Advancing Blockchain Ecosystems: Smart Contract Security and Scalability Examined

Ziqi Tang

Faculty of Information Technology, Monash University, Melbourne, VIC 3800, Australia

ztan0123@student.monash.edu

Abstract. Smart contracts, pivotal to blockchain technology, are rapidly proving their worth across the digital economy. Yet, as their application widens, issues concerning their security and scalability have emerged as major industry concerns. This paper provides a comprehensive analysis of these challenges, uncovering potential security vulnerabilities and scalability obstacles facing smart contracts. By delving into these critical areas, the study not only enhances understanding but also offers robust theoretical insights and practical recommendations to support the evolution and broader deployment of smart contracts. Furthermore, this research proposes innovative strategies and methods to address these pivotal issues, thereby facilitating the sustainable growth and widespread adoption of smart contracts in the digital economy. This work is expected to significantly influence the development trajectory of blockchain applications by providing new perspectives on improving smart contract frameworks in an increasingly digital world.

Keywords: smart contract, blockchain, security, scalability.

1 Introduction

Blockchain technology integrates advanced methodologies such as cryptography, distributed storage, consensus mechanisms, and smart contracts to forge a new form of quantifiable trust and incentive system [1]. This integration not only enhances transaction transparency and reduces trust risks but also diminishes costs and boosts efficiency, fundamentally transforming the socioeconomic mechanisms of value delivery and trust. As such, blockchain is recognized as a transformative core technology in this technological renaissance.

Central to the evolution of blockchain are smart contracts, which provide a reliable execution environment that revolutionizes traditional contractual processes. Unlike conventional agreements, smart contracts facilitate transparent, traceable, and irreversible transactions without the dependency on third parties [2]. This mechanism significantly enhances the security and transparency of contract execution, streamlines the process, and markedly reduces operational costs. The concept of smart contracts, first introduced by American computer scientist and cryptographer Nick Szabo in 1995, aimed to integrate these contracts into tangible entities, thus creating smart assets

Y. Wang (ed.), *Proceedings of the 2024 2nd International Conference on Image, Algorithms and Artificial Intelligence (ICIAAI 2024)*, Advances in Computer Science Research 115,
https://doi.org/10.2991/978-94-6463-540-9_37

characterized by flexibility and control [3]. Initially, the concept did not see widespread adoption due to the absence of a trusted execution environment. However, with the rapid advancement of blockchain technology, the definition and functionality of smart contracts have evolved. They are now seen as programmable, event-driven systems with state management capabilities, operating on a replicable and shared ledger that effectively manages ledger assets.

As smart contracts become increasingly prevalent, their security and scalability issues have come to the fore. Given their role in managing encrypted digital assets on blockchain platforms, smart contracts are economically valuable and thus a lucrative target for attacks, which can result in significant economic damages [4]. Furthermore, the intrinsic aim of smart contracts to utilize blockchain's trust-enhancing properties means that any vulnerabilities could transform a fair agreement into an "unequal contract," undermining the foundational principles of smart contracts. Moreover, scalability challenges restrict smart contracts' applications in high-volume transaction environments. As transaction volumes grow, the efficiency and processing capacity of smart contracts may degrade, leading to slower transaction speeds, increased costs, and potentially risking system collapse.

This paper delves into the pressing issues of security and scalability within smart contracts, offering critical insights and potential solutions. Through rigorous analysis, it seeks to inspire improvements and promote the robust development and extensive adoption of smart contract technology, thereby supporting the ongoing advancement of the blockchain field.

2 Basic analysis of smart contracts and blockchain

2.1 Blockchain Technology

The CAP theorem is an important theory in the field of distributed systems. It points out that a distributed system cannot satisfy the three characteristics of Consistency, Availability and Partition tolerance at the same time [5]. The current blockchain technology mainly uses the consensus mechanism based on Proof of Work (PoW) proposed by Satoshi Nakamoto and the decentralization feature to achieve a balance of these three features to a certain extent, ensuring the authenticity and authenticity of the data on the blockchain. Credibility provides a strong guarantee for the execution of smart contracts to maintain high performance and stability when processing large-scale, highly concurrent transactions.

