Implementation and Security Analysis of Cryptocurrencies Based on Ethereum

PENGFEI GAO* and DECHAO KONG*, Hainan University, China
XIAOQI LI, Hainan University, China

Blockchain technology has set off a wave of decentralization in the world since its birth. The trust system constructed by blockchain technology based on cryptography algorithm and computing power provides a practical and powerful solution to solve the trust problem in human society. In order to make more convenient use of the characteristics of blockchain and build applications on it, smart contracts appear. By defining some trigger automatic execution contracts, the application space of blockchain is expanded and the foundation for the rapid development of blockchain is laid. This is blockchain 2.0. However, the programmability of smart contracts also introduces vulnerabilities. In order to cope with the insufficient security guarantee of high-value application networks running on blockchain 2.0 and smart contracts, this article will be represented by Ethereum to introduce the technical details of understanding blockchain 2.0 and the operation principle of contract virtual machines, and explain how cryptocurrencies based on blockchain 2.0 are constructed and operated. The common security problems and solutions are also discussed. Based on relevant research and on-chain practice, this paper provides a complete and comprehensive perspective to understanding cryptocurrency technology based on blockchain 2.0 and provides a reference for building more secure cryptocurrency contracts.

Additional Key Words and Phrases: Blockchain 2.0, Smart Contracts, Cybersecurity, Cryptocurrency

1 INTRODUCTION

Blockchain is a specialized form of distributed data storage that was first introduced as the underlying technology of Bitcoin in the paper Bitcoin: A Peer-to-Peer Electronic Cash System, published in 2008 by an individual or group under the pseudonym Satoshi Nakamoto during the subprime mortgage crisis. This technology pioneered a novel solution to the trust problem in distributed ledger storage through the combination of hash chaining and the proof-of-work mechanism. Due to its characteristics of data transparency, decentralization, and immutability, blockchain has been widely adopted in decentralized digital currency issuance and payment systems. The decentralization of digital currencies initiated and exemplified by Bitcoin is referred to as Blockchain 1.0[26, 37]. To enable blockchain to support more complex applications, programmable smart contracts were first introduced on top of the blockchain ledger structure, allowing Turing-complete programs to run on-chain. This advancement facilitated the development and execution of more sophisticated applications directly on blockchain platforms, significantly expanding its application scope and laying the foundation for its rapid evolution—this stage is known as Blockchain 2.0[1].

The most representative example of Blockchain 2.0 is Ethereum, which introduced a novel mechanism for token crowdfunding, commonly known as Initial Coin Offerings (ICOs). Developers can effortlessly create their own tokens on Ethereum via smart contracts—so-called programmable tokens—where the on-chain services provided by contracts serve as a fundamental value anchor for these tokens. Under this paradigm, the integration of smart contracts with digital assets has fostered a dynamic, decentralized, and multi-layered financial ecosystem known as Decentralized Finance (DeFi)[48, 51]. As illustrated in Figure 1, taking the Ethereum ecosystem as an example, this ecosystem is structured around Ethereum's native currency, ETH, as Layer 0. With Ethereum 2.0 supporting ETH staking, a bond market emerges, where interest rates regulate the flow of ETH throughout the ecosystem. Built upon this market, Layer 1 establishes a stability layer, ensuring stable value. MakerDAO, for instance, employs Collateralized Debt Position (CDP) contracts to lock ETH and generate DAI, a stablecoin pegged

*Both authors contributed equally to this research.

Authors' addresses: PENGFEI GAO; DECHAO KONG, Hainan University, Haikou, China, 3037317181@gmail.com; XIAOQI LI, Hainan University, Haikou, China, csxqli@ieee.org.

to the U.S. dollar[23]. Consequently, Layer 1 also functions as a capital formation layer, enabling individual users to participate in token minting. Moving up, Layer 2 represents the capital utility layer, where DAI-based lending mechanisms regulate borrowing costs through interest rate balancing. These tokens subsequently flow into the application layer, providing liquidity for various decentralized applications. At this level, atomic financial services such as token exchanges (Uniswap), prediction markets (Augur), and derivatives trading platforms (dYdX) operate seamlessly. Ultimately, the ecosystem culminates in a user aggregation layer, facilitating cross-chain transactions, credit card integrations, real estate exchanges, and other financial services.Even within the realm of decentralized finance alone, Blockchain 2.0 has demonstrated immense vitality, giving rise to a sophisticated value-driven internet. At the heart of this thriving value network lies smart contracts, which serve as the foundational infrastructure. Programmable tokens have unlocked boundless possibilities for cryptocurrencies, cementing blockchain's role as a transformative force in the digital economy.

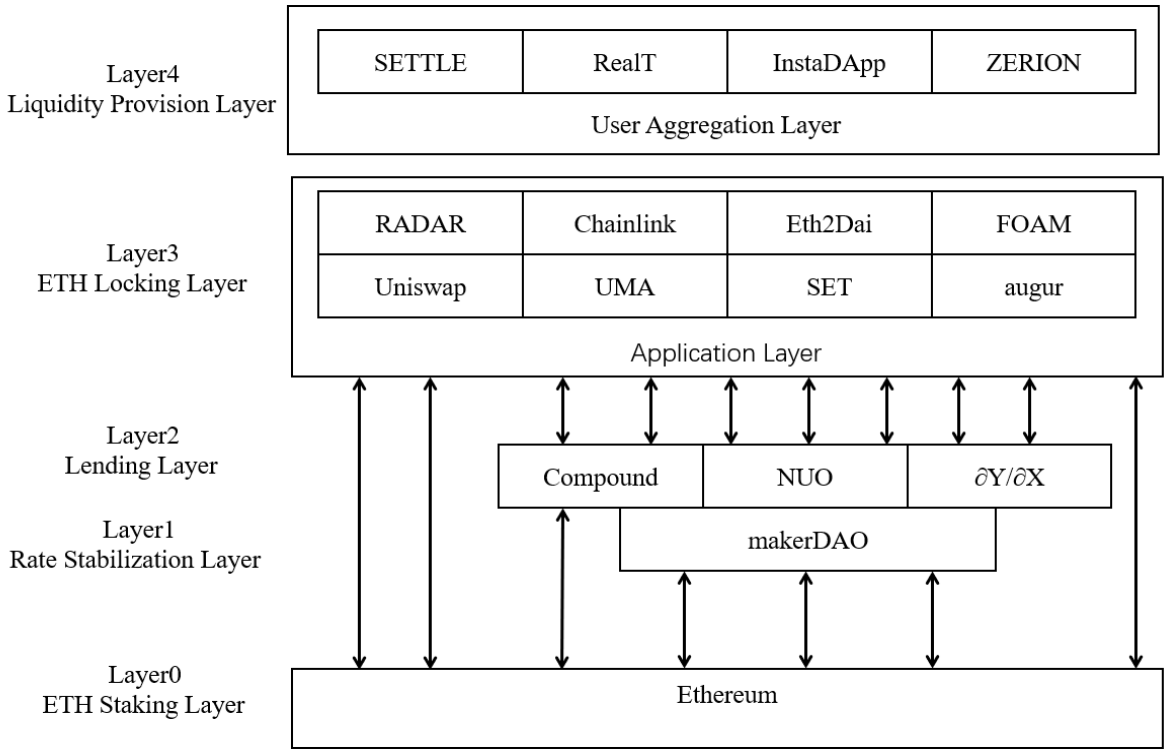

Fig. 1. Layered Architecture of the Ethereum Ecosystem.

