (UI) 附加相对应的动作.

3) 提案接受和部署: 如果一个提案被社区接受, 任何人都可以部署它, 这个步骤需要支付 Gas 费用, 因为提案的部署记录在区块链上.

4) 提案执行: 当提案被执行时, 通过自动调用钻石合约中的相应函数来执行与提案相关的动作. 例如当提议执行钻石切割的动作, 那么 IDiamond_Cut 接口中定义的 diamond_Cut 函数将被调用, 其中的参数定义具体动作.

社区投票的升级方式保留智能合约的原始状态, 通过模块化设计提供更高的扩展性, 支持按需升级特定功能模块, 而无需重写或替换整个实现合约, 减少对单一合约的存储依赖. 并通过社区投票机制引入去中心化治理, 允许社区成员参与合约的管理和升级. 这样既增强治理的透明度, 也赋予用户更大的控制权.

钻石模式的灵活性伴随着一定的复杂性, 尤其在权限边界定义和业务模块升级等方面. 由于逻辑和存储的分离, 多重模块化设计需要开发者明确每个 facet 的权限边界, 以确保不同模块的功能调用不会产生权限越界问题. 由于 Solidity 使用固定的存储槽管理变量, 不同模块若未采用明确的存储布局规划, 可能会导致关键变量 (如管理员地址、余额等) 被覆盖或篡改, 引发权限失控或功能异常. 如果在实现中发现错误并部署修复程序, 在多个切面依赖共享状态的情况下, 需要同步更新业务逻辑以防止不一致问题.

1.7 信标代理

2022 年, EIP-4788[58]定义了信标代理 (beacon proxy), 合约本身不包含任何业务逻辑, 而是通过更新一个指向最新实现合约的指针来实现升级. 因此, 不再需要在状态合约上保留任何状态, 消除对非结构化存储的需求. 如图 15 展示信标代理部署过程.

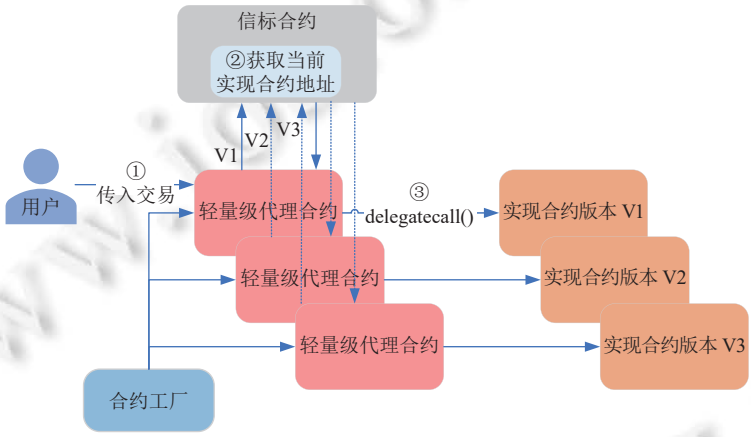

图 15 Beacon 部署过程图

1) 实现合约 (implementation contract): 实际执行业务逻辑的合约. 实现合约的地址可以由 beacon 合约动态更新.

2) 信标合约 (beacon proxy): Beacon 类似于指针, 通过公共函数向代理合约提供当前的实现合约的地址. 允许更新实现合约的地址.

3) 代理合约 (proxy contract): 所有用户的交互通过代理合约进行, 代理合约本身不包含任何业务逻辑, 只负责从 beacon 合约中获取当前的实现合约地址, 并将调用转发给该实现合约.

4) 合约工厂 (可选): 用于自动化部署和管理代理合约. 并在代理合约中设置 beacon 合约的地址, 以便代理合约可以从 beacon 合约获取逻辑地址.

当合约工厂的轻量级代理合约收到一个传入交易时, 根据轻量级代理合约中存储 beacon 的地址, 首先调用 beacon 上的 view 函数以获取当前的实现合约地址, 然后轻量级代理合约 delegatecall() 到实现合约的地址. 这种模式具有高度可扩展性, 因为每个额外的代理只需从 beacon 读取实现地址, 然后使用 delegatecall() 函数. 多个代理共享一个信标, 可以一次性集中升级所有代理的逻辑. 因此它们不需要关心诸如管理员是谁或如何更改实现地址等细节. 这样, 当需要升级实现合约时, 只需部署新的实现合约并更新 beacon 合约中的地址, 无需更改代理合约或用户的交互方式, 所以一般比 UUPS 或透明代理更简单和节省 Gas 费用.

然而信标模式中容易产生单点故障和中心化风险. Beacon 合约作为所有代理合约的核心, 如果 beacon 合约的逻辑或存储出现问题, 所有依赖它的代理合约都会受到影响. 例如实现合约地址更新到错误地址 (例如未初始化或不存在的合约) 或指向一个恶意或存在漏洞的实现合约. Beacon 合约的控制权通常集中于少数管理员. 如果这些管理员的权限被恶意行为者获取 (如私钥泄露), 攻击者可以更改实现地址, 影响所有代理合约的安全性.

2 升级需求

对于区块链系统而言, 修复漏洞最直接的方法是修补源代码, 因此智能合约代码的升级被称为“软分叉”. 然而, 合约源代码并不总是可供参考: 目前部署在以太坊上的 4 900 万智能合约中, 只有 0.3% 的源代码公开可用. 此外, 由于合约可能调用其他合约的函数, 闭源合约中存在的安全漏洞, 也可能影响到它依赖开源合约. 因此, 本节从合约优化、合约部署、合约维护[59]这 3 个方面分析当前的相关工作.

2.1 合约优化

目前关于可升级智能合约的应用研究仍较为薄弱[60,61], 特别是在部署和执行升级操作的 Gas 成本方面. 已部署合约中超过 90% 的包含死代码或不透明的代码, 并且超过 70% 的合约使用 Gas 消耗较大的循环语句[62], 意味着代码优化不足的问题几乎困扰着所有的智能合约[63].

Marin[17]基于 DApp 架构系统化分析可升级智能合约的设计模式, 通常需要考虑合约的结构大小、升级的频率以及调用的复杂性. Marchesi 等人[64]分类出 24 种设计模式, 并提供一套优化方案帮助在以太坊上开发可升级智能合约时节省 Gas. Chen 等人[65]总结 10 种 Gas 效率低下的编程模式, 并提出一种基于符号执行的方法, 用于检测智能合约字节码中的这些问题. Hu 等人[66]分析当前最新的智能合约构建和执行方案, 从合约构建的设计范式、工具、扩展方案中发现: 不完整的范式、效率低下的分析工具、复杂性有限的合约以及权限隐私性缺失是阻碍智能合约应用的主要挑战.

2.2 合约部署

一旦合约部署或升级, 存储在上一版本合约中的所有数据都会被清除[4], 这与当前通过代理合约解决漏洞、增加功能的思路并不相符. Kim 等人[67]提出一个允许加密资产恢复的智能合约, 通过设置截止日期使资产可退回, 用于确保区块链加密资产意外丢失或烧毁时能够恢复.

Dai 等人[68]讨论物联网产业升级的相关挑战, 针对工业中固件升级功能应如何在整个工业网络中部署. Dorri 等人[69]提出一个基于区块链的互联智能汽车框架, 提供及时的个性化服务例如无线远程软件更新及动态车辆保险费用等新兴服务. Wang 等人[70]提出基于区块链的预制施工供应链溯源和信息共享框架, 根据预制建筑项目供应链的加工情况对智能合约功能进行更新.

为保证数据合约的可信性与复杂实现合约的可操作性, Zhang 等人[71]提出一种基于区块链的碳交易智能合约设计, 将交易智能合约拆分为入口合约、实现合约、存储合约这 3 个模块. 这种拆分方式不仅绕过合约大小限制, 还简化智能合约的升级, 同时实现复杂功能. 出现问题时, 拆分方式还可以让管理员账号直接关闭关键操作. Leng 等人[72]设计一个四合一的金字塔型区块链智能合约以实现去中心化的工业制造控制, 合约上层实现各种个性化需求的初始任务调度, 而下层应对内部随机中断的快速自主控制. Lu 等人[73]在非现场物流和现场装配服务的背景下, 开发一个区块链赋能的建筑供应链管理系统架构, 并分为 4 个主要的智能合约对服务质量和信誉评价的更新.

当多个合约同时执行时, 同步和并发操作会导致访问合约逻辑数据的冲突问题. Feng 等人[74]提出一种分解方法将合约分解为多个原子合约. 并根据合约内容顺序或并行执行多个步骤, 防止死锁, 一旦合约履行或到期, 就需要注销. Nelaturu 等人[75]提出 VeriSolid 框架, 引入图形符号来指定合约类型之间可能的交互. 该框架允许开发人员在较高的抽象层次上推理和验证一组交互合约的行为. 访问控制逻辑与业务逻辑的解耦进一步提高系统效率. Sookhak 等人[76]讨论智能合约用户在医疗保健中的访问控制挑战, 在区块链中添加一个新区块用于数据拥有者更新智能合约. 然而, 在用户数量巨大的大公司中, 区块链上实现撤销功能的存储开销极高. Gilani 等人[77]提出一个

自我主权身份的概念, 用户可以维护与特定属性相关联的身份, 并通过智能合约的证明机制进行验证. Chatterjee等人[78]提出一个基于角色的 DApp 动态访问控制框架, 允许业务逻辑的独立管理和访问控制策略的执行. 促进访问控制策略的去中心化治理和高效的智能合约升级. Xue 等人[79]基于智能合约的移动车联网漫游服务分布式认证方案, 通过存储有限的智能合约实现撤销验证功能, 支持 5G 移动网络的庞大用户规模. 但也存在匿名和用户冒充攻击的漏洞.

为修复复杂的业务逻辑问题, Kim 等人[80]提出一种可更新智能合约的分布式和联邦身份验证方案, 对原有的区块链加密服务提供商模块进行升级. Yang 等人[81]提出一种联盟区块链信息系统, 智能合约通过权限管理减少数据总量, 提高查询效率. Kumar 等人[82]提出基于区块链的零接触网络 (zero touch network), 通过执行合约操作 (如签名验证、时间戳和公钥验证) 进行验证. 验证成功后, 返回的响应会被记录并授予访问权限 (如创建、读取、更新和删除操作).