2.2 Security Principles

Designing a complete smart contract involves multiple aspects such as privacy issues, legal issues, security issues, and mechanism design [6]. Therefore, it is not easy to design a smart contract that is free of security vulnerabilities, fair and trustworthy, compliant with regulations, and easy to follow the transaction process. The Fig. 1. below simulates the scenario where users deposit funds on the exchange. And the

security principles requiring attention in the smart contract based on this scenario will be listed.

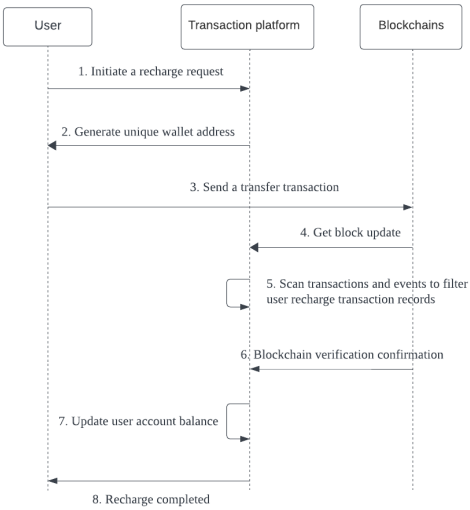

Fig. 1. Simulate User Recharging Flowchart .

Based on the above simulation scenario, designers need to verify the validity of the input parameters of the smart contract to prevent vulnerabilities and attacks caused by malicious input by attackers. When designing the permission control mechanism, restrict access and operations to the trading platform contract to ensure that only authorized users can perform specific operations. When designing blockchain verification, each state change in the contract should be completed after external calls to avoid the loss of funds caused by double-spending attacks [7]. Finally, the contract mechanism should be updated in a timely manner, and the upgrade and update mechanism of the contract should be considered to deal with the discovery of loopholes or the need for improvement, while ensuring that the contract update process is safe and reliable.

2.3 Market Requirement

The market demand for smart contracts is growing rapidly, especially in areas such as supply chain management, e-commerce, and healthcare. As requirements for supply chain transparency and security increase, blockchain technology is providing impetus to market growth by increasing automation, eliminating middlemen, and streamlining processes. In the context of the booming e-commerce industry, smart contracts have received increasing attention as a tool to improve transaction transparency and trust.

Furthermore, the demand for drug tracking and control in the healthcare field has further promoted the application of smart contracts [8]. Although the Covid-19 epidemic has previously caused some impact on the supply chain, the increasing demand for e-commerce, coupled with the promotion of AI and machine learning, has hastened the embrace of blockchain-based supply chain solutions [9].

With the continuous development of blockchain technology and the further improvement of smart contract functions, smart contracts are expected to play a role in a wider range of industries by improving transaction efficiency and reducing costs, thereby promoting wider market adoption of the technology [10]. Therefore, people are increasingly focus on smart contracts' security and scalability issues, which are crucial to maintaining the long-term stability and sustainable development of smart contracts.

3 Smart Contract Security Research Analysis and Countermeasures

Smart contract is the core components of the blockchain and is divided into the application layer (as shown in Fig. 2. below) [11]. Compared with ordinary programs, smart contracts have many unique characteristics. These characteristics make smart contracts vulnerable to unusual attack methods by attackers and trigger new security threats.

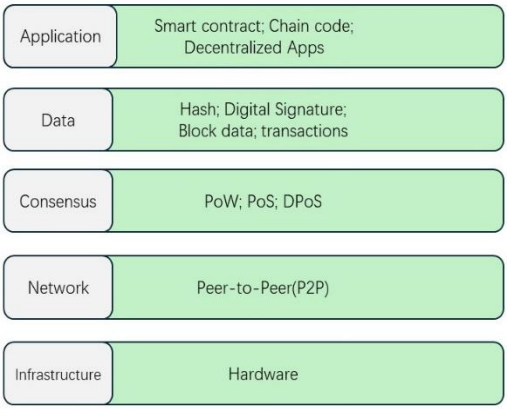

Fig. 2. Blockchain five-layer structure.

The primary security risks associated with smart contracts stem from various security vulnerabilities. These vulnerabilities primarily arise from three distinct levels: high-level language, virtual machine, and blockchain [12]. This section will delve into the main security weaknesses present at each of these levels and discuss the security incidents that have resulted from these vulnerabilities.