As the Value Internet continues to expand, an increasing amount of capital has flowed into tokenized assets, leading to an exponential surge in token valuations over recent years. Meanwhile, the security concerns associated with smart contracts have become increasingly critical. On-chain token contracts are immutable, meaning that once deployed, all associated transactions are irrevocably determined. While this immutability ensures fairness and transparency, it also introduces significant risks[5, 11]. If vulnerabilities exist in the smart contract underpinning a token, any security breach could result in substantial digital asset losses, with no way to reverse or amend the deployed contract[27, 35]. In June 2016, the large-scale The DAO project, which raised $150 million through an

ICO within a month, was found to have a reentrancy vulnerability in its smart contract[34]. Exploiting this flaw, hackers drained $60 million worth of Ether, causing a sharp drop in Ethereum's market price and ultimately leading to a hard fork of the Ethereum blockchain. In August 2016, the major exchange Bitfinex was attacked, resulting in the theft of 119,756 Bitcoin, valued at approximately $65 million at the time.In July 2017, the widely used Parity Ethereum wallet was compromised, leading to the theft of 150,000 Ether, worth $30 million. Later that year, in November, another vulnerability in the Parity wallet resulted in 513,701 Ether being permanently locked. In April 2018, the BEC and SMT token contracts fell victim to an integer overflow attack, allowing hackers to mint and dump massive amounts of tokens, effectively reducing their value to near zero.In April 2020, the Lendf.Me lending protocol was exploited due to reentrancy issues and security flaws in its unique token composition, leading to a total depletion of assets from the contract, with losses amounting to $25 million.

In recent years, significant progress has been made in blockchain and smart contract security research both domestically and internationally[13, 22]. Security-compliant audits and formal verifications have substantially reduced the occurrence of major financial incidents on blockchain networks. However, security breaches leading to economic losses remain frequent. Analyzing various attacks on smart contracts reveals that while the underlying blockchain technology is inherently robust and rarely encounters critical failures, vulnerabilities are prevalent at the smart contract layer. These issues primarily stem from two factors: inherent logical flaws introduced by the programmability of smart contracts and security vulnerabilities arising from interactions between smart contracts and the contract virtual machine. The application of blockchain in currency-related use cases necessitates heightened attention to potential risks. Any oversight in the development of smart contracts can result in the deployment of insecure applications onto the blockchain, where their immutable nature makes rectification nearly impossible, leading to irreversible financial losses[14, 30]. Therefore, research on the security of smart contracts in Blockchain 2.0 is of critical significance. As shown in Figure 2, the second section of this paper will begin by discussing the foundational mechanisms that support blockchain, progressively introducing the various technologies within the Blockchain 2.0 system that incorporates smart contracts. In the third section, the Ethereum smart contract system will be presented as a typical example of Blockchain 2.0, along with its model. The fourth section will explore the implementation of cryptocurrencies in the form of smart contracts, and finally, in the fifth section, we will replicate significant security incidents that occurred on the Ethereum blockchain by writing a simple token contract, analyzing common security issues, and discussing feasible solutions and mitigation strategies.

2 BACKGROUND

2.1 Blockchain

Blockchain technology is a new distributed infrastructure and computing paradigm that uses a block-based data structure to verify and store data, employs a consensus algorithm from distributed nodes to update data, ensures the security of data transmission and access through cryptography, and utilizes smart contracts (automated script code) to program and manipulate data[47].

Blockchain establishes a decentralized model under zero trust, making it the core of encrypted digital currencies[31]. The main components of blockchain include blocks, chains, and the operations stored within them, namely transactions.

- `Block`: A block records transactions and states within a specific period, serving as the fundamental storage unit of the blockchain and a collection of transactions that have been completed.
- `Chain`: A chain structure that links blocks in chronological order using hashes.
- `Transaction`: All operations in the blockchain network are treated as transactions, also known as operations, through which records are generated on the blocks and the corresponding states are altered.

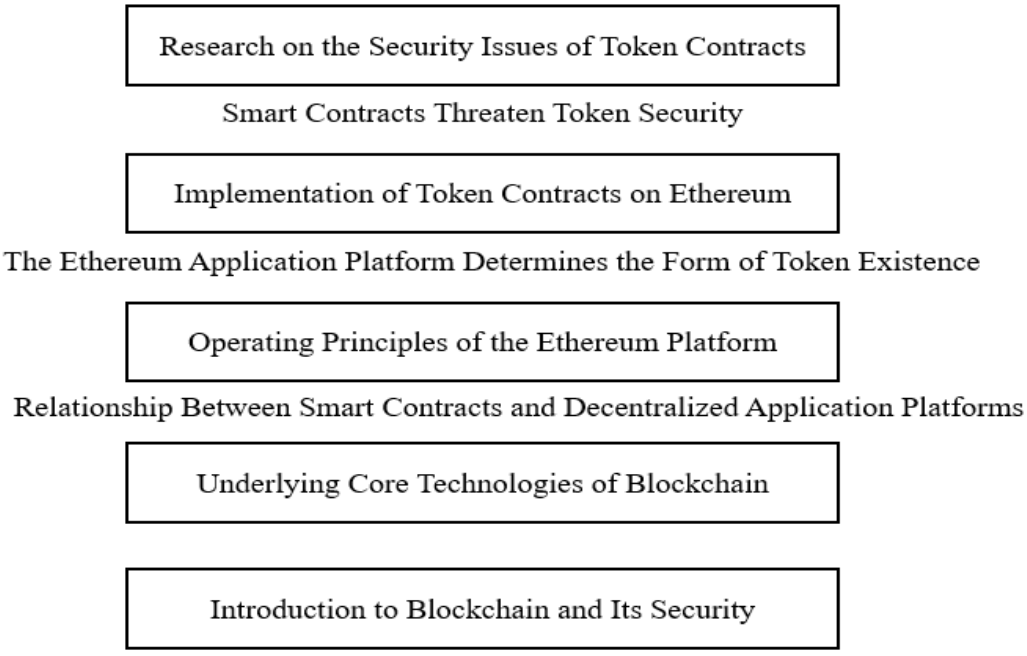

Fig. 2. Structural Framework of Token Contract Security Research.

2.2 Typical blockchain hierarchical structure

As shown in Figure 3, from the perspective of the blockchain hierarchical structure, the blockchain is composed of the data layer, network layer, consensus layer, incentive layer, contract layer, and application layer, from bottom to top[2]. At the data level, the blockchain structure with chain-based storage and Merkle trees stores data in parallel, ensuring the integrity of the content through technologies such as hashing, digital signatures, and asymmetric encryption. Transactions initiated by clients are broadcasted in the P2P network after verification and temporarily stored as unconfirmed transactions on all nodes. Every period, machines across the network package all transactions within that time slice into blocks, and consensus is reached at the consensus layer, ensuring consistency of the data on the blockchain across all nodes in the network[49]. The typical consensus mechanism in the blockchain 1.0 era was Proof of Work, and once the block achieved consensus, it possessed the characteristic of immutability[50]. Later, Ethereum introduced the Proof of Stake (POS) mechanism. The operation of the blockchain relies on miners, and at the incentive layer, virtual currency issuance and redistribution are realized. Both running contracts and transactions require paying miners with Ether as transaction fees. On this basis, blockchain 2.0 introduces smart contracts, which invoke contracts through data embedded in transactions. When packaging blocks, the defined virtual machine executes the contract scripts, enabling complex programmable functions. Finally, various user applications are realized through the interfaces provided by the smart contracts[45].

2.3 Consensus Mechanism

To address the trust issue in distributed networks, an effective consensus mechanism is required to balance system efficiency and usability.

