Functional and Security-Related Aspects of Blockchains and Smart Contracts

Simran Tinani

1 Introduction

In recent years, blockchain technology has emerged as a pivotal and extensively researched advancement, demonstrating diverse applications across various domains. Although the initial surge in public interest was triggered by the Bitcoin white paper [20], blockchain technology has since evolved to accommodate a wide range of use cases such as cryptocurrencies, smart contracts, supply chain management, tokenization of assets, intellectual property protection, etc.

In loose terms, a blockchain is a decentralized ledger that records information across a network of computers and aims to preserve its accuracy and integrity without the appeal to a central authority. On the other hand, a smart contract is a self-executing contract with machine-encoded terms of the agreement. Smart contracts run on a blockchain network, utilizing its framework to execute and record contractual actions without the need for a central authority. This paper is an exploration of the functional and security aspects of blockchains, and more specifically, smart contracts.

In the first two sections of this paper, we explain the meaning, working principles, and key components of a blockchain, describing the role of cryptography and consensus mechanisms in shaping its architecture. We then provide a closer examination of smart contracts, which, we will see, are a special case of transactions on a blockchain. This discussion includes an explanation of some key terms related to the Ethereum network, which is the best-known blockchain hosting smart contracts. In the second half of the paper, we delve in-depth into the security aspects of both blockchain networks in general, and smart contracts specifically. We describe some of the most well-known vulnerabilities in the design and implementation of blockchains and smart contracts, providing some real-world examples of exploits in the past and some well-known mitigation strategies.

2 Distributed Networks, Decentralization, and Blockchain

2.1 Decentralized, distributed, and peer-to-peer systems

A distributed system is one in which multiple independent entities called nodes communicate and collaborate to achieve a common goal and provide a unified service. Decentralization is an architectural approach to implementing software systems, under which the network of components system is designed to have no central point of control, authority, or failure. In a decentralized system (in contrast to a centralized system), decision-making, control, and resources are distributed across

multiple nodes or entities. A peer-to-peer (P2P) system is a distributed network where individual participants ("nodes") communicate and share computational resources (e.g., processing power, storage capacity, or information) directly with each other without the need for a centralized authority. Many peer-to-peer systems exhibit decentralized characteristics, but it is possible to have P2P systems with varying degrees of centralization.

Many blockchain networks, such as Bitcoin and Ethereum, are both distributed and decentralized [11]. Each node in the network has a copy of the entire blockchain, and decisions are reached through a consensus mechanism without the need for a central authority.

2.2 Defining Blockchain

The blockchain is a distributed, decentralized peer-to-peer system of ledgers (a chronological record-keeping system) utilizing an algorithm, which records information across a network of computers (nodes) in the form of ordered and connected blocks, using cryptographic tools to maintain its integrity, security, transparency, and tamper-resistance.

Each node maintains its copy of the ledger. Transactions (state-changing functions) are grouped into blocks, and each block contains a reference to the previous block, thus forming a chain. This ensures the chronological order and immutability of the data. Once a block is added to the blockchain, it is nearly impossible to alter or delete. Immutability enhances the security and reliability of the ledger.

3 Working of a blockchain

As discussed in the previous section, blockchain networks are decentralized, so there is no central authority governing the system. In the absence of a central authority, alternative mechanisms are required to ensure that nodes across the network collectively agree on the validity and order of the data and to make the system robust, secure, and immutable.

To understand how a blockchain works, one needs to grasp the fundamentals of cryptography and consensus algorithms.

3.1 Cryptography

Cryptography plays a pivotal role in the correct functioning and security of the blockchain. Two vital cryptographic tools involved in the blockchain are hash functions and digital signatures.

3.1.1 Hash Functions

A hash function h is a mathematical function that unambiguously maps arbitrarily large words on an alphabet A of letters to words of fixed length n:

$$h : A^* \rightarrow A^n$$

Such a function therefore performs a compression action on messages, returning fixed-length outputs. Most cryptocurrencies use the alphabet *A={0,1}* and a value of *n* typically between 160 and 256.

A *collision* of *h* is a pair of distinct strings *(x, x')* over the alphabet such that *h(x)=h(x')*. The hash function *h* is called collision-resistant if it is infeasible to compute a collision of *h*. It is called preimage-resistant if given any fixed length string *s*, it is computationally infeasible to compute a string *x* such that *h(x)=s*. A *cryptographic hash function* is a hash function which is efficiently computable, preimage-resistant, and collision-resistant.