Smart contracts are encoded in high-level languages, then compiled into bytecode, triggered by blockchain transactions, and executed on a virtual machine with blockchain as the storage basis. The entire process faces various security threats, alongside the discovery of security vulnerabilities.

3.1 Security Analysis from High-level Languages Level

A diverse range of high-level languages can be utilized for the development of smart contracts, with Solidity being one of the most popular choices. There are two main reasons why smart contracts pose security threats in this regard: One is the security

issues introduced by the flaws in the design of high-level languages. The other is security vulnerabilities caused by poor code quality when developers coding high-level languages.

Variable Coverage. The variable coverage leak is a common security weakness stemming from design flaws in high-level languages. Certain contracts designed with specific versions of Solidity will be affected by this vulnerability [13].

When developing smart contracts using Solidity, variables within functions typically default to Memory type, effective only during function calls and recycled upon return. However, in specific versions of Solidity, array or struct variables declared by default may be erroneously treated as Storage type by the compiler. Manipulating these variables can lead to unauthorized access to the contract's Storage area.

Given the high gas cost associated with accessing and manipulating Storage variables, they are typically reserved for storing crucial information like administrator addresses and balances. Any accidental overwrite of these variables can lead to contract malfunctions, posing data and economic security risks.

Integer Overflow. Integer overflow is a universal characteristic of computer programming languages. Its occurrence primarily stems from the fact that programming languages have inherent limits on the storage space designated for integer types, resulting in restrictions on the range of integers that can be represented. Whenever the outcome of an integer operation surpasses this designated range, an integer overflow ensues.

This issue primarily arises from users' failure to conduct secure overflow checks on operation results. Unexpected integer overflow can lead to operational errors in smart contract programs and potentially be exploited by attackers to bypass checks and tamper with crucial data, such as balances.

In the Blockchain ecosystem, BEC and SMT were two equity crowdfunding contracts that complied with the ERC20 standard [14]. However, in April 2018, two contracts were attacked by integer overflow vulnerabilities. Attackers exploited these vulnerabilities and successfully created huge amounts of Tokens up to 2^{256} in magnitude, which were then dumped on exchanges. This behaviour led to a sharp decline in the market value of these two contracts, ultimately resulting in their values falling to zero. This incident highlighted the serious threat of integer overflow vulnerabilities to the security of smart contracts, reminding developers and users of the importance of conducting thorough security checks in smart contracts to avoid similar economic losses and security risks.

Denial of Service. Denial of service in the context of smart contracts refers to the failure of the contract to fulfill its intended response function as designed. There are numerous reasons that may lead to denial of service in smart contracts. Common causes include triggering unexpected exceptions, reaching the block gas limit, unexpected self-destruct, and hardcoding incorrect administrator addresses. These reasons are typically introduced due to unsafe coding practices employed during the development of contract code.

Denial of service vulnerabilities in smart contracts come in various forms. Depending on the impacted entity, denial of service that causes blockage of the blockchain network is typically classified as active denial of service, whereas denial of service resulting from errors introduced by contract developers that prevent the contract from operating normally is referred to as passive denial of service.

KotET is a multiplayer game contract that allows victorious players to purchase the “Throne” with virtual currencies, thus taking over the position from the current "king" player. However, this contract contains a denial-of-service vulnerability. In January 2019, attackers exploited this vulnerability and elevated their maliciously crafted contract account to the position of "king". This contract included a complex callback function, resulting in the failure of any contract calls attempting to transfer funds to it. This active denial-of-service attack prevented other KotET players from purchasing the throne, ultimately allowing the attackers to become the permanent "king".

3.2 Security Analysis from Virtual Machine Level

The virtual machine level is susceptible to two primary security threats. Firstly, there are some inherent flaws in the smart contract bytecode specification and operating mechanism outlined in the Ethereum Yellow Paper. The second aspect is the issue arising from different blockchain clients failing to strictly adhere to the manual specifications and thereby introducing external functionalities in a hasty manner during the implementation of virtual machines. In this part of the paper, typical attacks and cases are mainly analysed and discussed.