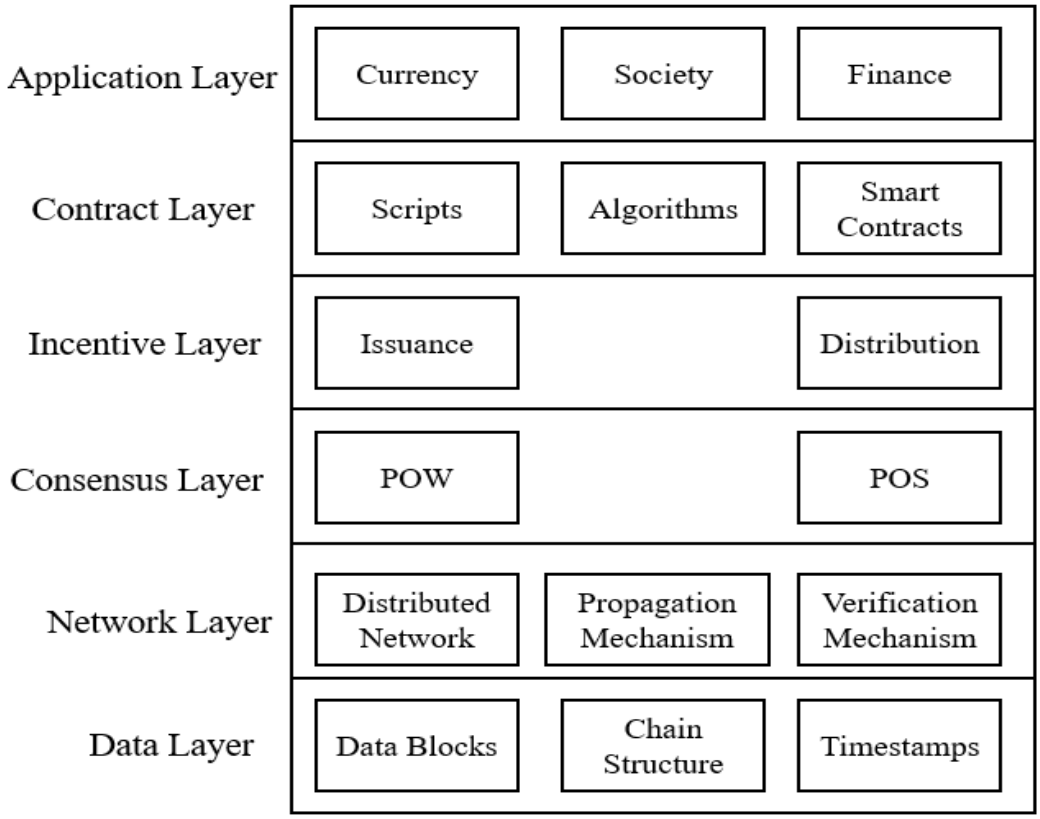

Fig. 3. Layered Technical Stack of Blockchain Systems.

The consensus mechanism used by early virtual currency systems such as Bitcoin and the early stages of Ethereum is Proof of Work (PoW), which ensures data consistency and consensus through computational power competition among network nodes[16]. All nodes attempt to find a random number (Nonce) that makes the hash of the current block smaller than a certain value or starts with a specific number of zeros. When a node discovers this random number, it gains the right to record the block and is rewarded as a miner. The essence of "mining" in blockchain is the computational brute force to break the hash. The Proof of Work mechanism can secure the blockchain, ensuring that the accounting rights are random and providing effective protection for the secure operation of blockchain systems like Bitcoin, as long as more than 51% of the computational power is not controlled maliciously. Compared to Proof of Work, Proof of Stake significantly improves energy efficiency by eliminating the need for large-scale ASIC computational power competition[4, 7, 40]. At the same time, it greatly increases the cost of attacks and lowers the entry barriers for participating in blockchain maintenance, thus enabling more individuals to get involved and providing stronger decentralization.

3 THE OPERATING PRINCIPLES OF THE ETHEREUM PLATFORM

This paper takes Ethereum as a typical representative of Blockchain 2.0 and studies the security issues of smart contracts under Blockchain 2.0. Therefore, it first introduces the technical details of the Ethereum platform.

3.1 Ethereum Application Technology

3.1.1 Ethereum Virtual Machine. The Ethereum Virtual Machine serves as the execution environment for smart contracts[18]. All smart contracts and state modifications on the Ethereum blockchain are conducted through transactions, and every transaction on the Ethereum network is executed by the EVM. The EVM introduces an abstraction layer above Ethereum nodes, enabling Turing completeness through the execution of specific operations defined by 140 opcodes[15, 17]. This capability allows the EVM to support the deployment and execution of various smart contracts with diverse functionalities [16]. Essentially, the Ethereum Virtual Machine is a stack-based machine, with its primary function being the execution of smart contracts. It operates with a 256-byte word size, a stack depth of 1024, and is designed with simplicity, determinism, space efficiency, blockchain-oriented functionality, security assurances, and optimization in mind. Data within the EVM can be stored in three distinct locations: the stack, temporary storage, and persistent storage[3, 36]. The EVM executes operations by interpreting opcodes within smart contracts, manipulating on-chain and transaction data to produce a deterministic and unique outcome.

3.1.2 Smart Contract. A computer program capable of automatically enforcing contractual terms is defined as a smart contract. Although the concept of smart contracts emerged almost simultaneously with the internet, there was no perfect and reliable technological solution to ensure the security and trustworthiness of contract execution until the advent of blockchain technology[9, 20]. In the context of Ethereum, a smart contract is an executable program that runs on the Ethereum blockchain. These contracts are stored on-chain and assigned a unique address. Their execution is triggered by transactions sent to this address, incurring computational costs and modifying the blockchain state. Smart contracts also serve as a public interface for interactions between users and decentralized applications [6**?**]. Once deployed, a smart contract remains persistently available, is difficult to modify, and cannot be revoked.

3.1.3 Ethereum Nodes. For an application to interact with the Ethereum blockchain, it must connect to an Ethereum node, which serves as the gateway to the entire Ethereum network. An Ethereum node is a computer running an Ethereum client—an implementation of the Ethereum protocol capable of validating all transactions within each block, thereby ensuring network security and data accuracy. Ethereum nodes collectively maintain the state of the blockchain and achieve consensus on state changes through the underlying consensus algorithm[46]. By facilitating communication between applications and the blockchain, Ethereum nodes play a crucial role in maintaining the integrity and functionality of the Ethereum ecosystem.

3.1.4 Ethereum Client API. Applications connect to and communicate with the Ethereum blockchain through API libraries developed and maintained by the Ethereum open-source community. These APIs abstract much of the complexity associated with direct interaction with Ethereum nodes, significantly reducing the technical burden on developers. Additionally, these libraries provide convenient functions that allow developers to spend less time dealing with the intricacies of Ethereum clients and instead focus on implementing the business logic of their applications.

3.1.5 End-User Applications. At the top of the stack are user-facing applications, which primarily include two common types: web applications and mobile applications. Due to well-designed encapsulation and development, users often do not need to be aware that the applications they are using are built on blockchain technology.

3.2 Ethereum Blockchain Model

Compared to the Bitcoin system, Ethereum, despite having a similar cryptocurrency (ETH) that follows nearly identical intuitive rules, offers more powerful functionality through smart contracts. Rather than merely serving as a distributed ledger, Ethereum is better characterized as a distributed state machine. The state of Ethereum is a

large data structure that not only records all accounts and balances but also maintains a machine state that can transition between blocks according to a predefined set of rules and execute arbitrary machine code[33]. The specific rules governing state transitions within blocks are defined by the Ethereum Virtual Machine.

The Ethereum state contains a vast number of transactions stored within blocks, which are linked sequentially over time. Each time a new block is generated, it must be validated through a consensus algorithm to ensure network-wide agreement.

3.2.1 Account. The global state of Ethereum consists of individual accounts, each with its own state and a unique 20-byte address. Ethereum accounts are categorized into externally owned accounts and contract accounts[24]. EOAs, controlled by external private keys, are not associated with any code and are entirely managed by their respective private key holders. In contrast, contract accounts are governed exclusively by the smart contract code deployed on the Ethereum network. Both types of accounts can receive, hold, and transfer Ether and tokens, as well as interact with deployed smart contracts. An EOA can initiate a transaction by signing it with its private key to transfer assets to another EOA or a contract account. When a transaction is sent to a contract account, it triggers the execution of the associated contract code[42, 44]. Unlike EOAs, contract accounts cannot independently initiate transactions; they can only generate transactions in response to received transactions through their triggered code execution.

3.2.2 State. In the context of Ethereum, the state is a large data structure known as the modified Merkle Patricia Trie, which links all accounts through hashes, allowing them to be traced back to a single root hash stored on the blockchain. The state is divided into account state and global state, and these two types of state will be described separately. The account state consists of four components: the nonce, balance, storage root, and code hash.