2.3 合约维护

以太坊智能合约开发部署后通常会面临一系列的代码维护问题. Chen 等人[83]总结 4 种常见的维护类型, 即纠正维护、适应性维护、完善维护和预防性维护. Samreen 等人[84]讨论 DApp 代码复杂性、安全性、可维护性之间的关系, 并定义维护合规系数. 结果显示: 以太坊 DApp 的可维护性与代码大小、函数数量成正比, 传出调用操作与语句的数量成正比.

目前在闭源合约 (即 EVM 字节码) 上运行的最流行的修补工具是 EVM Patch[85], 利用字节码重写器将最小侵入式补丁应用到 EVM 智能合约中. 如图 16, EVM Patch 框架由以下 4 个部分组成.

1) 漏洞检测引擎: 由自动分析工具和公开的漏洞披露组成, 用于检测合约中的漏洞.

2) 字节码重写器: 将补丁应用到合约中, 通过字节码重写来修复漏洞.

3) 补丁测试机制: 验证补丁的有效性, 确保补丁不会影响合约之前的事务.

4) 合约部署组件: 上传补丁版本的合约, 将更新后的合约部署到区块链上.

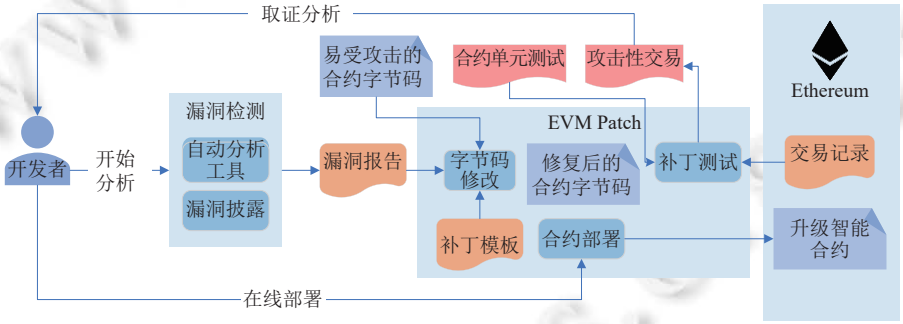

图 16 EVM Patch 的体系结构

漏洞检测引擎首先识别漏洞的位置和类型. 然后将这些信息传递给字节码重写器, 字节码重写器根据先前定义的补丁模板对合约进行补丁. 补丁合约随后被转发给补丁测试人员, 测试人员将所有过去的交易重新回放到合约中, 该过程不仅对合约进行修补, 还允许开发商检索原始合约和补丁合约之间异常行为和结果的交易列表. 这些交易可作为潜在攻击的指示器, 如果列表为空, EVM Patch 框架将自动在链上部署补丁合约.

如图 17 所示, 这种字节码重写方法允许开发者通过 ADD 指令自动引入补丁, 并将其部署在合约的末尾 (0xFFB 处) 的基本块, 而不需要更改位于较高编号地址的代码中的任何地址. 用 INVALID 指令填充原始基本块的其余部分, 以确保基本块与原始基本块具有完全相同的大小. 0xCD 处的基本块通过结尾处与之前的基本块相连. 并以 JUMPDEST 指令作为合法的跳转目标. 为确保在地址 0xCD 处的原始合约代码能够继续执行, 再写入器将在 0xFFB 处的补丁基本块上追加一个跳转指令.

类似的方法, 如 Zhang 等人[86]提出 Smart Shield 用于自动修复智能合约中的安全漏洞, 并确保修复的合约对后续重入、拒绝服务等攻击免疫, 而且对 Gas 消耗友好. 如图 18 所示, Smart Shield 工具通过 3 个步骤来修复合约中的安全漏洞.

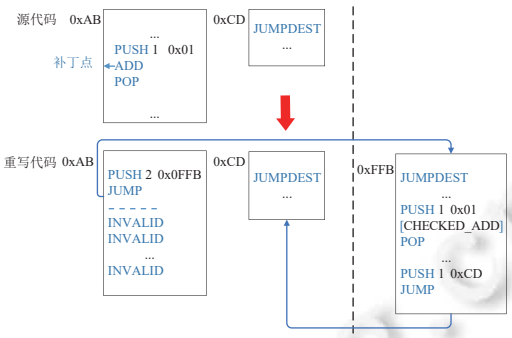

图 17 源代码和重写代码的控制流图

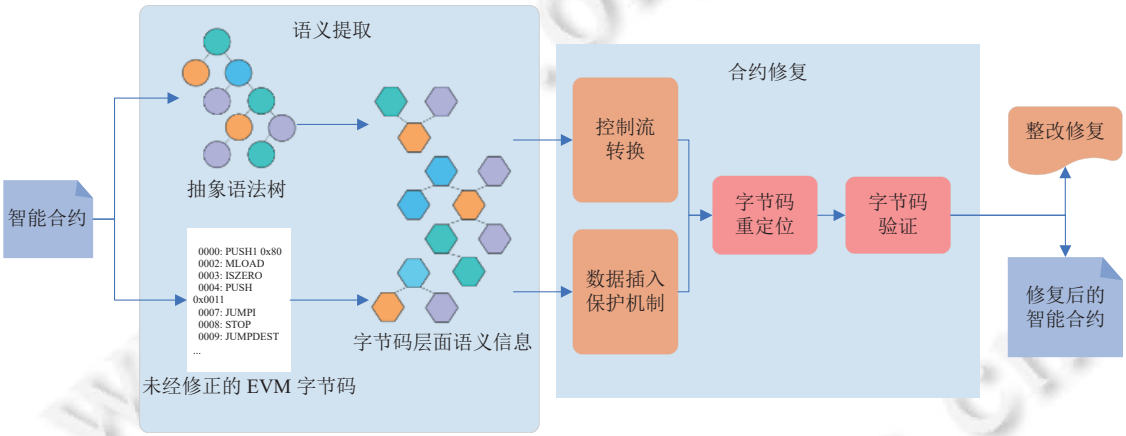

图 18 Smart Shield 工作流程

1) 语义提取: 分析每个合约的抽象语法树及其未校正的 EVM 字节码, 修复不安全的代码模式, 同时保持其他功能的一致性.

2) 合约修复: 采用控制流转换和数据插入保护机制两种策略来修复合约中的漏洞, 并优化 Gas 消耗.

3) 修正合约生成: 字节码重定位和字节码验证可能导致功能行为变化或兼容性副作用, 并将修复报告发送给开发人员改进合约的代码质量.

EVM Patch 和 Smart Shield 都是利用纯硬编码模板的补丁修复模式, 因此无法动态适应正在打补丁的字节码, 这严重限制它们的灵活性和可扩展性. 例如, 当使用硬编码模板修补整数溢出时, 需要使用特定的补丁模板, 因为要检查的边界对于每个整数大小都是不同的 (即一个模板用于 uint256, 另一个模板用于 uint64). Ferreira 等人[87]提出 Elysium, 能够在字节码级别实现自动智能合约修复的可扩展. 通过从字节码推断上下文信息, 结合基于模板和语义补丁自动修补合约中的 7 种不同类型的漏洞, 并且可以使用新的模板和漏洞查找工具进行扩展. Qin 等人[88]提出一种可理解描述语言合约, 实现从自然语言合约到“可理解描述句子”的合约转换, 用于修补合谋风险. Chen 等人[89]提出一种将 EVM 智能合约升级到 WASM 的二进制翻译方案 (EVMBT), 并配备 LLVM 汇编的优化能力和 5 个修复智能合约漏洞的插件. Chu 等人[90]总结现有的研究, 主要集中在合约部署前如何对检测到的漏洞进行修复. 表 4 为基于可升级智能合约的应用升级需求方法的小结.

表 4 基于可升级智能合约的应用升级需求方法小结

类型	名称	发表年份	主要工作
合约效率	Lopez Marin[17]	2022	分析DApp架构的可升级智能合约设计模式
	Marchesi等人[64]	2020	分析24种设计模式, 提供一套基于以太坊的Gas优化方案
	Chen等人[65]	2020	分析10种Gas效率低下的编程模式, 并提出一种基于符号执行的方法
	Hu等人[66]	2021	分析当前最新的智能合约构建和执行方案
合约应用	Kim等人[67]	2022	允许加密资产恢复的智能合约
	Dai等人[68]	2019	讨论工业智能合约的固件升级功能如何部署在工业网络中
	Dorri等人[69]	2017	基于区块链的互联智能汽车个性化远程软件更新服务
	Wang等人[70]	2020	根据预制建筑项目供应链的加工情况进行合约功能更新
	Zhang等人[71]	2023	提出一种基于碳交易的智能合约拆分设计
	Leng等人[72]	2023	智能合约个性化需求调度更新服务
	Lu等人[73]	2021	对建筑供应链管理服务质量和信誉评价的更新
	Feng等人[74]	2019	智能合约分解, 根据合约内容顺序或并行执行多个步骤, 防止死锁
	Nelaturu等人[75]	2022	引入图形符号 (称为部署图) 来指定合约类型之间交互推理
	Sookhak等人[76]	2021	数据拥有者的用户控制访问权限更新智能合约
	Gilani等人[77]	2022	自我主权维护与特定属性相关联的身份
	Xue等人[79]	2022	移动车联网中基于智能合约的漫游服务实现撤销验证功能
	Kim等人[80]	2023	可更新智能合约的分布式和联邦身份验证方案
	Yang等人[81]	2022	提出一种联盟区块链信息系统优化查询效率
	Kumar等人[82]	2022	基于区块链的零接触网络执行合约升级操作授予访问权限
合约维护	Chen等人[83]	2021	讨论部署后以太坊智能合约开发的维护相关问题
	Samreen等人[84]	2023	定义DApp代码维护合规系数
	Rodler等人[85]	2021	利用字节码重写器将最小侵入式补丁应用到EVM智能合约中
	Zhang等人[86]	2020	通过控制流转换和数据插入保护机制两种策略来修复合约
	Ferreira等人[87]	2022	提出Elysium, 在字节码级别实现自动智能合约修复的可扩展方法
	Qin等人[88]	2021	提出一种可理解描述语言合约, 实现合约转换, 用于修补合谋风险
	Chen等人[89]	2024	提出一种将EVM智能合约升级到WASM的二进制翻译方案EVMBT
	Chu等人[90]	2023	总结现有的研究主要集中在合约部署前如何对检测到的漏洞进行修复

3 升级框架

2021 年, Reijers 等人[91]通过反思法律哲学中关于构建实证主义法律秩序的长期辩论, 探讨“链上”和“链下”治理的问题[18]. 对于未部署的智能合约, 需要更高效的链下修复方案来保证合约安全. 而对于已部署的合约, 则需要进一步发展链上升级技术, 以便实时更新补丁合约确保合约安全. 为实现这一目标, 需要全面分析智能合约安全升级框架, 从链下和链上两个维度来保护智能合约的安全.