Reentrancy vulnerability. The reentrancy vulnerability arises when a contract performs the operations of transferring funds and modifying storage variables in a non-atomic order, with the transfer occurring before the storage update. Owing to the execution model of Ethereum's EVM (Ethereum Virtual Machine), when contract A transfers funds to contract B and contract B lacks a designated function to receive these funds, the fallback function of contract B is automatically invoked. If the fallback function is malicious, it can recursively call the function of contract A, leading to repeated receipt of transferred funds and subsequent financial losses.

In April 2016, some blockchain developers utilized the Ethereum platform to establish a decentralized autonomous organization named the DAO. The organization functions as a decentralized venture capital fund that raises and utilizes funds for project investment through crowdfunding via smart contracts on Ethereum. All crowdfunding participants are allocated voting rights based on their respective investment shares to vote on investment projects.

However, in June 2016, hackers discovered and exploited the reentrancy vulnerability in The DAO's smart contract, launching an attack (as illustrated in Fig. 3. below). This attack resulted in a loss of over $50 million for the organization. With the emergence of The DAO attack, smart contract security has gradually garnered increasing attention.

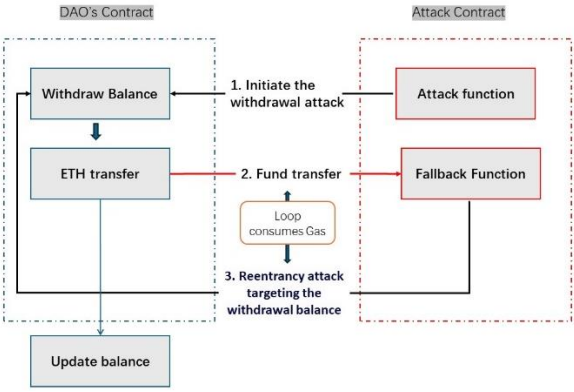

Fig. 3. The DAO attack simulation diagram.

3.3 Security Analysis from Blockchain Level

Smart contracts leverage blockchain technology to offer decentralization, immutability, and trust. The blockchain platform significantly influences the operation of these contracts. While blockchain is the backbone of smart contract security and trust, its inherent features can also be attacked by security vulnerabilities.

Timestamp Dependency. Timestamp dependency is a common security risk in smart contracts. The block timestamp represents the time when a specific transaction is included in the blockchain. Although this timestamp appears to be randomly generated based on time, it can be manipulated by miners within a certain value range. If the design of a smart contract heavily relies on precise timestamps as decision-making parameters, it may face potential security threats. For attackers posing as miners, they can affect the execution of the contract by selectively delaying transaction confirmations or constructing blocks with specific timestamps.

This ability allows attackers to circumvent the limitations designed in the contract based on timestamps, which may lead to the destruction of the contract's logic and even result in a loss of funds.

Race Condition. The race condition vulnerability in smart contracts arises from their reliance on transaction order as the sole decision-making criterion. Smart contracts are invoked through transactions, but the timing of transaction initiation and its confirmation, which marks the effectiveness of the contract invocation, are not directly linked.

Illustratively, a contract may offer a reward to the first account submitting the correct answer. However, the first submitter may not always receive the reward. Because after initiating the transaction, it can be observed by network nodes, but there's a delay before it's packaged. Since miners prioritize transactions with higher fees, attackers can rapidly initiate similar transactions and secure priority by increasing fees. The source of this vulnerability lies in the transaction packaging and fee mechanism of the blockchain, which decouples the transaction confirmation sequence from the initiation sequence.

The Approve function in the well-known ERC20 contract standard used to have this vulnerability.

3.4 Security defence measures

Security defence is a crucial aspect of smart contract security research and a vital step in countering the attacks described above. Smart contracts are immutable once deployed, making it impractical to directly patch vulnerable programs, rendering their security defence efforts significantly more challenging compared to other programs. Security defence solutions for smart contracts are mainly divided into three directions: secure programming practices for smart contracts, hot upgrades for smart contracts.

Secure Programming. In current research efforts, the primary approach to enhancing the security of Ethereum smart contract programming involves equipping the Solidity programming language with secure and convenient third-party libraries. OpenZeppelin provides many standardized Solidity code libraries that have undergone strict security audits. These libraries encompass a wide range of functionalities, including ERC standard tokens, administrator permission access control, encryption and decryption, secure arithmetic operations, and more [15]. Their primary objective is to assist developers in swiftly constructing secure and dependable smart contract applications. During contract development, developers can simply inherit or import these code libraries to use the corresponding library functions for smart contract development.