In a blockchain, hash functions are used to identify blocks and ensure transaction integrity. Before transactions are added to a block, they are hashed, and compiled into a *Merkle Tree,* which is a layered data structure, with each layer's values containing the combined hash of two of the hash values in the previous layer. The hashes are computed recursively until a single hash value (root) is obtained. The block's overall hash is computed from the root hash of the Merkle tree and the previous block's hash value.

A change to one of the transaction hash values in the Merkle tree would thus affect all the subsequent layers of the tree and so the block's overall hash, and therefore also the hash of all the blocks following it. Owing to the collision-resistance of the hash function, making all these changes is extremely computationally expensive. This property therefore protects the blockchain transactions from tampering. Further, each block contains a hash of the previous block's header. This ensures that each block is linked uniquely to the one preceding it and maintains the order of the blocks.

3.1.2 Digital signatures

A *digital signature* is a cryptographic tool used to provide authenticity, integrity, and non-repudiation to digital messages, thus serving as a digital equivalent of a handwritten signature.

Each digital signature involves a cryptographic private key and its corresponding public key. The private key is known only to the signer and is used to generate the signature. The corresponding public key is made available to others and is used to verify the digital signature.

In most cases, the signer first applies a hash function to the message to be signed, and then uses their private key to encrypt the hash value, thus creating the digital signature. The result is a unique, encrypted signature that is specific to both the content and the private key. For verification, the same hash function is applied to the message received, and the public key is used to decrypt and verify the digital signature. If the decrypted signature matches the generated hash value, the signature is valid.

Digital signatures play a crucial role in ensuring the security and authenticity of data within a blockchain by associating each transaction reliably to the node which initiated it. When a participant Each transaction is signed by the participant using their private key before it is broadcast to the network. Other participants can use the sender's public key to verify the digital signature. Once a transaction is verified and added to a block, the digital signature becomes an integral part of that block. Any attempt to alter the content of a block would therefore require recalculating the digital signatures for all subsequent blocks, which is computationally infeasible.

Most cryptocurrencies use the ECDSA (Elliptic-Curve Digital Signature Algorithm) with the elliptic curve “secp256k1”. Here, the secret key is 256 bits long and the public key is 512 bits long.

For a more detailed exploration of these concepts, as well as others in cryptography, the interested reader is referred to [15, 26].

3.2 Consensus Mechanism

A consensus mechanism is a protocol that allows nodes in a decentralized, trustless network to agree on the state of the blockchain. Once a transaction is confirmed through the consensus process and added to a block, it is considered irrevocable. Consensus mechanisms play a vital role in maintaining the integrity and security of the entire system.

In a public peer-to-peer system, it is imprudent to assume that every node always behaves innocuously and correctly. *Byzantine Fault Tolerance* is a property of a distributed system that allows it to continue operating reliably and reach consensus, even in the presence of a certain number of faulty or malicious nodes.

There are various types of consensus mechanisms, each with its own approach to achieving agreement among nodes. Below, we explain the two most used consensus mechanisms.

3.2.1 Proof of Work (PoW)

Under this consensus mechanism, nodes are called miners and compete to validate transactions and add new blocks to the blockchain. Bitcoin, the first and most well-known cryptocurrency, relies on PoW.

Proof of Work proceeds as follows. Transactions are first broadcast to the network and are collected and verified by miners. Miners then select a set of unconfirmed transactions to include in a new block. To add a new block, a miner must solve a computationally intensive mathematical puzzle. The puzzle is typically constructed using a cryptographic hash function and requires the miner to compute a candidate "nonce" which is a preimage of a certain subset of hash values satisfying specific predefined criteria. For example, in the case of Bitcoin, the criterion is that the hash value must have a certain number of leading zeros. There are multiple correct solutions to this puzzle, but to find any of them, the miner must iterate randomly through the set of all possibilities for the nonce (brute force).

The first miner to find a valid hash value broadcasts the new block to the network, along with the solution to the cryptographic puzzle (the nonce). Other nodes in the network verify the validity of the solution. Unlike the brute-force search for the nonce, the validation process is quick and efficient, as it requires just a single computation by each node. If it is correct, the nodes reach a consensus that this miner has the right to add the new block to the blockchain. As a reward for their effort and computational work, the successful miner receives some newly created cryptocurrency as a reward, (e.g., bitcoins) and any transaction fees from the transactions included in the block.

A crucial drawback of PoW is that it is extremely energy-intensive and wasteful by default, and therefore has a significant negative environmental impact.