3.1 链下升级框架

智能合约的链下升级通常指在区块链之外对合约数据或逻辑进行更新, 这种更新不会直接影响链上合约代码. 链下升级涉及数据层和计算层的优化, 例如使用链下计算来减轻链上计算负担, 或者通过链下数据存储来提高效率. 这种方法的优点在于能够提高交易性能和扩展性, 但同时会牺牲一定的去中心化程度, 因为链下组件可能不如链上合约那样透明和不可篡改.

OpenZeppelin[92]是用于在以太坊平台上构建安全、模块化和可重用智能合约的插件. 它提供一套全面的库和工具, 利用代理允许开发人员部署合约作为用户和实现合约之间的中介, 帮助开发人员编写包括透明代理、可升级代理或信标代理等升级模式. 主要目标是通过提供经过审计和测试的合约模板和工具, 减少智能合约开发中的

安全漏洞和常见错误. 例如, 许多 DeFi 协议使用 OpenZeppelin 升级合约来更新和改进功能, DAO 将其用于升级新的治理规则和决策.

类似的工具例如 Foundry[93], 它由 Paradigm 开发, 用于以太坊智能合约开发的高效、模块化和安全的框架. Foundry 提供两个主要开发组件: Forge 和 Cast. Forge 用于编写和测试合约的工具, 而 Cast 是用于与以太坊网络交互的命令行工具. 此外 Foundry 也能够提供详细的 Gas 消耗报告, 帮助开发者识别和优化高 Gas 消耗的操作. 虽然正在快速发展, 但相比于更成熟的框架如 Truffle 和 Hardhat. Foundry 的社区和文档资源相对较少. 对于没有使用过该框架的开发者来说, 可能需要一些时间来适应 Foundry 的工具链和工作流程.

2020 年, Angelo 等人[94]提出一种通过对大量智能合约进行分类, 使用差分代码的方式来支持智能合约链下升级的方法. 同年, Shao 等人[95]开发一个智能合约漏洞监控框架, 该框架将日志异常分析方法引入区块链日志系统, 并将检测结果传输到相关合约, 最后实现智能合约链下升级的提示. Jean-Louis 等人[29]提出基于可信执行环境的智能合约修复区块链架构 SGXonerated, 讨论访问模式泄露、软件升级机制和状态一致性等问题, 并介绍基于可信执行环境的智能合约安全平台, 包括秘密网络 (secret network)、Oasis、Phala 和 Obscuro 等.

3.2 链上升级框架

链上升级通常指的是在区块链上直接对智能合约进行升级. 这种方式需要在合约代码中实现升级机制, 允许合约逻辑在保持合约地址和状态不变的情况下进行更新. 链上升级的常见实现方式是使用代理模式, 其中代理合约负责存储状态和转发调用, 而实现合约包含实际的业务逻辑. 当需要升级时, 只需更新代理合约中指向实现合约的引用地址即可. 这种方式的优点是用户无需知道合约已经升级, 因为他们总是与代理合约交互. 缺点是实现相对复杂, 并且如果代理模式使用不当, 可能会引入安全风险, 如函数选择冲突等问题. 如图 19 所示, 传统的 3 层智能合约模型包括接口层、业务层、数据层. 其中数据层不能升级, 同时也忽略合约链上升级的安全性.

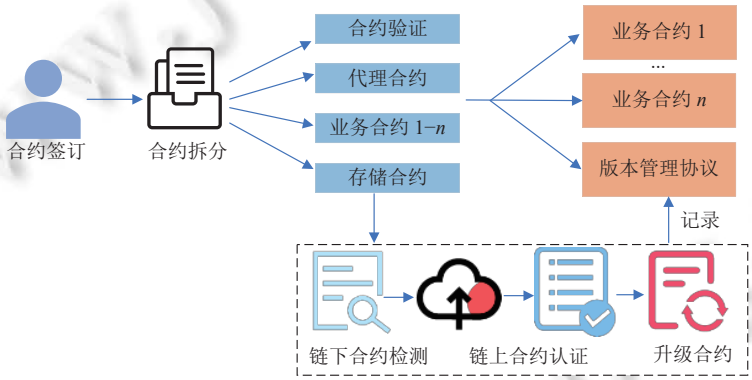

图 19 智能合约部署架构和链下数据流程

1) 接口层 (interface layer): 通常由代理合约实现, 作为与用户或其他合约直接交互的接口, 定义包括函数的声明和用户可以调用的方法.

2) 业务层 (business logic layer): 包含合约的核心业务逻辑, 放置在一个或多个独立的实现合约中. 当需要升级时, 只需替换或更新这些实现合约, 而不需要改变接口层.

3) 数据层 (data layer): 数据层通常由代理合约实现, 它保存实现合约的地址和合约的状态变量.

当需要升级智能合约时, 通常会部署一个新的实现合约, 并通过接口层的代理合约更新对实现合约的引用, 而不需要改变接口层或数据层. 接口层将调用转发到业务层的实现合约, 实现合约包含实际执行的代码. 数据层存储合约的状态 (如账户余额、所有权信息等), 这些状态信息由代理合约管理. 因此用户通过同一个接口层与合约交互, 无需知道背后发生升级, 而合约的业务逻辑已经更新.

2021 年, Liu 等人[96]提出一个智能合约访问控制服务框架, 通过将昂贵的访问控制规则 (access control rule, ACR) 验证和管理操作的负担转移到链下基础设施, 同时仅实现链上轻量级的基于令牌的访问控制. 例如, 存储公

钥、解析令牌和每次调用都会执行一次签名验证. 这种方法减少链上存储和计算需求, 以低成本实现智能合约的可更新和复杂的 ACR. 2021 年, 刘云霞等人[97,98]提出基于区块链的通证智能合约链上升级方法研究, 将传统智能合约拆分为接口合约、实现合约、数据合约这 3 个子集, 3 个子集间保持互相调用的松耦合关系, 共同实现对事务的处理. 此外, 考虑到现有的 3 层模式只能实现部分链上升级, 无法保证链上升级的安全性. 2023 年, Du 等人[99]提出一个 4 层合约模型应用于区块链架构, 如图 20, 包括代理层、验证层、业务层和存储层. 与 3 层模型相比, 重新定义存储层, 以面向库表开发方式抽象底层的库表合约, 并实现数据的统一存储. 增加一个验证层, 并引入交易回滚机制, 所提出的 4 层模型和算法能够实现链上升级, 以部分整体部署为代价降低合约复杂度和数据迁移成本.

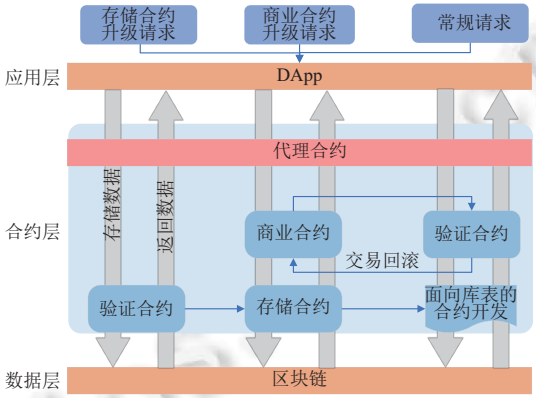

图 20 4 层模型的总体设计

Nelaturu 等人[75]引入 VeriSolid 框架, 用于具有严格操作语义的基于抽象状态机模型, 指定合约的正式验证. 该框架利用 initialize_public 方法, 可在整个合约生命周期的任何时间点使用, 用于动态改变合约逻辑. VeriSolid 支持正确设计、开发和部署多个交互智能合约. 并扩展功能以满足复杂应用场景的需求. Jin 等人[100]提出一种名为 Aroc 的通用智能合约修复程序, 能够修补部署在链上的脆弱合约. Aroc 首先基于固定模板生成包含安全规则的补丁合约, 并将补丁部署到区块链上, 然后将存在安全漏洞的合约与补丁合约捆绑在一起, 从而使后续交易在调用原始合约之前, 必须先调用补丁合约. 表 5 为基于链上/链下升级框架的方法小结.

表 5 基于链上/链下升级框架的方法小结

类型	名称	发表年份	主要内容
链下升级框架	Angelo等人[94]	2020	对大量智能合约进行分类, 使用差分代码来支持智能合约链下升级
	Shao等人[95]	2020	智能合约漏洞日志异常分析, 实现智能合约链下升级的提示
	Jean-Louis等人[29]	2024	基于可信执行环境的智能合约修复区块链架构SGXonerated
链上升级框架	Liu等人[96]	2020	智能合约访问控制服务框架, 实现链上轻量级的基于令牌的访问控制
	刘云霞等人[97,98]	2021	基于区块链的通证智能合约链上升级方法
	Du等人[99]	2023	提出4层合约模型, 引入交易回滚机制实现链上升级
	Nelaturu等人[75]	2022	提出VeriSolid框架, 用于使用具有严格操作语义的基于抽象状态机的模型指定的合约的正式验证
	Jin等人[100]	2021	提出Aroc通用智能合约修复程序, 修补部署在链上的脆弱合约

4 升级规范

尽管以太坊的生态系统有了明显的改善, 但开发人员仍然声称缺乏标准、组件库和有用的参考代码[101]. 这种情况在设计可升级智能合约中体现得更加明显. 由于上市时间的压力、发布的紧迫性、软件开发工具的不成熟、高质量培训资源的缺乏, 以及以太坊路线图的不明确和生态系统的快速发展, DApp 开发者难免会在智能合约中

引入一定程度的技术债务[19], 例如代码中的漏洞代码、临时性解决方法和非最优实现. Ebrahimi 等人[19]研究智能合约中的技术债务, 指出开发人员通常通过代码注释在源代码中明确承认存在的技术债务, 导致选择一个快速的、次优的解决方案需要更长的时间来解决该方案所造成的额外工作成本. 此外, Li 等人[47]讨论一些合约升级时潜在的安全问题, 包括缺少权限限制性检查、逻辑地址检查、合约版本检查问题.