Hot Upgrades. Whether it is providing safer contract libraries or using more secure high-level languages, secure programming solutions for smart contracts require eliminating all vulnerabilities in the contracts before their release to ensure the safety of the deployed contracts. However, new vulnerabilities in smart contracts continue to emerge. Despite efforts to eliminate all known vulnerabilities prior to deployment, it is still helpless for new vulnerabilities discovered after deployment. Researchers have proposed the concept of a "proxy contract" to achieve indirect hot upgrading of contracts and fix contract vulnerabilities.

The two most important parts of a smart contract are its storage and code logic. Based on this situation, the OpenZeppelin organization utilizes the proxy contract mechanism to divide contract deployment into two steps [16]. It uses a relatively secure and concise proxy contract as the entry point for the smart contract. The proxy contract manages the storage of the contract and calls the contract code in the logic contract through delegated calls. Once a vulnerability is discovered in the code implementing the main logic of the contract, developers can fix the vulnerability and redeploy a new logic contract. They can then point the logic contract address in the proxy contract to this new contract, thus achieving hot upgrading of the contract.

This hot upgrading solution can effectively mitigate the response capabilities of contracts to newly discovered vulnerabilities after deployment. However, as it alters the immutable nature of smart contracts once deployed, malicious developers may change contract rules or introduce unequal contract terms during the upgrade process,

compromising the tamper-resistance and reducing the trustworthiness of the contract. Therefore, this direction still deserves further research.

4 Smart Contract Scalability Challenges and Solutions

Cryptocurrencies and blockchain are booming, attracting a surge in users and transactions. Despite blockchain's revolutionary potential, scaling up systems to meet this demand while maintaining decentralization and security remains challenging. Highly decentralized and secure blockchain networks often struggle with high throughput, known as the blockchain trilemma. It's difficult for a decentralized system to balance those characteristics. To tackle this, various scaling solutions are being explored. Some aim to modify the main blockchain's architecture, and others focus on building Layer 2 protocols on the top of underlying network.

This part will analyze and discuss the current mainstream Ethereum expansion solutions from two aspects: On-Chain Scaling (Layer-1) and Off-Chain Scaling (Layer-2).

4.1 On-Chain Scaling

On-Chain Scaling or called Layer-1 scaling. Typically, on-chain scaling refers to solutions that occur directly on the blockchain by changing the block size or data structure to improve throughput.

One on-chain scaling technique aims to minimize the block size while maintaining the transaction count per block, thereby boosting overall throughput. A prime example is Segregated Witness (Segwit). By excluding signatures from transaction data and incorporating them into metadata with a separate script, Segwit frees up valuable block space since signatures account for about 65% of transaction data. This extra space enables increased transactions within each block [17]. Consequently, this approach can increase the transaction volume within a block by approximately fourfold, subsequently boosting throughput. Additionally, Segwit also enlarges block size to 4 MB, resolves the secondary hashing issue, and expedites payment channels (e.g. the Lightning Network). Nevertheless, despite Segwit's capability to facilitate soft scaling for Bitcoin, its effectiveness remains constrained, limiting the throughput improvement to a range of 17 to 23 transactions per second (TPS).

The second approach to on-chain scaling involves directly enlarging the block size. This method is employed by two schemes: Bitcoin-cash and Bitcoin-unlimited [18]. Nevertheless, this solution gives rise to increased propagation delays, potentially resulting in fork issues and DoS attacks, thereby posing significant security challenges. Given the constraints of Segwit, some miners have opted for Bitcoin-Cashing as an alternative. Bitcoin-Cashing initially escalated the Bitcoin block size to 8 MB and subsequently to 32 MB. However, it's worth noting that the block size does not have a direct linear correlation with throughput, hence the method's limited effectiveness in significantly enhancing throughput.

Another approach to on-chain scaling is known as sharding, primarily employed in distributed processing systems. In this method, data is partitioned and stored as

individual shards for processing. Each shard concurrently handles transactions and data storage, thereby enhancing overall throughput. Additionally, sharding technology mitigates communication overhead within the BFT consensus network. However, shard allocation, security concerns surrounding small shards, and cross-shard communication overhead pose significant challenges to the effectiveness of sharding technology. Inadequacies in these areas can potentially lead to severe security vulnerabilities.