- `Nonce`: A block records transactions and states within a specific period, serving as the fundamental storage unit of the blockchain and a collection of transactions that have been completed.
- `Balance`: A chain structure that links blocks in chronological order using hashes.
- `StorageRoot`: All operations in the blockchain network are treated as transactions, also known as operations, through which records are generated on the blocks and the corresponding states are altered.
- `CodeHash`: For externally owned accounts (EOAs), the corresponding field is empty. In contrast, for contract accounts, this field stores the hash of the account's contract code.

The global state of Ethereum is represented as a mapping from account addresses to their corresponding account states. This mapping is maintained in a data structure known as the Merkle Patricia Trie (MPT), a specialized form of binary tree. The MPT is composed of a set of nodes and exhibits the following two properties.

- A large number of leaf nodes that contain the underlying data.
- Each parent node stores the hash values of its two child nodes.

The MPT enables lightweight clients in the blockchain network to process information such as transactions, events, and balances without storing the entire blockchain. Due to the hash propagation property of the MPT, maliciously submitted falsified data (e.g., fake transactions) can be effectively prevented[25]. By verifying the hashes in the block header, all nodes can validate a small subset of Ethereum's global state without needing to maintain the full chain.

3.2.3 Transaction Fees. Similar to typical blockchain systems, all transactions on Ethereum require the payment of a transaction fee, known as gas. Users specify a gasLimit to cap the maximum amount of gas they are willing to consume for a transaction, and a gasPrice to determine the amount they are willing to pay per unit of gas. If the gas provided is insufficient to complete the transaction, the transaction fails, and all state changes made during its execution are reverted. Only when sufficient gas is supplied will the transaction be validated and confirmed. The consumed gas serves as a reward for miners (or, in the context of Ethereum 2.0, validators).

Miners have the autonomy to select which transactions they wish to validate or ignore. As a result, when constructing a block, miners tend to prioritize transactions with higher gas prices in order to maximize their rewards. Consequently, transaction initiators often increase the gas price to improve the likelihood of their transactions being included in a block.

For contract-related transactions, gas is also required to cover the additional costs of computation and storage[8, 19]. Computation on Ethereum is intentionally expensive, a design choice aimed at safeguarding the network's integrity: every computation performed on-chain incurs a cost, thereby discouraging the submission of spam or malicious activity. To further mitigate risks such as unintended or malicious infinite loops and other forms of computational waste, each transaction must specify a limit on the number of computation steps it is allowed to execute.

3.2.4 Transaction. Transactions are the fundamental operations in Ethereum, enabling the transition of the system from one state to another. There are two types of transactions: message calls and contract creation. Both types share the same structure, which includes the following fields.

- `Nonce`: A sequential number indicating the sender's transaction count.
- `Gas Price`: The amount the sender is willing to pay per unit of gas.
- `Gas Limit`: The maximum amount of gas the sender is willing to provide for the transaction.
- `To (Recipient Address)`: The address of the recipient. This field is left empty (i.e., set to the zero address) for contract creation transactions.
- `Value`:The amount of Ether to be transferred to the recipient address.
- `Init`:A field used only in contract creation transactions. It contains the initialization code for the smart contract. Upon execution, it returns the address of the newly deployed contract.
- `Data`:A field used only in message call transactions. It carries input parameters to be passed during the invocation of the contract function.

4 IMPLEMENTATION OF ETHEREUM TOKEN CONTRACTS

4.1 Token Standard Interface

Ethereum smart contracts offer powerful and reliable execution capabilities. Once predefined conditions are met, the contract automatically executes according to the logic encoded within it. This makes smart contracts particularly well-suited for applications in the domain of digital assets. One common use case is the development of tokens on the Ethereum platform. A token represents a unit of value or ownership on the blockchain. Like traditional cryptocurrencies, tokens can be transferred between contracts, queried for their total supply, or checked for individual account balances.

In late 2015, Fabian Vogelsteller proposed the ERC-20 standard (also known as EIP-20), which defines a standardized interface capturing these token characteristics[12]. This interface enables Ethereum wallets and other contracts to interact with tokens in a unified manner. ERC-20 specifies a set of rules that all fungible Ethereum tokens should follow, allowing developers to predict how new tokens will behave within the broader Ethereum ecosystem. For instance, token exchange protocols like Uniswap can support any newly issued ERC-20-compliant token without modification.

The ERC-20 standard is closely associated with Initial Coin Offering contracts, where a predefined amount of Ether is raised, and corresponding tokens are issued to users once the fundraising goal is met[10]. ERC-20 defines only the standard interface and not its concrete implementation; developers must implement and maintain the actual contract code themselves. Since ERC-20 tokens focus solely on token quantity and treat all units as identical and interchangeable, they are classified as fungible tokens, and ERC-20 is the de facto standard for such tokens[39, 41, 43].

In contrast, non-fungible tokens, where each token is unique and distinguishable from others, follow the ERC-721 standard[21, 38]. This type of token is widely used in scenarios involving digital collectibles, intellectual property rights, and digital agreements. The core interface includes the following functions:

- `totalSupply`: Returns the total supply of the token. Although token supply is usually fixed, this function allows the contract to return the effective circulating supply.
- `balanceOf`: Takes an address as input and returns the token balance of that address. All token balances are publicly visible.
- `transferFrom`: Transfers a specified number of tokens from one address to another. This operation must emit a Transfer event. It is primarily used to allow a designated spender to transfer tokens on behalf of the owner. The number of tokens that can be transferred is constrained by the allowance, which must be set via the approve function.
- `approve`: Grants permission to a spender to withdraw a specified number of tokens from the owner's account.

4.2 Contract Compilation and Deployment

4.2.1 Tools.

- `MetaMask`: MetaMask is currently the most widely used Ethereum wallet and gateway. Available as a browser extension and mobile application, it manages digital assets through a secure local key vault. MetaMask enables users to buy, store, send, and exchange tokens, and also provides a simple and secure interface for connecting to Ethereum-based blockchain applications via RPC endpoints. This allows seamless interaction with decentralized applications directly from web pages using the local wallet.
- `TestNet`: All nodes operating on the same blockchain are considered part of a single network. The network most commonly used in practice is the main network (MainNet), where users conduct real transactions and deploy production smart contracts. Due to its large user base, the MainNet provides high security and strong resistance against attacks and tampering, making it a robust and trustworthy environment. Consequently, Ether on the MainNet has real-world value, and testing on the MainNet incurs a significant cost. While a blockchain typically has only one MainNet, it may also support multiple test networks (TestNets) designed for experimentation, learning, and testing. A TestNet is an independent blockchain that starts from a different genesis block and may employ a different consensus mechanism than the MainNet. Users can mine blocks and perform testing without incurring real-world costs, making TestNets essential for safe and effective development workflows.

4.2.2 Deployment. After compiling the contract in Remix, the Injected Web3 environment is used to connect to the Rinkeby test network via MetaMask. The constructor parameter is set to 2048, indicating the issuance of a total of 2048 tokens. All tokens are initially assigned to the deployer's own address to facilitate testing.

After confirming the transaction and paying the gas fee, the deployment is completed once the transaction is mined and confirmed. The contract can then be inspected via Etherscan. The transaction was confirmed in block number 10593657 on the test network. The previously empty To field is filled with the generated contract address. Since the transaction is only intended to deploy or invoke a contract, the Ether transfer amount can be zero. Given that a sufficient amount of gas was provided, the transaction was successfully executed and all allocated gas was consumed.

As discussed earlier, Ethereum operates as a distributed state machine. This transaction resulted in state transitions for three different account addresses. As shown in Figures 4 and 5, a new contract address was created with associated storage space, and the sender's account balance was reduced due to the gas fee. Additionally, the nonces of both the sender and the contract addresses were incremented by one after the transaction.