4.1 升级性安全管理

首先, 需明确区分“合约升级”与“合约可升级性”这两个概念. Qasse 等人[18]认为合约升级是指在部署后修改智能合约代码, 同时维护合约数据和状态的过程. 而合约可升级性指的是合约是否具备被修改和升级的潜在能力, 因此并非所有具备可升级性的合约都会被实际修改或升级. 区分这两者一个关键的特性是: 可升级性是否可能在不经意间被移除? 在代理模式中, 如果升级机制被设计在代理合约之外, 无论是嵌入在实现合约中还是在外部合约控制, 合约的可升级性都可能被意外且永久地移除. 一些以太坊改进提案 (EIP, 如 EIP-1822) 警告, 为保留可升级性, setter() 必须存在于实现合约中. 但其他一些以太坊改进提案 (如 EIP-2535) 则持不同立场, 认为使其代码不可变是一种功能特性而非代码漏洞.

4.1.1 升级性风险

在设计智能合约时, 合约的可升级性是一个重要的考虑因素. Xu 等人[102]研究两种旨在提高合约升级能力的模式: 合约注册模式和数据合约模式. 合约注册模式通过合约注册中心来管理合约地址, 从而实现合约的升级, 例如, AAVE 的早期版本就使用这种模式. 数据合约模式则是将数据和逻辑分离, 通过升级实现合约来实现合约的升级, 同时保留数据不变. 此外, 考虑到合约升级的安全性, 提出嵌入许可和工厂合约两种模式. 嵌入许可模式通过在合约中嵌入权限控制逻辑, 确保只有授权的实体才能执行关键操作. 工厂合约则通过创建标准化的合约实例来简化部署和管理过程. 例如钻石框架就是一种特殊的合约升级模式, 通过预定义的逻辑, 在不影响其状态的情况下来处理升级, 并有效避免硬分叉问题.

在设计复杂的智能合约时, 必须综合考虑包括可升级功能、版本约束以及数据治理在内的多个方面[103]. 在代理合约中, 如果未正确地实现升级功能, 具有升级控制的单个实体可能会引入潜在的漏洞或恶意更改. 此外, 在去中心化的系统就升级达成共识是复杂和漫长的, 同样会增加用户信任和运营担忧. Salehi 等人[46]将可升级性的安全控制策略细分为 3 个主要类别: 私钥控制问题、版本升级延迟以及代码重用灾难.

1) 私钥控制问题: 由于私钥的高度集中, 一个恶意的管理员或被泄露的私钥可能是灾难性的[57]. 事实上已经报道几起关于代理合约的事件. 例如, 在 PAID 事件中, 攻击者破坏管理员的私钥并篡改实现合约, 使他们能够随意燃烧和铸造 PAID 代币. 如果攻击者获得控制代理的管理特权, 他们可以将代理重定向任何逻辑.

2) 版本升级延迟: 如果在可升级合约中发现漏洞并需要修补, 恶意用户在下一个版本升级之前的时间窗口内利用该漏洞. Chen[104]提出一种通过比较以太坊智能合约的历史版本来识别安全问题的方法, 通过寻找自毁合约的更新版本并计算代码相似性, 来追踪升级过程中的安全问题. 然而, 代理模式为 DApp 引入额外的层次, 这可能会增加事务处理时间. 此外, 与单个合约相比, 开发人员至少需要维护两个合约 (代理合约和实现合约). 这种升级增加维护负担, 因为开发人员需要监控、更新并确保在不同版本的代理合约之间的兼容性. 对于最终用户来说, 他们可能并不总是清楚地知道他们正在与代理合约进行交互, 这种差异导致对当前合约实际功能的混淆或误解, 特别是在用户不知情的情况下更改逻辑.

3) 代码重用灾难: Abuhashim 等人[105]将智能合约聚类为 39 个可升级的集群, 每个集群包含大量在嵌套智能合约数量和每个组件如何编写方面相似的智能合约. 一旦发现初始重用合约存在漏洞时, 代码重用将会无形中传播漏洞灾难.

4.1.2 自毁智能合约

无论是不可升级的合约, 还是使用代理模式的可升级合约, 基本共识是: 已经部署在区块链上的原始合约是不可变的. 然而 EVM 有一个操作码 SELFDESTRUCT, 使得改变链上的合约可以被删除. 自毁合约是从以太坊区块链中删除智能合约的唯一方法. 当执行此 SELFDESTRUCT 时, 合约可以将所有以太币余额转移到特定地址, 然后

废弃合约本身. 通过使用 SELFDESTRUCT, 开发人员可以在紧急情况发生时 (如受到攻击) 删除智能合约并转移以太币. 此外, 当不再需要合约时, 调用 SELFDESTRUCT 可以帮助清理以太坊环境.

自毁智能合约[106]的概念主要有两种实现方式, 它们允许在特定条件下通过合约自身的机制终止其功能.

1) 如果实现合约中包含 SELFDESTRUCT 操作, 并通过调用它来执行自毁.

2) 对具有 SELFDESTRUCT 逻辑的合约进行 delegatecall() 或 CALLCODE.

在这种情况下, 应该检查实现合约是否使用 SELFDESTRUCT 方法或将 delegatecall()、CALLCODE 设置为恶意方可以控制的地址. 如果有该方法, 恶意方可以通过调用 SELFDESTRUCT 来自毁合约, 所有对代理的调用失败. 这就构成对 DApp 的拒绝服务攻击. 当然, 代理合约的管理员也可以通过升级到新版本的实现合约来应对这种情况. 另外, 可以通过计算代码相似性来识别自毁合约 (即前任合约) 及其更新版本 (继任者合约). 然而 SELFDESTRUCT 函数对于合约用户和合约所有者而言也存在一定的风险, 恶意方可能会利用自毁功能打开攻击向量[107].

尽管使用自毁功能和开发可升级合约在成本上可能更为经济, 但也会因合约可变性而产生信任威胁. Chen[104]提出一种权利分配方法来减轻使用自毁函数所带来的信任和安全问题, 这种方法同样适用于可升级合约的安全管理. 作者建议开发者应该将权利分配给合约的使用者, 让他们投票决定合约是否应该被销毁或升级. 通过引入延迟升级步骤以降低以太币被锁定的风险, 因为转移到销毁地址的以太币将永远无法取回. 在投票和延迟步骤中, 开发者应该暂停合约的功能, 以防止攻击或其他不必要的行为.

4.2 形式化升级规范

代理模式允许对智能合约进行修补, 但它并未解决故障升级问题: 如何确保升级后合约内容的正确性. 此外, 升级过程通常赋予合约维护者较大的权力, 允许他们在缺乏适当监管的情况下更改合约代码, 然而这种更新过程没有任何强制执行保证; 只要正确的参与者批准更改, 合约实现就可以相当随意地更改. 鉴于智能合约的不可变性和自动执行特性, 以及“代码即法律”的原则[108], 在合约的创建和升级之前, 必须进行严格的代码规范验证, 确保其符合预定的“法律”和逻辑规范.

4.2.1 合约升级规范

Casolari 等人[109]就合约规范探讨《欧盟数据法》提案对智能合约的实际影响, 并强调在创建和升级合约之前进行正式验证的必要性. 然而, 目前大多数合约规范都是非正式的, 一些区块链社区使用的规范格式, 例如以太坊征求意见 (ERC)[110], 在 ERC-20 中仅描述合约实现的功能签名, 以及关于功能应如何表现的简短文本、非正式解释等相关规范. 并未提及如何确保合约规范在其生命周期内保持不变. 参与者可以通过框架检查合约是否已部署, 以便他们可以确定他们想要执行的合约具有预期的行为.

Antonino 等人[108,111]提出一种“规范即法律”智能合约安全部署与演化方法, 该方法基于可信部署者创建的安全框架来管理智能合约升级. 该框架通过在链上的钻石代理执行升级操作, 而链下依靠受信任的部署者对合约创建以及升级规范进行审查. 仅当新的实现满足预期规范时才允许执行升级操作. 图 21 介绍受信任部署者的基础架构. 其中可信部署程序依赖于以下组件:

1) 验证者 (verifier): 检查合约的创建和升级是否符合规范.

2) 升级者 (upgrader): 执行合约的创建与升级.

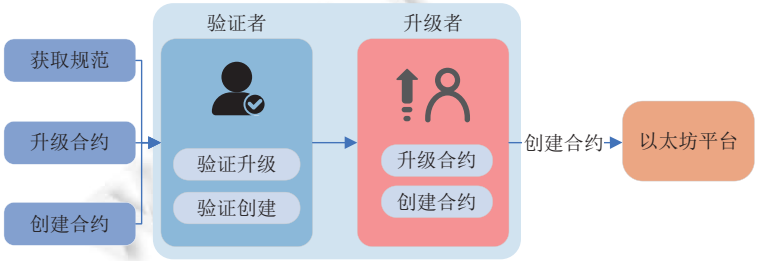

图 21 可信部署者架构

部署者在执行合约的创建与升级操作时, 会先调用验证创建和验证升级进行合约验证. 若验证通过, 部署者才会将调用请求转发给升级者, 最终在以太坊平台上完成合约的创建或升级; 获取规范函数可用于查询受信任的部署者是否已经部署某个合约实例, 并确定该合约满足的规范. 该机制确保智能合约的演化过程始终符合既定的安全要求, 同时避免未经验证的恶意升级.

4.2.2 合约升级证明

区块链的共识机制在于确保智能合约的所有更新都符合原始规范, 任何后续对实现合约的更新都必须伴随着规范化的有效证明. Dickerson 等人[112]将 Necula[113]的携带证明代码改造成携带证明的智能合约, 将正式证明 (API 和原始规范) 附加到智能合约上, 在部署前信任一个规范的第三方来验证所有合约升级. 图 22 展示携带证明的智能合约的升级过程.

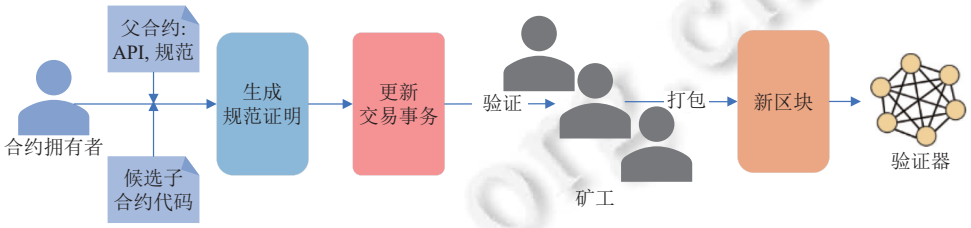

图 22 携带证明的智能合约

1) 假定合约拥有者需要发布父合约 (包含 API 和规范) 以及候选子合约的代码, 因为这些操作不需要验证. 合约拥有者首先生成一个证明, 证明提议的子合约满足规范的不变量.