4.2 Off-Chain Scaling

Off-Chain Scaling refers to any innovative way to provide external execution for underlying chains such as Ethereum. These innovations are called Layer-2 scaling solutions, which involve executing transactions on a second-layer network outside of Ethereum, thereby enhancing the performance of Ethereum's Layer-1.

State Channel. The state channel is an off-chain scaling solution grounded in multi-signature smart contracts, significantly broadening the functionality of payment channels. Except basic payment capabilities, state channels facilitate state updates on the blockchain, effectively altering the internal state of smart contracts. In a state channel network, crypto-assets like ETH can be locked within these contracts, enabling the creation of two-side payment channels among users.

Users of state channels are not confined to payment transactions but can execute smart contracts off-chain as well. The operation model of the state channel is shown in Fig. 4. Take users Alice and Bob establishing a state channel as an example; they gain access to a "simulated contract ledger" where they can execute contract operations without the need for registration on the main blockchain. As long as both parties maintain consensus, the solution can be implemented seamlessly. The security of state channels stems from the ability for both parties to "confirm" the current off-chain state of the channel on the main blockchain at any given time, ensuring fair execution of contract terms by the channel contract. State channels find diverse applications, encompassing digital content distribution, online gaming, and decentralized token exchanges [19].

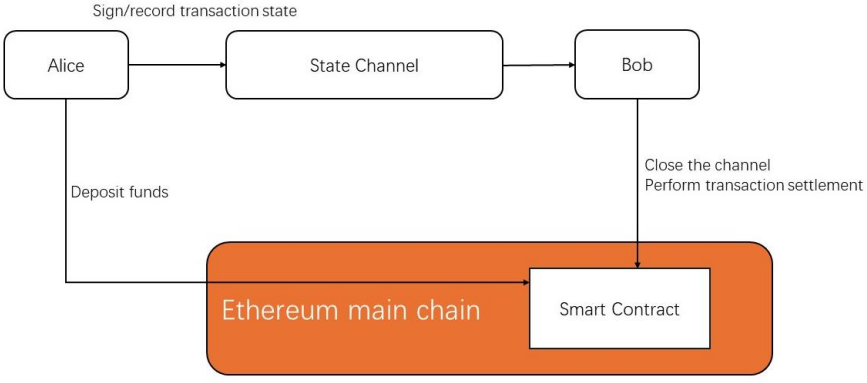

Fig. 4. State channel operating model.

Plasma. Plasma is a framework designed to facilitate the construction of scalable applications on the Ethereum platform. Its primary objective is to enhance the throughput and scalability of the Ethereum network by introducing Ethereum-based sidechains [20]. Plasma addresses the limitations posed by state channels by enabling the establishment of additional sub-chains on the Ethereum main chain, which in turn can give birth to their own subordinate chains. At the subchain level, numerous intricate operations can be executed, facilitating the running of entire applications catering to thousands of users, while minimizing interaction with the main Ethereum chain. These child chains offer faster operations and reduced transaction fees, as there's no requirement to maintain a replica of the entire Ethereum blockchain. A key distinction between Plasma and state channels is that Plasma facilitates the execution of smart contracts. If state channels represent the scaling of transaction throughput, then Plasma signifies the scaling of computing capabilities. Plasma relocates both computational tasks and data storage to Layer-2, with Layer-2 executors periodically submitting state commitments in the form of Merkle roots to the main chain.

Fig. 5. presents a simplified diagram of Plasma's operation: if an executor submits an invalid state, the user can provide a fraud proof to the smart contract on the main chain. Once it is confirmed that the executor has committed fraud, the smart contract will forfeit their deposit. Although fraud proofs can punish executors who make invalid commitments on the main chain, if Plasma executors refuse to disclose data on the main chain, users will be unable to obtain incorrect data and therefore cannot provide fraud proofs. Consequently, the current major challenge facing Plasma lies in the availability of transaction data. To address this issue, Plasma has proposed several solutions, such as extending the time frame for assets to withdraw from Layer-2. This allows a significant amount of assets to be withdrawn from the Plasma chain in the event of malicious behaviour.