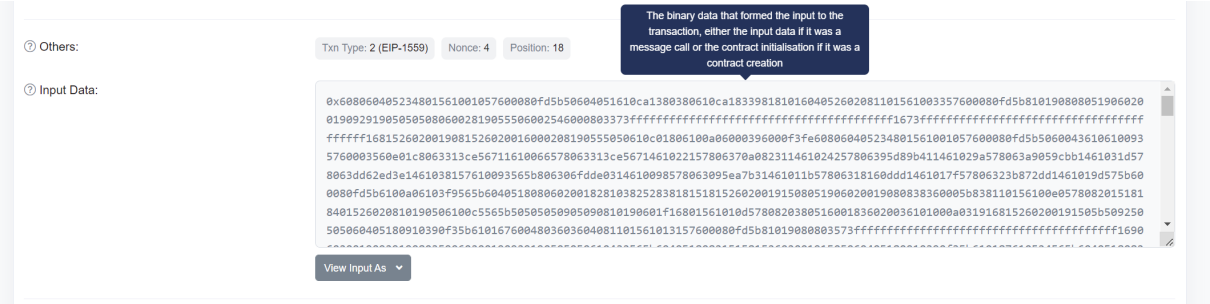

Fig. 4. Contract Bytecode Embedded in the Transaction.

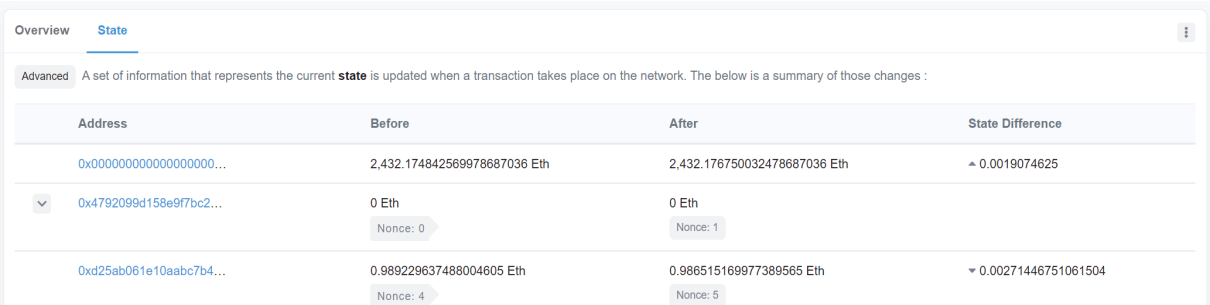

Fig. 5. State Transitions Caused by the Transaction.

As illustrated in Figure 6, the contract's storage was initially empty upon deployment. This transaction initializes the storage layout. Specifically, the first storage slot holds the totalSupply_ variable set during construction, while the second slot stores the mapping entry that associates the current wallet address with its token balance.

Fig. 6. Storage Changes at the Contract Address Resulting from the Transaction.

As shown in Figure 7, the transaction that invoked the contract included a sequence of encoded data. This data encodes the function selector followed by its parameters in order: the target address (parameter 1) and the amount to transfer (parameter 2).

Due to the standardized ERC-20 interface, Etherscan can automatically recognize the token properties and generate a token tracking page with key metadata, as shown in Figure 8. As a result of the transfer operation

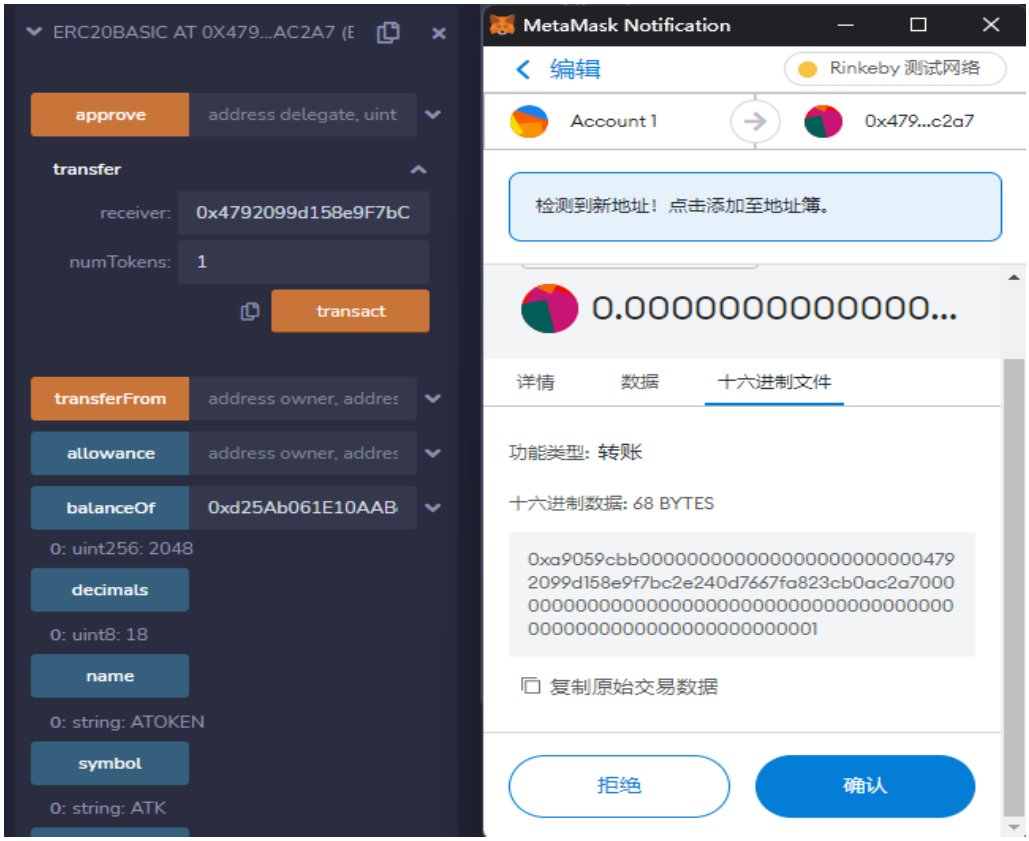

Fig. 7. Token Transfer Invocation via Remix's Contract Interaction Interface.

performed in this transaction, the number of token-holding addresses increased from one to two, with token balances of 2047 and 1, respectively.

5 COMMON SECURITY THREATS IN TOKEN CONTRACTS

Smart contract vulnerabilities refer to potential security flaws in the contract code that, if exploited by attackers, can result in asset losses within the contract[28, 29]. Since The DAO attack, hackers have realized that smart contracts represent a lucrative target, leading to an era of active vulnerability hunting. Numerous security issues have since been uncovered. In this paper, we classify these vulnerabilities into two categories: those arising from the design characteristics of the Ethereum platform, and those caused by traditional attack techniques. Classic vulnerabilities are reproduced and analyzed to better understand their causes and consequences.

5.1 Ethereum Platform-Induced Vulnerabilities

5.1.1 Re-Entrancy Vulnerability. Re-entrancy is a vulnerability where an attacker exploits the fallback function to recursively invoke a vulnerable transfer method, allowing repeated withdrawals before the contract's state is correctly updated. This recursive behavior continues until the transaction either runs out of gas or reaches a specified termination condition. As a result, attackers can withdraw tokens far exceeding their initial deposit.