2) 合约拥有者将此证明打包并更新事务, 作为交易传输到区块链网络进行挖矿.

3) 矿工验证子合约的代码证明, 并生成一个包含安全更新的新区块. 如果证明无效, 区块仍会将尝试的更新记录为事务, 但安全更新将失败并出现异常.

4) 最后, 区块链网络的其余参与者重新运行并验证安全更新.

此外, 合约之间重用关系也可以实现升级证明, Kondo 等人[114]研究显示, 超过 79.2% 的智能合约包含代码克隆, 且克隆合约大部分为代币合约. 从而增加代码维护的可重用性因素. Ringer 等人[115]通过对比新旧合约之间的结构相似性, 在更新的代码上重用形式化完成修复证明. 在此基础上, Sorensen 等人[116,117]提出合约演化概念, 提出一个合约升级的形式化框架 in ConCert, 形式化地编码智能合约之间的结构关系, 用它来指定和推理合约升级的结构和行为. 解决对区块链系统进行范式转换的再工程问题. Galimullin 等人[118]引入的前后时间关系指的是以前版本的合约, 从而为代码设置增加自省功能, 用于验证和规范化智能合约升级的逻辑.

4.3 结构化开发流程

区块链软件开发的范式与传统方法存在显著差异. 主要源于区块链系统对安全性和可靠性的严格要求, 以及去中心化应用开发领域所特有的挑战, 包括数据不可变性、软件升级复杂性, 以及在高度复杂、去中心化的网络环境中运行的安全需求. 然而商业浪潮总是催促应用程序的快速开发, 往往以牺牲良好的开发实践、全面的测试和安全评估为代价. 尽管敏捷开发流程在传统软件开发中取得成功, 但其在满足区块链系统的高标准要求方面存在局限性.

面向区块链的软件工程是一个新兴领域. Porru 等人[119]于 2017 年首次呼吁使用面向区块链的软件工程. 强调“需要新的专业角色、增强的安全性和可靠性、新颖的建模语言和专门的指标”, 并提出“面向区块链的软件工程的新方向, 关注大型团队之间的协作、测试活动和用于创建智能合约的专门工具”. 还建议采用现有的设计符号, 如统一建模语言 (UML), 以明确地指定和记录 DApp 开发流程. 尽管有大量资金投入开发, 但健全的软件工程流程和实践的应用仍然相对较低. 因此目前的可升级智能合约更像是一种系统维护手段.

开发既安全又可升级的智能合约是一项重大挑战, 它要求开发人员具备高度的安全意识和专业知识. 随着越

来越多的智能合约集成升级功能, 攻击者针对这些合约攻击的可能性也成倍增加[47]. Destefanis 等人[120]通过对平价钱包这一研究案例的深入分析, 强调建立专门的区块链软件工程学科的必要性. 智能合约依赖于非标准的软件生命周期, 然而根据常规生命周期, 交付的应用程序很难通过发布新版本的软件来更新或解决错误. Wöhrer 等人[121]认为在智能合约应用 DevOps 中, 需要一种更加灵活的结构化方法和指导方针, 以便将新版本完全自动化地部署到生产环境中. Klinger 等人[66]考虑到不断发展的业务需求或不断变化的业务合作伙伴, 引入流程版本控制概念, 设置民主化的投票机制, 以确保智能合约的可信升级. Marchesi 等人[122]提出一种名为 ABCDE 的敏捷区块链 DApp 工程框架, 该框架详尽地描述设计、开发、测试和集成智能合约及链外软件所必需的关键活动. 该方法使用形式化图表来记录智能合约, 旨在辅助开发、安全评估和维护工作. 它具有通用性, 并通过适当的调整, 可以适用于任何区块链软件项目. Taherdoost[123]探讨数据科学智能合约的潜在发展方向, 提出一种可扩展的方法论, 以支持数据科学领域中可信的数据共享、去中心化的信任建立、数据完整性维护和访问控制. 表 6 为基于可升级智能合约的升级规范小结.

表 6　基于可升级智能合约的升级规范小结

类型	名称	发表年份	主要内容
可升级性问题	Qasse等人[18]	2023	讨论以太坊区块链平台上智能合约的可升级性研究
	Xu等人[102]	2018	认为合约注册和数据合约是两种旨在提高智能合约升级能力的模式. 嵌入许可和工厂合约两种模式旨在提高智能合约的安全性
	Salehi等人[46]	2022	总结3类安全控制问题
	Chen[104]	2020	比较以太坊智能合约的历史版本来识别安全问题
	Abuhashim等人[105]	2022	分类编写方面相似的智能合约发现初始重用合约存在漏洞问题
	Chen等人[107]	2020	投票方法应用于可升级合约, 来减少使用自毁函数的信任和安全问题
形式化升级规范	Casolari等人[109]	2023	讨论《欧盟数据法》提案中智能合约的法律形式
	Antonino等人[108,111]	2022/2024	提出一种新的“规范即法律”智能合约安全部署与演化方法
	Dickerson等人[112]	2018	提出携带证明的智能合约
	Kondo等人[114]	2020	发现超过79.2%的智能合约包含代码克隆, 提出代码的可重用性维护
	Ringer等人[115]	2021	对比新旧合约之间的代码相似性, 代码重用完成修复证明
	Sorensen等人[116,117]	2022/2024	提出合约演化概念, 推理升级智能合约的结构和行为
	Galimullin等人[118]	2022	引入的智能合约的前后时间关系用于规范和验证智能合约升级
结构化开发流程	Porru等人[119]	2017	提出面向区块链的软件工程
	Destefanis等人[120]	2018	通过案例分析, 提出区块链软件工程学科
	Wöhrer等人[121]	2021	在智能合约应用DevOps需要一种结构化的方法和相应的指导方针
	Klinger等人[66]	2020	引入流程版本控制概念, 设置民主化投票机制进行智能合约的可信升级
	Marchesi等人[122]	2020	提出一种ABCDE——敏捷区块链DApp工程框架
	Taherdoost[123]	2023	提出数据科学智能合约潜在发展作为一种可扩展的方法

5　总结与展望

在可编程区块链的框架下, 引入可升级智能合约极大的提升区块链的灵活性. 随着业务需求与技术的不断演进, 开发者能够借助可升级智能合约轻松添加或更新功能, 而无需重新部署整个合约. 这一特性不仅简化合约的维护流程, 也显著降低部署的成本与复杂度. 然而, 当前基于代理模式的可升级智能合约的具体特性尚不为公众所熟知. 本文从可升级智能合约、应用需求、升级框架与安全监督这 4 个角度进行总结, 涵盖可升级智能合约的设计、实现、测试、部署及运维多个阶段, 总结当前可升级智能合约的研究进展, 最后通过本文的总结与展望, 以期为区块链可升级智能合约应用未来发展提供参考.

随着区块链技术的不断进步与应用的深入拓展, 智能合约可升级技术的持续改进与优化, 以满足高效、安全、去中心化的需求. 当前研究的核心目标是优化 Gas 费用、提升可信升级、实现自动优化, 并增强跨链兼容性. 在此

背景下, 未来的研究方向和挑战主要集中在以下 4 个方面.

1) 优化 Gas 费用

智能合约升级通常依赖代理模式来维护状态的连续性, 但其调用逻辑会额外消耗 Gas. 为降低升级成本, 目前主要采用模块化和分层设计, 使单个模块的升级不会影响整体架构, 从而优化合约的性能与可维护性. 例如 UUPS 通过存储插槽减少额外的存储操作. Compound 协议探索模块化分层设计, 使业务功能与治理机制相互独立, 降低全局更新的复杂性和成本. Polygon zkEVM 采用预编译合约进行计算密集型任务 (如哈希计算、椭圆曲线签名验证), 减少 Gas 费用. 除架构上的模块化复用, 还可以利用抽象语法树和代码复用机制来减少升级时的代码变更范围, 提高可扩展性和开发效率.

2) 可信升级技术

对于被赋予合约升级权限的开发者, 试图在不变性与灵活性之间寻求平衡. 当前被广泛采纳的折衷方案是引入延迟升级功能. 例如 Uniswap V3 的治理机制, 允许社区成员在提案批准后有足够时间审查和反应, 以防止恶意升级. 此外, 智能合约的升级权通常由治理合约或多签钱包控制, 然而此模式可能导致中心化风险. 因此, 寻求可信的升级机制是目前的重要研究方向, 一些区块链项目采用携带证明代码和零知识证明. 以增强升级的内容可验证性和安全性. 例如, zkSync 在 Layer2 通过零知识证明实现高效计算的同时, 确保升级过程的透明性, 防止未经验证的合约升级. 未来的研究重点在升级权限与升级内容之间的双重保障, 以确保智能合约在演化过程中兼顾安全性与透明性.

3) 自动优化技术

在智能合约中执行 for 或 while 循环会导致 Gas 消耗迅速增加, 尤其是在链上存储和计算涉及大量数据时. 未来智能合约将不仅局限于“if-then”式的规则, 而是通过结合特定任务的离散动作与语言描述, 实现在大规模语言模型中融合推理与执行的能力, 例如 ChainGPT 结合 AI 和区块链技术, 使智能合约能够通过自然语言解析与逻辑推理自动优化代码, 并适应不断变化的业务需求, 这为智能合约的自动优化提供新的可能性. 因此如何结合大规模语言模型形成更为复杂的多智能体-合约系统是未来的研究方向之一.

4) 跨链兼容性

为确保合约在不同区块链上的逻辑和数据一致性, 去中心化跨链治理协议通过统一的升级规则进行协作和治理. 例如, XCM、IBC 跨链协议允许不同的区块链通过共享的中继链进行安全通信, 实现智能合约的跨链升级. 链下存储或 Layer2 扩展将状态存储从合约中完全分离的架构, 提高系统可扩展性. 未来可以结合 UUPS、透明代理和信标代理等模式的优点, 设计适用于多场景的混合升级架构, 确保合约在多链之间的逻辑和数据一致性情况下, 兼顾灵活性、安全性和性能.

上述研究方向不仅将推动区块链技术的发展, 还将为各行各业带来更高效、更智能的解决方案. 随着升级技术的不断成熟, 可升级智能合约将在未来的数字经济中扮演至关重要的角色.