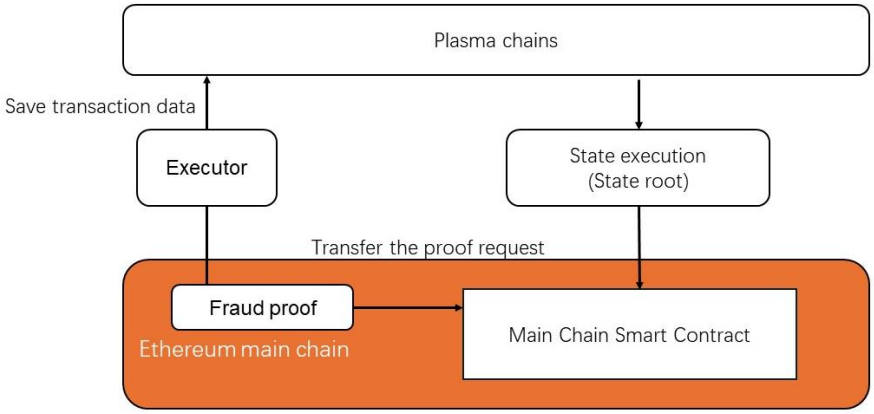

Fig. 5. Plasma chains' operating model.

4.3 Comparison of Scaling solutions

Table 1. Scaling solutions.

Layer-type	Approach	Throughput	Cost	Capacity
On-Chain Scaling				
	Segwit	↑	↓	↑
	enlarge block size	↑	↓	↑
	sharding	↑	---	↓
Off-Chain Scaling				
	State Channel	↑	↓	↓
	Plasma	↑	---	↓

It can be obviously seen from Table 1, different scaling technologies have their own advantages, disadvantages, and usage scenarios. For on-chain scaling solutions, Segwit improves throughput by separating signatures, but it increases code complexity and has low portability; directly increasing the block size reduces costs, but it also increases the possibility of orphan blocks; Sharding can process many transactions in parallel, but the possibility of being attacked also increases accordingly.

In terms of Layer-2 scaling solutions, state channels occupy an irreplaceable position in the scaling field due to their unique nature. Compared with state channels, Plasma has the advantage of being able to run smart contracts.

4.4 Future Developments in Scalability

There are currently many scaling solutions that have achieved certain results in weighing the issues of blockchain in scalability, decentralization, and security. However, blockchain's scalability challenges still haven't been fully solved, meaning blockchain still hasn't reached its full potential. Currently, blockchain scalability has the following challenges:

Sharding, a popular scalability enhancement method, boasts impressive throughput, scalable storage, minimal latency, and robust Byzantine fault tolerance. The communication cost of individual transactions lies at the heart of blockchain scalability concerns. It is only when the complexity of sorting transactions is reduced to O(n) that the blockchain can be deemed scalable. In the future, in order to obtain more bandwidth for data availability, research can be conducted from aspects such as load balancing and cost-benefit expenditure balancing.

Maintaining atomicity in cross-shard transactions poses another scalability challenge. When multiple shards operate simultaneously, a timeline is necessary to authenticate the order of these operations. In scenarios where shards need to handle a significant volume of legitimate or illegitimate cross-shard transactions, a load balancing mechanism can be employed to address issues such as miner node exhaustion and DoS attacks.

Blocks are prone to form multiple branches, which are called forks and will also affect scalability. Currently the longest chain rule is utilized to address forks. Without

a mechanism to prevent forks, resources remain wasted. To solve this problem, a fork monitoring committee composed of incentive nodes can be established, which is separated from the mining nodes, and allocates mining work according to the capabilities of the mining nodes, and monitors in real time and avoids forks in time.

5 Conclusion

In summary, the focal areas of contemporary research on blockchain technology pertain to the security, scalability, and associated risks of smart contracts. This paper provides a comprehensive analysis of the security threats to smart contracts, incorporating detailed case studies and a review of the advancements in security defense mechanisms. In addition, this study examines dual-layer scalability solutions, enhancing the understanding of current methodologies and technologies. The paper aggregates and synthesizes current research findings on smart contracts, offering a forward-looking perspective on potential developments in this domain. It articulates strategic directions for bolstering the security framework of mainstream smart contracts and enhancing their scalability. By systematically addressing these critical areas, the paper contributes significantly to the discourse on smart contracts, proposing robust solutions that could lead to more secure, scalable, and efficient blockchain implementations. This approach not only aims to mitigate the inherent risks but also strives to unlock the full potential of smart contracts within the broader digital economy.