3.2.2 Proof of Stake (PoS)

Proof of Stake (PoS) is an alternative consensus mechanism used in blockchain networks to achieve agreement on the state of the blockchain. In PoS, nodes are called validators. Unlike Proof of Work (PoW), where participants (miners) compete through computational work, PoS chooses validators to create new blocks and validate transactions based on the amount of cryptocurrency they hold and are willing to "stake" as collateral. The more cryptocurrency a validator stakes, the higher the chance they have to be selected to create a new block. The selection process may also include a randomization algorithm.

Transactions are broadcast to the network, and validators collect and verify these transactions. The chosen validator creates a new block, including a set of transactions, and signs it with their private key. Other nodes in the network then verify the validity of the block and the signature using the validator's public key. If the block is valid, it gets added to the blockchain, and the validator is rewarded with new cryptocurrency (block reward) and sometimes also transaction fees from the transactions in the block.

To discourage malicious behaviour, PoS systems often include a mechanism called slashing. If a validator is found to be acting dishonestly, such as by double-signing or attempting to create an invalid block, a portion of their staked cryptocurrency may be "slashed" or forfeited. Validators have a vested interest in acting honestly because malicious behavior risks losing a portion of their collateral funds.

One of the key advantages of PoS over PoW is its energy efficiency and faster transaction finality in the blockchain. Since PoS does not require the same level of computational work as PoW, it is far more environmentally friendly.

4 Applications of blockchain technology

In broad terms, the purpose of a blockchain is to help achieve and maintain the integrity of a distributed software system. More specifically, the blockchain is the tool used to create a secure, transparent, and decentralized system for recording, verifying, and transferring ownership of assets. Blockchain's features such as decentralization, immutability, and transparency provide trust in the integrity of the recorded data and contribute to the reliability of ownership management in diverse industries, including finance, real estate, supply chain, and more. This makes blockchains suitable for various applications beyond cryptocurrencies, including smart contracts, supply chain management, and decentralized finance.

Some prominent applications of blockchains are cryptocurrencies (e.g., Bitcoin, Ethereum, etc.), decentralized finance (borrowing, lending assets without a centralized intermediary like a bank), supply chain data management and tracking, healthcare data management and storage, voting systems, digital identity management, digital asset (e.g., digital art, intellectual property) management real estate data recording, and smart contracts. In the rest of this paper, we will be concerned only with smart contracts.

5 Smart Contracts

A smart contract is a self-executing tool with coded rules that automate and enforce the execution of contractual agreements, reducing the need for intermediaries in various industries, including legal, insurance, and real estate. As opposed to traditional legal contracts, smart contracts are written in formal code and interpreted by machines. They automatically execute and enforce themselves when predefined conditions are satisfied.

Smart contracts run on blockchain platforms and thus inherit the decentralized and immutable nature of the blockchain, thereby enhancing trust in the execution of agreements. For example, Ethereum is a blockchain platform that supports smart contracts. Smart contracts function as decentralized custodians of digital assets and allow the transfer of value and information without the need to trust a third party.

Smart contracts have numerous advantages including speed, independence of third parties, reliability, non-repudiation, and immutability. However, they also have drawbacks, like lack of regulation and governance, difficulty in making changes and fixing bugs, non-reversibility of transactions, and difficulty in implementation [19].

5.1 Ethereum

Smart contracts and their integration into blockchain technology were introduced by Ethereum with its mainnet launch on July 30, 2015 [7]. Ethereum is a decentralized blockchain platform that facilitates the creation and deployment of decentralized applications (DApps) and smart contracts. These contracts run on the Ethereum Virtual Machine (EVM), a decentralized runtime environment. Every node in the Ethereum network is responsible for upkeeping the network's state. This state encompasses a record associating each account address with its respective ether balance, nonce, storage, and program code.

Ethereum has its native cryptocurrency, Ether (ETH), used to pay for transaction fees, computational services by validators, and interactions with smart contracts on the Ethereum network. Gas is a unit used to measure the computational work or resources required to perform operations on the Ethereum network, such as executing smart contracts, sending transactions, or interacting with decentralized applications (DApps). Gas cost is denominated in ether. When one initiates a transaction or executes a smart contract, one specifies a gas limit (maximum amount of gas one is willing to spend) and a gas price (amount of ether one is willing to pay per unit of gas). The higher the gas price, the faster your transaction is likely to be processed.

Ethereum smart contracts are typically written in Solidity [27], a high-level programming language specifically designed for this purpose.