References

[1] Szabo N. Smart contracts: Building blocks for digital markets. EXTROPY: The Journal of Transhumanist Thought, 1996, 18(2): 28.

[2] Grigg I. The ricardian contract. In: Proc. of the 1st IEEE Int'l Workshop on Electronic Contracting. San Diego: IEEE, 2004. 25–31. [doi: 10.1109/WEC.2004.1319505]

[3] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. 2008. https://bitcoin.org/bitcoin.pdf

[4] Buterin V. Ethereum white paper. 2013. https://ethereum.org/en/whitepaper/

[5] Cui ZQ, Yang HW, Chen X, Wang LZ. Research progress of security vulnerability detection of smart contracts. Ruan Jian Xue Bao/Journal of Software, 2024, 35(5): 2235–2267 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/7046.htm [doi: 10.13328/j.cnki.jos.007046]

[6] Dong WL, Liu Z, Liu K, Li L, Ge CP, Huang ZQ. Survey on vulnerability detection technology of smart contracts. Ruan Jian Xue Bao/Journal of Software, 2024, 35(1): 38–62 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6810.htm [doi: 10.13328/j.cnki.jos.006810]

[7] EOS.IO. EOS.IO Technical White Paper v2. 2018. https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md

[8] Neo. Neo Documentation. 2024. https://docs.neo.org/docs/index.html
[9] Androulaki E, Barger A, Bortnikov V, Cachin C, Christidis K, De Caro A, Enyeart D, Ferris C, Laventman G, Manevich Y, Muralidharan S, Murthy C, Nguyen B, Sethi M, Singh G, Smith K, Sorniotti A, Stathakopoulou C, Vukolić M, Cocco SW, Yellick J. Hyperledger fabric: A distributed operating system for permissioned blockchains. In: Proc. of the 13th EuroSys Conf. Porto: ACM, 2018. 30. [doi: 10.1145/3190508.3190538]
[10] Amsden Z. Libra Whitepaper. 2019. https://whitepaper.io/document/476/libra-1-whitepaper
[11] The ZILLIQA team. The ZILLIQA Technical Whitepaper. 2017. https://docs.zilliqa.com/whitepaper.pdf
[12] Maesa DDF, Mori P. Blockchain 3.0 applications survey. Journal of Parallel and Distributed Computing, 2020, 138: 99–114. [doi: 10.1016/j.jpdc.2019.12.019]
[13] Bagozi A, Bianchini D, De Antonellis V, Garda M, Melchiori M. Services as enterprise smart contracts in the digital factory. In: Proc. of the 2019 IEEE Int'l Conf. on Web Services (ICWS). Milan: IEEE, 2019. 224–228. [doi: 10.1109/ICWS.2019.00046]
[14] Negara ES, Hidayanto AN, Andryani R, Syaputra R. Survey of smart contract framework and its application. Information, 2021, 12(7): 257. [doi: 10.3390/info12070257]
[15] Compound. Compound.finance. 2022. https://compound.finance/
[16] Open Sea. OpenSea NFT marketplace. 2022. https://opensea.io/
[17] Lopez Marin JC. Upgradeable smart contracts design patterns for DApps architectures [MS. Thesis]. Dublin: National College of Ireland, 2022.
[18] Qasse I, Hamdaqa M, Jónsson BÞ. Smart contract upgradeability on the Ethereum blockchain platform: An exploratory study. arXiv:2304.06568. 2023.
[19] Ebrahimi AM, Oliva GA, Hassan AE. Self-admitted technical debt in Ethereum smart contracts: A large-scale exploratory study. IEEE Trans. on Software Engineering, 2023, 49(9): 4304–4323. [doi: 10.1109/TSE.2023.3289808]
[20] Alamsyah A, Kusuma GNW, Ramadhani DP. A review on decentralized finance ecosystems. Future Internet, 2024, 16(3): 76. [doi: 10.3390/fi16030076]
[21] Zhang MY, Zhang XK, Zhang YQ, Lin ZQ. TXSPECTOR: Uncovering attacks in Ethereum from transactions. In: Proc. of the 29th USENIX Security Symp. USENIX Association, 2020. 2775–2792.
[22] Johnson N. Go implementation of the Ethereum protocol. 2016. https://github.com/Arachnid
[23] Araoz M. Proxy libraries in Solidity. 2017. https://blog.openzeppelin.com/proxy-libraries-in-solidity-79fbe4b970fd?gi=1f8d084a621e
[24] Palladino S. The transparent proxy pattern. 2018. https://blog.Openzeppelin.com/the-transparent-proxy-pattern
[25] Barros G, Gallagher P. ERC-1822: Universal upgradeable proxy standard (UUPS). 2019. https://eips.ethereum.org/EIPS/eip-1822
[26] OKX Research Institute. Ethereum 2.0 scheme and progress report. 2022 (in Chinese). https://www.okx.com/zh-hans/learn/eth-2-0-scheme-and-progress
[27] HTX Ventures. Current status and development trends of layer 2. 2023. https://htxventures.medium.com/layer-2-%E7%8E%B0%E7%8A%B6%E5%8F%8A%E5%8F%91%E5%B1%95%E8%B6%8B%E5%8A%BF-3d56606d5795
[28] OpenZeppelin. Introducing OpenZeppelin contracts 5.0. 2023. https://blog.openzeppelin.com/introducing-openzeppelin-contracts-5.0
[29] Jean-Louis N, Li YQ, Ji Y, Malvai H, Yurek T, Bellemare S, Miller A. SGXonerate: Finding (and partially fixing) privacy flaws in TEE-based smart contract platforms without breaking the TEE. Proc. on Privacy Enhancing Technologies, 2024, 2024(1): 617–634. [doi: 10.56553/popets-2024-0035]
[30] Chainlink. How to deploy and use upgradable smart contracts. 2023. https://blog.chain.link/upgradable-smart-contracts-zh/
[31] Breaking the immutability of blockchain: How to achieve intelligent cooperation between agent modes and agents. 2023 (in Chinese). https://www.web3sj.com/news/31407/
[32] ChinaDeFi. Rust network programming in practice: Writing a mini TCP reverse proxy (minginx) with Tokio. 2025. https://learnblockchain.cn/article/8081
[33] ChinaDeFi. Understanding the EIP-2535 diamond standard. 2020. https://learnblockchain.cn/article/1933
[34] ChinaDeFi. In-depth explanation of minimal proxy "EIP-1167". 2021 (in Chinese). https://learnblockchain.cn/article/3400
[35] DApp Migrate.n.d. 2025 (in Chinese). https://devdocs.platon.network/docs/zh-CN/DApp_migrate/
[36] Ortner M, Eskandari S. Smart contract sanctuary: A home for Ethereum smart contracts verified on Etherscan.n.d. 2025. https://github.com/tintinweb/smart-contract-sanctuary
[37] Upgrading smart contracts. 2023 (in Chinese). https://ethereum.org/en/developers/docs/smart-contracts/upgrading/
[38] Proxy patterns. 2018. https://blog.openzeppelin.com/proxy-patterns
[39] Mudge N. Diamond-2-Hardhat: Gas-optimized EIP-2535 diamond reference implementation using Hardhat and Solidity 0.8.*. 2025.

https://github.com/mudgen/diamond-2-hardhat

[40] Huang Y, Wu XY, Wang QQ, Qian ZA, Chen XP, Tang MD, Zheng ZB. The sword of damocles: Upgradeable smart contract in Ethereum. In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Program Comprehension. Lisbon: ACM, 2024. 333–345. [doi: 10.1145/3643916.3644426]

[41] Bodell WE III, Meisami S, Duan Y. Proxy hunting: Understanding and characterizing proxy-based upgradeable smart contracts in blockchains. In: Proc. of the 32nd USENIX Security Symp. Anaheim: USENIX Association, 2023. 1829–1846.

[42] Amri SAL, Aniello L, Sassone V. A review of upgradeable smart contract patterns based on open zeppelin technique. The Journal of the British Blockchain Association, 2023, (6): 1–8. [doi: 10.31585/jbba-6-1-(3)2023]

[43] Ebrahimi AM, Adams B, Oliva GA, Hassan AE. A large-scale exploratory study on the proxy pattern in Ethereum. Empirical Software Engineering, 2024, 29(4): 81. [doi: 10.1007/s10664-024-10485-1]

[44] Meisami S, Bodell WE III. A comprehensive survey of upgradeable smart contract patterns. arXiv:2304.03405, 2023.

[45] Liu Y, Li S, Wu XH, Li Y, Chen ZY, Lo D. Demystifying the characteristics for smart contract upgrades. arXiv:2406.05712, 2024.

[46] Salehi M, Clark J, Mannan M. Not so immutable: Upgradeability of smart contracts on Ethereum. In: Proc. of the 2022 Int'l Conf. on Financial Cryptography and Data Security. Cham: Springer, 2022. 539–554. [doi: 10.1007/978-3-031-32415-4_33]

[47] Li XF, Yang J, Chen JQ, Tang YZ, Gao X. Characterizing Ethereum upgradable smart contracts and their security implications. In: Proc. of the 2024 ACM on Web Conf. Singapore: ACM, 2024. 1847–1858. [doi: 10.1145/3589334.3645640]

[48] Yuan Y, Wang FY. Editable blockchain: Models, techniques and methods. Acta of Automatica Sinica, 2020, 46(5): 831–846 (in Chinese with English abstract). [doi: 10.16383/j.aas.2020.y000002]

[49] Trabelsi H, Zhang KW. Early detection for multiversion concurrency control conflicts in hyperledger fabric. arXiv:2301.06181, 2023.

[50] Lehar A, Parlour CA, Zoican M. Liquidity fragmentation on decentralized exchanges. arXiv:2307.13772v5, 2023.