References

1. Sherman, A. T., Javani, F., Zhang, H., Golaszewski, E.: On the Origins and Variations of Blockchain Technologies. IEEE Security & Privacy 17(1), 72–77 (2019).
2. Röscheisen, M., Baldonado, M., Chang, K., Gravano, L., Ketchpel, S., Paepcke, A.: The Stanford InfoBus and its service layers: Augmenting the internet with higher-level information management protocols. Digital Libraries in Computer Science: The MeDoc Approach, 213–230 (1998).
3. Fries, M., Paal, B.: Smart Contracts. Mohr Siebeck GmbH & Co. KG, Tuebingen (2019).
4. Khan, S.N., Loukil, F., Ghedira-Guegan, C., et al.: Blockchain smart contracts: Applications, challenges, and future trends. Peer-To-Peer Networking and Applications 14(1), 2901–2925 (2021).
5. Dongfang, J., Wang, L.: Research on smart contract technology based on block chain. 2022 International Conference on Artificial Intelligence in Everything (AIE), Lefkosa, Cyprus, 664-668 (2022).
6. Sifra, E. M.: Security Vulnerabilities and Countermeasures of Smart Contracts: A Survey. 2022 IEEE International Conference on Blockchain (Blockchain), Espoo, Finland, 512-515 (2022).
7. Consensys: Ethereum Smart Contract Security Best Practices. (2022).
8. Poleshchuk, E. M., Shcherbinina, I. A., Putilova, S. E.: Security Analysis of Smart Contracts in Blockchain Networks. 2022 Ural-Siberian Conference on Biomedical

Engineering, Radioelectronics and Information Technology (USBEREIT), Yekaterinburg, Russian Federation, 252-254 (2022).
9. Han, X., Yuan, Y., Wang, F.: Security Problems on Blockchain: The State of the Art and Future Trends. ACTA AUTOMATICA SINICA 45(1), 206-225 (2019).
10. MordorIntelligence: Blockchain Supply Chain Market Size & Share Analysis - Growth Trends & Forecasts (2024 - 2029). (n.d.).
11. Chen, H., Pendleton, M., Njilla, L., Xu, S.: A Survey on Ethereum Systems Security. ACM Computing Surveys 53(3), 1–43 (2020).
12. Zhu, X., Huang, Y., Wang, X., Wang, R.: Emotion recognition based on brain-like multimodal hierarchical perception. Multimedia Tools and Applications 1-19 (2023).
13. Atzei, N., Bartoletti, M., Cimoli, T.: A Survey of Attacks on Ethereum Smart Contracts (SoK). Lecture Notes in Computer Science, 164–186 (2017).
14. Rahimian, R., Eskandari, S., Clark, J.: Resolving the Multiple Withdrawal Attack on ERC20 Tokens. 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), 320-329 (2019).
15. OpenZeppelin: Contracts-OpenZeppelin Docs. (2024).
16. Al Amri, Shaima, Aniello, L., Sassone, V.: A Review of Upgradeable Smart Contract Patterns based on OpenZeppelin Technique. 6(1), 1–8 (2023).
17. Sanka, A. I., Cheung, R. C. C.: A systematic review of blockchain scalability: Issues, solutions, analysis and future research. Journal of Network and Computer Applications 195, 103232 (2021).
18. Wang, Z., Chaliasos, S., Qin, K., Zhou, L., Gao, L., Berrang, P., Livshits, B., Gervais, A.: On How Zero-Knowledge Proof Blockchain Mixers Improve, and Worsen User Privacy. ArXiv.org (2023).
19. Dziembowski, S., Faust, S., Hostáková, K.: General State Channel Networks. ACM SIGSAC 2018, 949–966 (2018).
20. Bez, M., Fornari, G., Vardanega, T.: The scalability challenge of ethereum: An initial quantitative analysis. 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), 167-176 (2019).