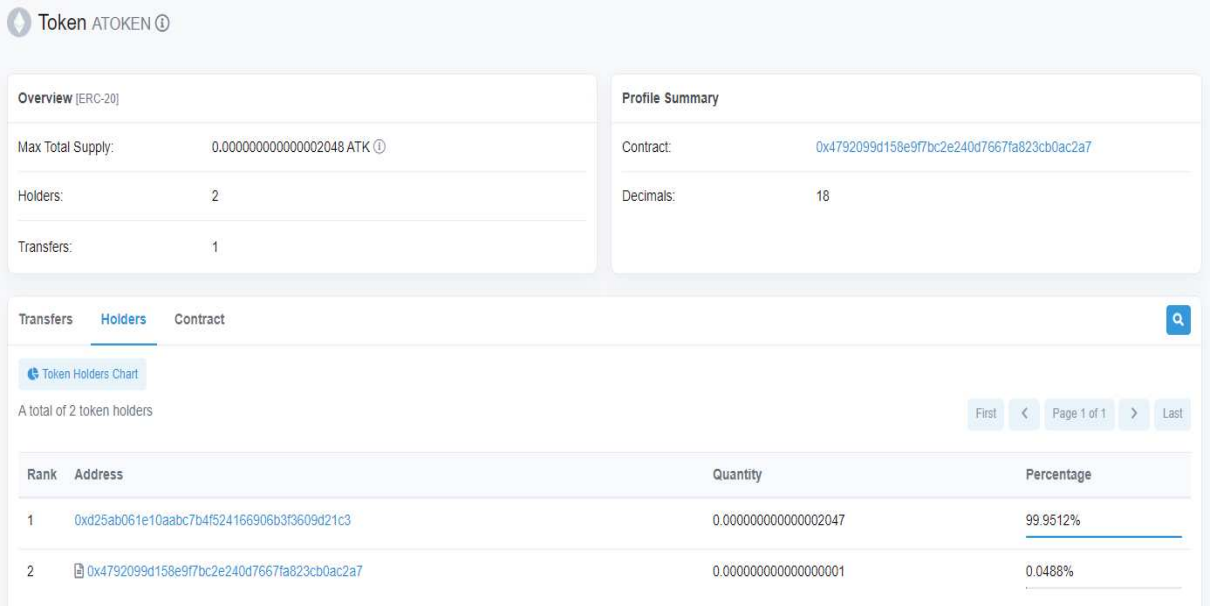

Fig. 8. Token Tracking Page for ATOKEN on Etherscan.

The fallback function is a special unnamed function in a contract, and each contract may define only one such function. In versions prior to Solidity 0.4.x, visibility was not strictly required. From version 0.5.x onward, fallback functions must be explicitly marked as external. These functions can be virtual, overridden, or decorated with modifiers, and are triggered in the following two scenarios:

- `Function Not Found`: If a function call does not match any defined function signatures in the contract, the fallback function is automatically invoked. This means that any invalid call (i.e., one with an undefined function selector) will result in fallback execution.
- `Using send() to Transfer Ether`: When Ether is sent to a contract via the send() method without any attached data, the fallback function is invoked. As long as no valid contract method is explicitly called, the fallback function will be executed by default.

The infamous The DAO vulnerability on Ethereum falls into this category. This vulnerability led to over $10 million in losses and directly triggered the Ethereum hard fork. It also inspired a wave of vulnerability research in the smart contract community. In the payout method, if the _recipient is a contract address, the call instruction will invoke the recipient's fallback function. Since call does not restrict gas usage, the fallback function can recursively invoke payout, leading to a re-entrant attack that drains the contract's balance. After deploying the vulnerable contract, we fund it by transferring Ether to its address. Based on the vulnerability, we can write an attack contract as shown in Listing 1. The core part of the attack is implemented in the fallback function, where a call to the splitDAO function of the TheDAO contract is made to achieve the reentrancy effect. A limit on the number of iterations is set to prevent the transaction from reverting due to gas exhaustion. Finally, a withdrawal function is used to extract the Ether obtained from the attack.

Listing 1: Attack contract

```
contract HackCode {
    address public daoContract;
    uint public count = 50;
    uint public n;
    function setDAO(address _addr) public {
        daoContract = _addr;
    }
    function getBalance() public view returns (uint) {
        return address(this).balance;
    }
    function withdraw() public {
        msg.sender.transfer(address(this).balance);
    }
    function setCount(uint newCount) public {
        count = newCount;
    }
    function () public payable {
        if(n < count){
            n++;
            TheDAO(daoContract).splitDAO();
        }
    }
}
```

Attack Execution Process:

- Deploy the attack contract.
- Invoke the setDAO method via a transaction to set the target address variable to the victim contract.
- Invoke the setCount method via a transaction to set the count to an appropriate value.
- Trigger the fallback function of the attack contract through a regular transaction.
- The attack contract will call the victim DAO contract's splitDAO method, which internally invokes withdrawRewardFor, and ultimately calls payout. Within payout, a call is made to transfer Ether to the attack contract, which triggers the fallback function again, leading to re-entrancy. The splitDAO method is entered again, causing Ether to be withdrawn repeatedly. Even if a check is placed after the transfer, it will not take effect during this process.
- Upon a successful attack, the attack contract extracts most of the Ether from the victim contract. At this point, the attacker can call withdraw to retrieve the stolen funds from the attack contract and end the attack.

Mitigation and Prevention:

- Whenever possible, use send or transfer instead of call. The gasLimit of send and transfer is 2300, which is insufficient to support even the simplest function call, thus preventing fallback execution.
- For functions involving transfers and payments, adopt a "checks-effects-interactions" pattern: first validate conditions, then update state, and finally perform the transfer. This helps prevent many potential issues.

5.1.2 Delegatecall Vulnerability. In Solidity, there are two methods to invoke external contracts: call and delegatecall, each with different contextual behaviors.

- `Call`: When using call to invoke an external contract method, the execution context is that of the external contract. When Contract A calls a function of external Contract B via call, it executes in the context of Contract B and then returns to Contract A to continue execution.
- `delegatecall`: When using delegatecall to invoke an external contract method, the execution context is that of the local contract. When Contract A calls a function of external Contract B via delegatecall, it is equivalent to copying Contract B's code and executing it within Contract A's context.

Attack Execution Process:

- Account 1 deploys the Delegate contract.
- Account 1 deploys the Delegation contract, specifying the address of the Delegate contract to act as its proxy.
- At this point, verify that the owner of both contracts is the same, i.e., the address of Account 1.
- Account 2 invokes the fallback function of Delegation, modifying the owner address.
- Check the owner of Delegation, which has now been changed to the address of Account 2.

Mitigation and Prevention:

- Use delegatecall with caution, and clearly define function visibility. Sensitive functions should be declared as external to prevent unintended external invocation.

5.2 Traditional Vulnerabilities Reproduced on the Ethereum Platform

5.2.1 Integer Overflow Vulnerability. Due to the limited number of bits that a register can represent, when a stored value exceeds the maximum representable range, overflow may occur. Maximum value overflow wraps around to the minimum value, and minimum value overflow wraps around to the maximum. This issue can occur on any platform and is one of the most common and universal types of vulnerabilities [25]. Similarly, the Ethereum Virtual Machine (EVM) assigns fixed-size data types for integers. An integer variable can only represent values within a specific range. For example, the largest integer type is uint256, which has a maximum value of 2**256 - 1. Exceeding this range will result in overflow.

In the transfer condition check, require(balances[msg.sender] - _value >= 0) presents a clear integer overflow issue. Since uint is actually uint256, transferring more than the initial balance causes an underflow, turning (balances[msg.sender] - _value) into a large positive integer. This allows transferring more tokens than the actual balance and inflates the sender's balance to an extremely large number. The attack method involves directly calling the transfer function with a value greater than the available balance.

To protect contracts from overflow vulnerabilities, arithmetic functions in the SafeMath library are typically used to replace regular addition, subtraction, multiplication, and division. However, any oversight may lead to severe vulnerabilities.

As shown in Listing 3, the vulnerability that once caused BeautyChain tokens to be minted infinitely and their value to drop to zero is of this integer overflow type. Although the contract included and used the SafeMath library, one instance of default multiplication led to the problem.

Listing 2: The vulnerability function of Meitu Coin

```
contract BeautyChain {
    using SafeMath for uint256;
    mapping (address => uint256) public balances;

    function batchTransfer(address[] _receivers, uint256 _value) public returns (bool) {
        uint cnt = _receivers.length;
        uint256 amount = uint256(cnt) * _value;
        require(cnt > 0 && cnt <= 20);
        require(_value > 0 && balances[msg.sender] >= amount);

        balances[msg.sender] = balances[msg.sender].sub(amount);
        for (uint i = 0; i < cnt; i++) {
            balances[_receivers[i]] = balances[_receivers[i]].add(_value);
        }

        return true;
    }
}
```

The variable amount is then used as a condition in a subsequent transfer operation. Since this is a public function, both _receivers and _value are controllable. By causing amount to overflow upwards into a very small value, the validation can be bypassed to allow the transfer of a large value. A feasible set of attack vectors can be constructed accordingly.