[51] Gamma E, Helm R, Johnson R, Vlissides J. Design patterns: Abstraction and reuse of object-oriented design. In: Proc. of the 7th European Conf. on Object-oriented Programming. Kaiserslautern: Springer, 1993. 406–431. [doi: 10.1007/3-540-47910-4_21]

[52] Murray P, Welch N, Messerman J. ERC-1167: Minimal proxy contract. Ethereum improvement proposals. 2018. https://eips.ethereum.org/EIPS/eip-1167

[53] MixBytes Team. Collisions of Solidity storage layouts. 2025. https://mixbytes.io/blog/collisions-solidity-storage-layouts

[54] Palladino S, Giordano F, Croubois H. ERC-1967: Proxy storage slots. Ethereum improvement proposals. 2019. https://eips.ethereum.org/EIPS/eip-1967

[55] 0age. Metamorphic: A metamorphic smart contract proxy pattern. 2019. https://github.com/0age/metamorphic

[56] Mudge N. ERC-2535: Diamonds, multi-facet proxy. Ethereum improvement proposals. 2020. https://eips.ethereum.org/EIPS/eip-2535

[57] van Vulpen P, Heijnen H, Mens S, Kroon T, Jansen S. Upgradeable diamond smart contracts in decentralized autonomous organizations. Frontiers in Blockchain, 2024, 7: 1481914. [doi: 10.3389/fbloc.2024.1481914]

[58] Stokes A, Dietrichs A, Ryan D, Swende MH, lightclient. EIP-4788: Beacon block root in the EVM. Ethereum improvement oroposals. 2022. https://eips.ethereum.org/EIPS/eip-4788

[59] Klinger P, Nguyen L, Bodendorf F. Upgradeability concept for collaborative blockchain-based business process execution framework. In: Proc. of the 3rd Int'l Conf. on Blockchain. Honolulu: Springer, 2020. 127–141. [doi: 10.1007/978-3-030-59638-5_9]

[60] Benedetti A, Henry T, Tucci-Piergiovanni S. A comparative gas cost analysis of proxy and diamond patterns in EVM blockchains for trusted smart contract engineering. In: Proc. of the 2024 Int'l Workshops on Financial Cryptography and Data Security. Willemstad: Springer, 2025. 207–221. [doi: 10.1007/978-3-031-69231-4_14]

[61] Asif R, Hassan SR. Shaping the future of Ethereum: Exploring energy consumption in proof-of-work and proof-of-stake consensus. Frontiers in Blockchain, 2023, 6: 1151724. [doi: 10.3389/fbloc.2023.1151724]

[62] Chen T, Li XQ, Luo XP, Zhang XS. Under-optimized smart contracts devour your money. In: Proc. of the 24th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Klagenfurt: IEEE, 2017. 442–446. [doi: 10.1109/SANER.2017.7884650]

[63] Krupa T, Ries M, Kotuliak I, Kostal K, Bencel R. Security issues of smart contracts in Ethereum platforms. In: Proc. of the 28th Conf. of Open Innovations Association (FRUCT). Moscow: IEEE, 2021. 208–214. [doi: 10.23919/FRUCT50888.2021.9347617]

[64] Marchesi L, Marchesi M, Destefanis G, Barabino G, Tigano D. Design patterns for gas optimization in Ethereum. In: Proc. of the 2020 IEEE Int'l Workshop on Blockchain Oriented Software Engineering (IWBOSE). London: IEEE, 2020. 9–15. [doi: 10.1109/IWBOSE50093.2020.9050163]

[65] Chen T, Feng YZ, Li ZH, Zhou H, Luo XP, Li XQ, Xiao XZ, Chen JC, Zhang XS. GasChecker: Scalable analysis for discovering gas-inefficient smart contracts. IEEE Trans. on Emerging Topics in Computing, 2020, 9(3): 1433–1448. [doi: 10.1109/TETC.2020.2979019]

[66] Hu B, Zhang ZY, Liu JW, Liu YZ, Yin JY, Lu RX, Lin XD. A comprehensive survey on smart contract construction and execution:

Paradigms, tools, and systems. Patterns, 2021, 2(2): 100179. [doi: 10.1016/j.patter.2020.100179]

[67] Kim B, Kim HJ, Lee J. First smart contract allowing cryptoasset recovery. KSII Trans. on Internet and Information Systems (TIIS), 2022, 16(3): 861–876. [doi: 10.3837/tiis.2022.03.006]

[68] Dai HN, Zheng ZB, Zhang Y. Blockchain for Internet of Things: A survey. IEEE Internet of Things Journal, 2019, 6(5): 8076–8094. [doi: 10.1109/JIOT.2019.2920987]

[69] Dorri A, Steger M, Kanhere SS, Jurdak R. Blockchain: A distributed solution to automotive security and privacy. IEEE Communications Magazine, 2017, 55(12): 119–125. [doi: 10.1109/MCOM.2017.1700879]

[70] Wang ZJ, Wang TY, Hu H, Gong J, Ren X, Xiao QY. Blockchain-based framework for improving supply chain traceability and information sharing in precast construction. Automation in Construction, 2020, 111: 103063. [doi: 10.1016/j.autcon.2019.103063]

[71] Zhang TY, Feng TT, Cui ML. Smart contract design and process optimization of carbon trading based on blockchain: The case of China's electric power sector. Journal of Cleaner Production, 2023, 397: 136509. [doi: 10.1016/j.jclepro.2023.136509]

[72] Leng JW, Sha WN, Lin ZS, Jing JB, Liu Q, Chen X. Blockchained smart contract pyramid-driven multi-agent autonomous process control for resilient individualised manufacturing towards industry 5.0. Int'l Journal of Production Research, 2023, 61(13): 4302–4321. [doi: 10.1080/00207543.2022.2089929]

[73] Lu WS, Li X, Xue F, Zhao R, Wu LPF, Yeh AGO. Exploring smart construction objects as blockchain oracles in construction supply chain management. Automation in Construction, 2021, 129: 103816. [doi: 10.1016/j.autcon.2021.103816]

[74] Feng TY, Yu X, Chai YT, Liu Y. Smart contract model for complex reality transaction. Int'l Journal of Crowd Science, 2019, 3(2): 184–197. [doi: 10.1108/IJCS-03-2019-0010]

[75] Nelaturu K, Mavridou A, Stachtiari E, Veneris A, Laszka A. Correct-by-design interacting smart contracts and a systematic approach for verifying ERC20 and ERC721 contracts with VeriSolid. IEEE Trans. on Dependable and Secure Computing, 2023, 20(4): 3110–3127. [doi: 10.1109/TDSC.2022.3200840]

[76] Sookhak M, Jabbarpour MR, Safa NS, Yu FR. Blockchain and smart contract for access control in healthcare: A survey, issues and challenges, and open issues. Journal of Network and Computer Applications, 2021, 178: 102950. [doi: 10.1016/j.jnca.2020.102950]

[77] Gilani K, Ghaffari F, Bertin E, Crespi N. Self-sovereign identity management framework using smart contracts. In: Proc. of the 2022 IEEE/IFIP Network Operations and Management Symp. Budapest: IEEE, 2022. 1–7. [doi: 10.1109/NOMS54207.2022.9789831]

[78] Chatterjee A, Pitroda Y, Parmar M. Dynamic role-based access control for decentralized applications. In: Proc. of the 3rd Int'l Conf. on Blockchain. Honolulu: Springer, 2020. 185–197. [doi: 10.1007/978-3-030-59638-5_13]

[79] Xue KP, Luo XY, Ma YJ, Li J, Liu JQ, Wei DSL. A distributed authentication scheme based on smart contract for roaming service in mobile vehicular networks. IEEE Trans. on Vehicular Technology, 2022, 71(5): 5284–5297. [doi: 10.1109/TVT.2022.3148303]

[80] Kim K, Ryu J, Lee H, Lee Y, Won D. Distributed and federated authentication schemes based on updatable smart contracts. Electronics, 2023, 12(5): 1217. [doi: 10.3390/electronics12051217]

[81] Yang YT, Lin TX, Liu PH, Zeng P, Xiao S. UCBIS: An improved consortium blockchain information system based on UBCCSP. Blockchain: Research and Applications, 2022, 3(2): 100064. [doi: 10.1016/j.bcra.2022.100064]

[82] Kumar R, Kumar P, Aloqaily M, Aljuhani A. Deep-learning-based blockchain for secure zero touch networks. IEEE Communications Magazine, 2023, 61(2): 96–102. [doi: 10.1109/MCOM.001.2200294]

[83] Chen JC, Xia X, Lo D, Grundy J, Yang XH. Maintenance-related concerns for post-deployed Ethereum smart contract development: Issues, techniques, and future challenges. Empirical Software Engineering, 2021, 26(6): 117. [doi: 10.1007/s10664-021-10018-0]

[84] Samreen NF, Alalfi MH. An empirical study on the complexity, security and maintainability of Ethereum-based decentralized applications (DApps). Blockchain: Research and Applications, 2023, 4(2): 100120. [doi: 10.1016/j.bcra.2022.100120]

[85] Rodler M, Li WT, Karame GO, Davi L. EVMPatch: Timely and automated patching of Ethereum smart contracts. In: Proc. of the 30th USENIX Security Symp. USENIX Association, 2021. 1289–1306.

[86] Zhang YY, Ma SQ, Li JR, Li KL, Nepal S, Gu DW. SMARTSHIELD: Automatic smart contract protection made easy. In: Proc. of the 27th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). London: IEEE, 2020. 23–34. [doi: 10.1109/SANER48275.2020.9054825]

[87] Ferreira Torres C, Jonker H, State R. Elysium: Context-aware bytecode-level patching to automatically heal vulnerable smart contracts. In: Proc. of the 25th Int'l Symp. on Research in Attacks, Intrusions and Defenses. Limassol: ACM, 2022. 115–128. [doi: 10.1145/3545948.3545975]

[88] Qin P, Tan WM, Guo JZ, Shen BQ. Intelligible description language contract (IDLC)—A novel smart contract model. Information Systems Frontiers, 2024, 26(5): 1597–1614. [doi: 10.1007/s10796-021-10138-4]

[89] Chen WM, Luo XP, Wang HY, Cui HM, Zheng SY, Liu XZ. EVMBT: A binary translation scheme for upgrading EVM smart contracts

to WASM. In: Proc. of the 25th ACM SIGPLAN/SIGBED Int'l Conf. on Languages, Compilers, and Tools for Embedded Systems. Copenhagen: ACM, 2024. 131–142. [doi: 10.1145/3652032.3657570]

[90] Chu HT, Zhang PC, Dong H, Xiao Y, Ji SH, Li WR. A survey on smart contract vulnerabilities: Data sources, detection and repair. Information and Software Technology, 2023, 159: 107221. [doi: 10.1016/j.infsof.2023.107221]

[91] Reijers W, Wuisman I, Mannan M, de Filippi P, Wray C, Rae-Looi V, Vélez AC, Orgad L. Now the code runs itself: On-chain and off-chain governance of blockchain technologies. Topoi, 2021, 40(4): 821–831. [doi: 10.1007/s11245-018-9626-5]