- Set _receivers as an array containing two recipient addresses, such that _receivers.length = 2.
- Set _value = 0x8000000000000000000000000000000000000000000000000000000000000000.

Under this condition, _receivers.length * _value causes an overflow, resulting in amount = 0, thereby bypassing the balance check and allowing the transfer of the aforementioned amount of tokens.

Fix and Prevention:

- Simply replacing * with the mul function from the SafeMath library can prevent the overflow issue.

Such problems are easy to detect through tooling and auditing. When performing arithmetic operations in a contract, overflow checks must be properly implemented. Starting from Solidity version 0.8.0, all arithmetic operations include built-in overflow checks by default, eliminating the need for external libraries. Therefore, using a newer version of Solidity also contributes to improved security.

5.2.2 Random Number Predictability. Random numbers are widely used in scenarios such as lotteries, games, and signature algorithms. They form the foundation of cryptography and privacy security in traditional internet applications. However, achieving true randomness on the blockchain is particularly challenging. As a distributed consensus network, everything on the blockchain is public—including algorithms, variables, and states—and there is often no suitable source of entropy, making random numbers predictable. Platforms often rely on on-chain public information as randomness sources. Each platform has unique attributes that are publicly accessible; therefore, using such attributes as seeds for randomness undermines unpredictability. Many real-world attacks have occurred on platforms like Ethereum and EOS.

In this example, the most significant bit of the previous blockhash is used as the random bit for the coin flip. Since the hash of the previous block is fully known, one can observe it and then call the guessing function before

the next block is mined. An attacker can deploy a contract to predict and interact with the vulnerable contract, as shown in Listing 3, achieving consecutive wins through random number prediction.

Listing 3: Prediction attack contract targeting the coin-flip contract

```
contract Attack {
    CoinFlip fliphack;
    address victim;
    uint256 FACTOR =
        57896044618658097711785492504343953926634992332820282019728792003956564819968;

    function Attack(address victim) {
        fliphack = CoinFlip(victim);
    }
    function predict() public view returns (bool) {
        uint256 blockValue = uint256(block.blockhash(block.number - 1));
        uint256 coinFlip = uint256(uint256(blockValue) / FACTOR);
        return coinFlip == 1 ? true : false;
    }
    function hack() public {
        bool guess = predict();
        fliphack.flip(guess);
    }
}
```

In 2018, the FoMo3D contract on Ethereum suffered a prediction attack on its random airdrop algorithm because it used the previous block hash as the entropy source. The attacker used random number prediction to decide in advance whether to participate, thereby capturing large airdrop rewards.

Mitigation Strategies:

- Since all content on the blockchain is transparent to participants, using randomness in Ethereum is inherently complex. However, several approaches can improve security. For instance, using less predictable pseudo-random sources such as block timestamps, or relying on off-chain oracles to generate randomness for on-chain use [27].

6 CONCLUSION

This paper uses Ethereum as a case study to analyze the technical applications of cryptocurrency and associated security issues in Blockchain 2.0, characterized by support for smart contracts. Compared to traditional application platforms, smart contracts represent a relatively new paradigm. While they generally consist of smaller codebases, increasing complexity combined with insufficiently tested code and inherent platform features still present many security vulnerabilities. For a blockchain network that supports millions of contracts and a value-based ecosystem, security is a highly sensitive concern. Despite the current low levels of security assurance, substantial investment in blockchain will continue to drive security research. As new attack patterns and vulnerabilities are discovered, classification of related issues will continue to evolve.Research into smart contract security contributes to enhancing their robustness and supports the development of a more secure and reliable Blockchain 2.0 environment. Currently, the primary method for ensuring contract security involves auditing and detection-based validation [28], along with formal verification of certain components [29]. However, existing tools and audits cannot eliminate security risks. Many contracts that have passed both automated and manual checks still

exhibit vulnerabilities. Thus, the security of smart contracts needs further reinforcement, and the development of corresponding security detection tools remains an open and critical area for advancement[32].

REFERENCES

[1] Shubhani Aggarwal and Neeraj Kumar. 2021. Blockchain 2.0: smart contracts. In *Advances in computers*. Vol. 121. Elsevier, 301–322.

[2] Mahmoud Tayseer Al Ahmed, Fazirulhisyam Hashim, Shaiful Jahari Hashim, and Azizol Abdullah. 2022. Hierarchical blockchain structure for node authentication in IoT networks. *Egyptian Informatics Journal* 23, 2 (2022), 345–361.

[3] Elvira Albert, Maria Garcia de la Banda, Alejandro Hernández-Cerezo, Alexey Ignatiev, Albert Rubio, and Peter J Stuckey. 2024. SuperStack: Superoptimization of stack-bytecode via greedy, constraint-based, and SAT techniques. *Proceedings of the ACM on Programming Languages* 8, PLDI (2024), 1437–1462.

[4] Merlinda Andoni, Valentin Robu, David Flynn, Simone Abram, Dale Geach, David Jenkins, Peter McCallum, and Andrew Peacock. 2019. Blockchain technology in the energy sector: A systematic review of challenges and opportunities. *Renewable and sustainable energy reviews* 100 (2019), 143–174.

[5] Jiuyang Bu, Wenkai Li, Zongwei Li, Zeng Zhang, and Xiaoqi Li. 2025. Enhancing Smart Contract Vulnerability Detection in DApps Leveraging Fine-Tuned LLM. *arXiv preprint arXiv:2504.05006* (2025).

[6] Jiuyang Bu, Wenkai Li, Zongwei Li, Zeng Zhang, and Xiaoqi Li. 2025. SmartBugBert: BERT-Enhanced Vulnerability Detection for Smart Contract Bytecode. *arXiv preprint arXiv:2504.05002* (2025).

[7] Eric Budish. 2018. *The economic limits of bitcoin and the blockchain.* Technical Report. National Bureau of Economic Research.

[8] Jiachi Chen, Xin Xia, David Lo, John Grundy, and Xiaohu Yang. 2021. Maintenance-related concerns for post-deployed Ethereum smart contract development: issues, techniques, and future challenges. *Empirical Software Engineering* 26, 6 (2021), 117.

[9] Konstantinos Christidis and Michael Devetsikiotis. 2016. Blockchains and smart contracts for the internet of things. *IEEE access* 4 (2016), 2292–2303.

[10] Paul Cuffe. 2018. The role of the erc-20 token standard in a financial revolution: the case of initial coin offerings. (2018).

[11] Dion Curry. 2025. Limitations of trust and legitimacy in blockchain: exploring the effectiveness of decentralisation, immutability and consensus mechanisms in blockchain governance. *International Journal of Public Sector Management* 38, 1 (2025), 98–117.

[12] Fabian Dietrich, Louis Louw, and Daniel Palm. 2023. Blockchain-based traceability architecture for mapping object-related supply chain events. *Sensors* 23, 3 (2023), 1410.

[13] Ardit Dika and Mariusz Nowostawski. 2018. Security vulnerabilities in ethereum smart contracts. In *2018 IEEE international conference on Internet of Things (iThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 955–962.

[14] Phan The Duy, Nghi Hoang Khoa, Nguyen Huu Quyen, Le Cong Trinh, Vu Trung Kien, Trinh Minh Hoang, and Van-Hau Pham. 2025. Vulnsense: Efficient vulnerability detection in ethereum smart contracts by multimodal learning with graph neural network and language model. *International Journal of Information Security* 24, 1 (2025), 48.

[15] Dénes László Fekete and Attila Kiss. 2023. Toward building smart contract-based higher education systems using zero-knowledge Ethereum virtual machine. *Electronics* 12, 3 (2023), 664.

[16] Md Sadek Ferdous, Mohammad Jabed Morshed Chowdhury, and Mohammad A Hoque. 2021. A survey of consensus algorithms in public blockchain systems for crypto-currencies. *Journal of Network and Computer Applications* 182 (2021), 103035.

[17] Everett Hildenbrandt, Manasvi Saxena, Xiaoran Zhu, Nishant Rodrigues, Philip Daian, Dwight Guth, and Grigore Roşu. 2017. Kevm: A complete semantics of the ethereum virtual machine. (2017).