[92] OpenZeppelin. openzeppelin-contracts: OpenZeppelin contracts (formerly OZ-SDK)—A library for secure smart contract development. 2025. https://github.com/OpenZeppelin/openzeppelin-contracts

[93] Foundry. Foundry: Blazing fast, portable, modular toolkit for Ethereum development. 2025. https://getfoundry.sh/

[94] Di Angelo M, Salzer G. Characterizing types of smart contracts in the Ethereum landscape. In: Proc. of the 2020 Int'l Workshops on Financial Cryptography and Data Security. Kota Kinabalu: Springer, 2020. 389–404. [doi: 10.1007/978-3-030-54455-3_28]

[95] Shao W, Wang Z, Wang XL, Qiu KF, Jia CF, Jiang C. LSC: Online auto-update smart contracts for fortifying blockchain-based log systems. Information Sciences, 2020, 512: 506–517. [doi: 10.1016/j.ins.2019.09.073]

[96] Liu BW, Sun SW, Szalachowski P. SMACS: Smart contract access control service. In: Proc. of the 50th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN). Valencia: IEEE, 2020. 221–232. [doi: 10.1109/DSN48063.2020.00039]

[97] Liu YX, Hu DS, Jiang YM. Loose coupling model research for upgrading smart contracts already deployed on blockchain. Application Research of Computers, 2021, 38(5): 1309–1313 (in Chinese with English abstract). [doi: 10.19734/j.issn.1001-3695.2020.07.0160]

[98] Liu Y. Research on on-chain upgrade method of token smart contracts based on blockchain [MS. Thesis]. Chengdu: Sichuan University, 2021 (in Chinese with English abstract). [doi: 10.27342/d.cnki.gscdu.2021.000124]

[99] Du ZQ, Cheng H, Fu YF, Huang MH, Liu LX, Ma YF. A four-tier smart contract model with on-chain upgrade. Security and Communication Networks, 2023, 2023: 8455894. [doi: 10.1155/2023/8455894]

[100] Jin H, Wang ZL, Wen M, Dai WQ, Zhu Y, Zou DQ. Aroc: An automatic repair framework for on-chain smart contracts. IEEE Trans. on Software Engineering, 2022, 48(11): 4611–4629. [doi: 10.1109/TSE.2021.3123170]

[101] Zou WQ, Lo D, Kochhar PS, Le XBD, Xia X, Feng Y, Chen ZY, Xu BW. Smart contract development: Challenges and opportunities. IEEE Trans. on Software Engineering, 2021, 47(10): 2084–2106. [doi: 10.1109/TSE.2019.2942301]

[102] Xu XW, Pautasso C, Zhu LM, Lu QH, Weber I. A pattern collection for blockchain-based applications. In: Proc. of the 23rd European Conf. on Pattern Languages of Programs. Irsee: ACM, 2018. 3. [doi: 10.1145/3282308.3282312]

[103] Pierro GA, Mahugnon H. An analysis of the Oracles used in Ethereum's blockchain. In: Proc. of the 2023 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Macao: IEEE, 2023. 878–885. [doi: 10.1109/SANER56733.2023.00106]

[104] Chen JC. Finding Ethereum smart contracts security issues by comparing history versions. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM, 2020. 1382–1384. [doi: 10.1145/3324884.3418923]

[105] Abuhashim AA, Tan CC. Improving smart contract search by semantic and structural clustering for source codes. Blockchain: Research and Applications, 2023, 4(2): 100117. [doi: 10.1016/j.bcra.2022.100117]

[106] Khan ZA, Namin AS. Dynamic analysis for detection of self-destructive smart contracts. In: Proc. of the 47th IEEE Annual Computers, Software, and Applications Conf. (COMPSAC). Torino: IEEE, 2023. 1093–1100. [doi: 10.1109/COMPSAC57700.2023.00165]

[107] Chen JC, Xia X, Lo D, Grundy J. Why do smart contracts self-destruct? Investigating the selfdestruct function on Ethereum. ACM Trans. on Software Engineering and Methodology (TOSEM), 2021, 31(2): 30. [doi: 10.1145/3488245]

[108] Antonino P, Ferreira J, Sampaio A, Roscoe AW. Specification is law: Safe creation and upgrade of Ethereum smart contracts. In: Proc. of the 20th Int'l Conf. on Software Engineering and Formal Methods. Berlin: Springer, 2022. 227–243. [doi: 10.1007/978-3-031-17108-6_14]

[109] Casolari F, Taddeo M, Turillazzi A, Floridi L. How to improve smart contracts in the European union data act. Digital Society, 2023, 2(1): 9. [doi: 10.1007/s44206-023-00038-2]

[110] Vogelsteller F, Buterin V. ERC-20: Token standard. 2015. https://eips.ethereum.org/EIPS/eip-20

[111] Antonino P, Ferreira J, Sampaio A, Roscoe AW, Arruda F. A refinement-based approach to safe smart contract deployment and evolution. Software and Systems Modeling, 2024, 23(3): 657–693. [doi: 10.1007/s10270-023-01143-z]

[112] Dickerson T, Gazzillo P, Herlihy M, Saraph V, Koskinen E. Proof-carrying smart contracts. In: Proc. of the 2018 Int'l Workshops on Financial Cryptography and Data Security. Nieuwpoort: Springer, 2018. 325–338. [doi: 10.1007/978-3-662-58820-8_22]

[113] Necula GC. Proof-carrying code. In: Proc. of the 24th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. Paris: ACM, 1997. 106–119. [doi: 10.1145/263699.263712]

[114] Kondo M, Oliva GA, Jiang ZM, Hassan AE, Mizuno O. Code cloning in smart contracts: A case study on verified contracts from the

Ethereum blockchain platform. Empirical Software Engineering, 2020, 25(6): 4617–4675. [doi: 10.1007/s10664-020-09852-5]
[115] Ringer T, Porter RD, Yazdani N, Leo J, Grossman D. Proof repair across type equivalences. In: Proc. of the 42nd ACM SIGPLAN Int'l Conf. on Programming Language Design and Implementation. ACM, 2021. 112–127. [doi: 10.1145/3453483.3454033]
[116] Sorensen D. Towards formally specifying and verifying smart contract upgrades in Coq. Open Access Series in Informatics (OASIcs). 7:1–7:14. [doi: 10.4230/OASIcs.FMBC.2024.7]
[117] Sorensen D. WIP: Relational specification of smart contracts. 2022. https://derekhsorensen.com/docs/wip_relational_spec.pdf
[118] Galimullin R, Ågotnes T. Coalition logic for specification and verification of smart contract upgrades. In: Proc. of the 24th Int'l Conf. on Principles and Practice of Multi-agent Systems. Valencia: Springer, 2023. 563–572. [doi: 10.1007/978-3-031-21203-1_34]
[119] Porru S, Pinna A, Marchesi M, Tonelli R. Blockchain-oriented software engineering: Challenges and new directions. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering Companion (ICSE-C). Buenos Aires: IEEE, 2017. 169–171. [doi: 10.1109/ICSE-C.2017.142]
[120] Destefanis G, Marchesi M, Ortu M, Tonelli R, Bracciali A, Hierons R. Smart contracts vulnerabilities: A call for blockchain software engineering? In: Proc. of the 2018 Int'l Workshop on Blockchain Oriented Software Engineering (IWBOSE). Campobasso: IEEE, 2018. 19–25. [doi: 10.1109/IWBOSE.2018.8327567]
[121] Wöhrer M, Zdun U. DevOps for Ethereum blockchain smart contracts. In: Proc. of the 2021 IEEE Int'l Conf. on Blockchain (Blockchain). Melbourne: IEEE, 2021. 244–251. [doi: 10.1109/blockchain53845.2021.00040]
[122] Marchesi L, Marchesi M, Tonelli R. ABCDE—Agile block chain DApp engineering. Blockchain: Research and Applications, 2020, 1(1–2): 100002. [doi: 10.1016/j.bcra.2020.100002]
[123] Taherdoost H. Smart contracts in blockchain technology: A critical review. Information, 2023, 14(2): 117. [doi: 10.3390/info14020117]

附中文参考文献

[5] 崔展齐, 杨慧文, 陈翔, 王林章. 智能合约安全漏洞检测研究进展. 软件学报, 2024, 35(5): 2235–2267. http://www.jos.org.cn/1000-9825/7046.htm [doi: 10.13328/j.cnki.jos.007046]
[6] 董伟良, 刘哲, 刘逵, 黎立, 葛春鹏, 黄志球. 智能合约漏洞检测技术综述. 软件学报, 2024, 35(1): 38–62. http://www.jos.org.cn/1000-9825/6810.htm [doi: 10.13328/j.cnki.jos.006810]
[26] 欧易研究院. 以太坊 2.0 方案及进展研究报告. 2022. https://www.okx.com/zh-hans/learn/eth-2-0-scheme-and-progress
[31] 打破区块链的不可篡改, 代理模式如何以最佳方式实现智能合约升级? 2022. https://www.web3sj.com/news/31407/
[34] ChinaDeFi 去中心化金融社区. 深入了解智能合约的最小代理"EIP-1167". 2021. https://learnblockchain.cn/article/3400
[35] 以太坊 DApp 快速迁移教程. 2025. https://devdocs.platon.network/docs/zh-CN/DApp_migrate.
[37] 升级智能合约. 2023. https://ethereum.org/zh/developers/docs/smart-contracts/upgrading/
[48] 袁勇, 王飞跃. 可编辑区块链: 模型、技术与方法. 自动化学报, 2020, 46(5): 831–846. [doi: 10.16383/j.aas.2020.y000002]
[97] 刘云霞, 胡大裟, 蒋玉明. 面向智能合约链上升级的松耦合模型研究. 计算机应用研究, 2021, 38(5): 1309–1313. [doi: 10.19734/j.issn.1001-3695.2020.07.0160]
[98] 刘云霞. 基于区块链的通证智能合约链上升级方法研究 [硕士学位论文]. 成都: 四川大学, 2021. [doi: 10.27342/d.cnki.gscdu.2021.000124]

作者简介

郭涛, 博士生, 主要研究领域为区块链, 智能合约, 网络安全.
尚凤军, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为区块链, 新一代互联网, 物联网, 云计算, 大数据.
刘曼宇, 硕士生, 主要研究领域为区块链, 智能合约.
刘期烈, 博士, 教授, 博士生导师, 主要研究领域为网络通信, 移动大数据, 区块链.