[18] Yoichi Hirai. 2017. Defining the ethereum virtual machine for interactive theorem provers. In *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*. Springer, 520–535.

[19] Kimia Honari, Sara Rouhani, Nida E Falak, Yuan Liu, Yunwei Li, Hao Liang, Scott Dick, and James Miller. 2023. Smart contract design in distributed energy systems: a systematic review. *Energies* 16, 12 (2023), 4797.

[20] Shafaq Naheed Khan, Faiza Loukil, Chirine Ghedira-Guegan, Elhadj Benkhelifa, and Anoud Bani-Hani. 2021. Blockchain smart contracts: Applications, challenges, and future trends. *Peer-to-peer Networking and Applications* 14 (2021), 2901–2925.

[21] Dechao Kong, Xiaoqi Li, and Wenkai Li. 2024. Characterizing the Solana NFT ecosystem. In *Companion Proceedings of the ACM Web Conference 2024*. 766–769.

[22] Satpal Singh Kushwaha, Sandeep Joshi, Dilbag Singh, Manjit Kaur, and Heung-No Lee. 2022. Ethereum smart contract analysis tools: A systematic review. *Ieee Access* 10 (2022), 57037–57062.

[23] Wenkai Li, Xiaoqi Li, Zongwei Li, and Yuqing Zhang. 2024. Cobra: interaction-aware bytecode-level vulnerability detector for smart contracts. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 1358–1369.

[24] Wenkai Li, Zhijie Liu, Xiaoqi Li, and Sen Nie. 2024. Detecting Malicious Accounts in Web3 through Transaction Graph. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 2482–2483.

[25] Xiaoqi Li et al. 2021. Hybrid analysis of smart contracts and malicious behaviors in ethereum. *Hong Kong Polytechnic University* (2021).
[26] Xiaoqi Li, Ting Chen, Xiapu Luo, and Chenxu Wang. 2021. CLUE: towards discovering locked cryptocurrencies in ethereum. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. 1584–1587.
[27] Xiaoqi Li, L Yu, and XP Luo. 2017. On Discovering Vulnerabilities in Android Applications. In *Mobile Security and Privacy*. Elsevier, 155–166.
[28] Zongwei Li, Wenkai Li, Xiaoqi Li, and Yuqing Zhang. 2024. StateGuard: Detecting State Derailment Defects in Decentralized Exchange Smart Contract. In *Companion Proceedings of the ACM Web Conference 2024*. 810–813.
[29] Zongwei Li, Xiaoqi Li, Wenkai Li, and Xin Wang. 2025. SCALM: Detecting Bad Practices in Smart Contracts Through LLMs. *arXiv preprint arXiv:2502.04347* (2025).
[30] Ruichao Liang, Jing Chen, Cong Wu, Kun He, Yueming Wu, Ruochen Cao, Ruiying Du, Ziming Zhao, and Yang Liu. 2025. Vulseye: Detect smart contract vulnerabilities via stateful directed graybox fuzzing. *IEEE Transactions on Information Forensics and Security* (2025).
[31] Yizhi Liu, Xiaohan Hao, Wei Ren, Ruoting Xiong, Tianqing Zhu, Kim-Kwang Raymond Choo, and Geyong Min. 2022. A blockchain-based decentralized, fair and authenticated information sharing scheme in zero trust internet-of-things. *IEEE Trans. Comput.* 72, 2 (2022), 501–512.
[32] Zekai Liu and Xiaoqi Li. 2025. SoK: Security Analysis of Blockchain-based Cryptocurrency. *arXiv preprint arXiv:2503.22156* (2025).
[33] Zekai Liu, Xiaoqi Li, Hongli Peng, and Wenkai Li. 2024. GasTrace: Detecting Sandwich Attack Malicious Accounts in Ethereum. In *2024 IEEE International Conference on Web Services (ICWS)*. IEEE, 1409–1411.
[34] Junjie Ma, Muhui Jiang, Jinan Jiang, Xiapu Luo, Yufeng Hu, Yajin Zhou, Qi Wang, and Fengwei Zhang. 2025. Understanding Security Issues in the DAO Governance Process. *IEEE Transactions on Software Engineering* (2025).
[35] Yingjie Mao, Xiaoqi Li, Wenkai Li, Xin Wang, and Lei Xie. 2024. SCLA: Automated Smart Contract Summarization via LLMs and Semantic Augmentation. *arXiv preprint arXiv:2402.04863* (2024).
[36] Somnath Mazumdar, Daniel Seybold, Kyriakos Kritikos, and Yiannis Verginadis. 2019. A survey on data storage and placement methodologies for cloud-big data ecosystem. *Journal of Big Data* 6, 1 (2019), 1–37.
[37] Pratyusa Mukherjee and Chittaranjan Pradhan. 2021. Blockchain 1.0 to blockchain 4.0—The evolutionary transformation of blockchain technology. In *Blockchain technology: applications and challenges*. Springer, 29–49.
[38] Yuanzheng Niu, Xiaoqi Li, Hongli Peng, and Wenkai Li. 2024. Unveiling wash trading in popular NFT markets. In *Companion Proceedings of the ACM Web Conference 2024*. 730–733.
[39] Reza Rahimian and Jeremy Clark. 2021. TokenHook: Secure ERC-20 smart contract. *arXiv preprint arXiv:2107.02997* (2021).
[40] Johannes Sedlmeir, Hans Ulrich Buhl, Gilbert Fridgen, and Robert Keller. 2020. The energy consumption of blockchain technology: Beyond myth. *Business & Information Systems Engineering* 62, 6 (2020), 599–608.
[41] Mukund Soni and Ekta Gandotra. 2023. ERC-20 Token Exchange System Over Blockchain Network. (2023).
[42] Mukesh Thakur et al. 2017. Authentication, authorization and accounting with Ethereum blockchain. *Helsingfors universitet* (2017).
[43] Mettupalle Chinnaiahgari Venkata ViswasReddy, Vootkuri Sai Charan Reddy, Gudeme Jaya Rao, and N Rama Devi. 2024. Efficient Token Transfer Using ERC-20 Decentralized Exchange. In *2024 International Conference on Intelligent Systems for Cybersecurity (ISCS)*. IEEE, 01–06.
[44] Qin Wang and Shiping Chen. 2023. Account abstraction, analysed. In *2023 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 323–331.
[45] Shuai Wang, Liwei Ouyang, Yong Yuan, Xiaochun Ni, Xuan Han, and Fei-Yue Wang. 2019. Blockchain-enabled smart contracts: architecture, applications, and future trends. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49, 11 (2019), 2266–2277.
[46] Wenbo Wang, Dinh Thai Hoang, Peizhao Hu, Zehui Xiong, Dusit Niyato, Ping Wang, Yonggang Wen, and Dong In Kim. 2019. A survey on consensus mechanisms and mining strategy management in blockchain networks. *Ieee Access* 7 (2019), 22328–22370.
[47] Yishun Wang, Xiaoqi Li, Shipeng Ye, Lei Xie, and Ju Xing. 2024. Smart contracts in the real world: A statistical exploration of external data dependencies. *arXiv preprint arXiv:2406.13253* (2024).
[48] Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. 2023. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *Comput. Surveys* 55, 11 (2023), 1–50.
[49] Chunyang Yu, Xuanlin Jiang, Shiqiang Yu, and Cheng Yang. 2020. Blockchain-based shared manufacturing in support of cyber physical systems: concept, framework, and operation. *Robotics and Computer-Integrated Manufacturing* 64 (2020), 101931.
[50] Changqiang Zhang, Cangshuai Wu, and Xinyi Wang. 2020. Overview of blockchain consensus mechanism. In *Proceedings of the 2020 2nd International Conference on Big Data Engineering*. 7–12.
[51] Huanhuan Zou, Zongwei Li, and Xiaoqi Li. 2025. Malicious Code Detection in Smart Contracts via Opcode Vectorization. *arXiv preprint arXiv:2504.12720* (2025).