Ethereum has introduced various standards for tokens, which are digital asset representatives. The most important examples are ERC-20 for fungible tokens and ERC-721 for non-fungible tokens (NFTs). These standards establish consistent regulations and interfaces, facilitating the development and use of tokens on the Ethereum blockchain.

For a more thorough treatment of the Ethereum network, the interested reader is referred to [7].

5.2 Applications

Smart contracts are versatile and can be applied to various industries and use cases. They can be utilized to create and oversee tokens on blockchains, automate financial processes in DeFi, enforce and automate agreements in supply chain management, manage permissions in digital identity systems, and more. Two important and unique applications of smart contracts are decentralized autonomous organizations (DAOs) and decentralized applications (dApps).

A DAO is an autonomous digital entity that represents a new type of organizational structure [14]. In a DAO, rules and actions are encoded in smart contracts instead of being enforced by a small group of people, as is the case in a traditional company. DAOs enable decentralized decision-making through proposals and voting mechanisms. Members can submit proposals, and the community votes on whether to approve or reject them. DAOs operate on a blockchain, which means that they inherit its transparency and immutability. All actions, proposals, and transactions are recorded on the blockchain and can be audited by anyone. "The DAO" was a specific and influential project that existed on the Ethereum blockchain and aimed to function as a decentralized venture capital fund. However, The DAO is most famously remembered for a critical exploit that had significant consequences in the Ethereum community. We will discuss this exploit in detail later in this paper.

Decentralized applications (dApps) are software applications that operate on a peer-to-peer (P2P) network of computers, instead of a single computer. A dApp comprises a front-end, which is similar to a conventional web application, and a back-end, which is blockchain-based. The primary difference is that a dApp interacts with smart contracts that govern the back-end logic and with the blockchain data structure, rather than with a traditional database.

5.3 Deployment and Working

Smart contracts exist on a blockchain as transactions. In fact, in Ethereum, transactions are broadly of two types: regular transactions and smart contract transactions. Both types of transactions are subject to the consensus mechanism of the network. Deploying a smart contract is a one-time transaction that initializes the contract and makes it available for future interactions.

Before submission to the chain, the smart contract code is compiled into bytecode, which is a low-level representation that is executable by the virtual machine of the blockchain. All confirmed transactions added to the chain, whether regular or involving a smart contract, become final and irreversible. Unlike regular transactions, smart contract transactions can include invocations of specific functions within a deployed smart contract. Each such invocation is also treated as a separate transaction.

Developers of smart contracts use a programming language that is compatible with the underlying blockchain platform. For instance, Solidity for Ethereum. Apart from Ethereum, several other blockchain platforms also support smart contract functionality. Some examples are Binance Smart Chain (BSC), Cardano, Solana, Polkadot, and Algorand.

A general reference to better comprehend the workings of a smart contract is found in [29, 4].

6 Security of Blockchain and Smart Contracts

With the fundamental concepts, important terms, working principles, and some use cases of blockchains and smart contracts being explained, we now turn to the security aspects.

When we say that a blockchain is secure, it means that the blockchain system has measures in place to prevent unauthorized access, tampering, and attacks. Additionally, it ensures that the data and transactions are maintained with integrity, confidentiality, and availability. The security of blockchain is crucial for preserving the reliability, trustworthiness, and functionality of the technology.

Since smart contracts are built on blockchain technology, a disruption or attack on the underlying blockchain of a smart contract could also cause it to malfunction. While some attacks are more directly associated with the underlying blockchain infrastructure, the security and reliability of smart contracts depend on the robustness of the entire blockchain network.

In our discussion of security, we therefore first talk about the general security of a blockchain, including some common attack vectors and examples, and then later discuss the more specific aspects of smart contract security.

6.1 Attack Methods on a Blockchain

A general reference for this section can be found in [22].

51% Attack

In a proof-of-work blockchain, a 51% attack occurs when a single entity or a group of entities controls more than 50% of the network's mining or hashing power. Since hashing power is directly proportional to the likelihood of successfully adding new blocks to the blockchain, this attack gives the attackers majority control over the network's consensus mechanism, allowing them to manipulate transactions, double-spend, and potentially disrupt the normal functioning of the blockchain.

The consequences are multifaceted. The primary concern is the possibility of double-spending, where the attacker can spend the same cryptocurrency units in multiple transactions by creating a fork of the blockchain. This is achievable because the attacker controls the majority of the mining power and can outpace the rest of the network when adding new blocks. Apart from double-spending, the attacker can also perform block reorganization by disregarding the legitimate blockchain and produce their own version of the blockchain. Once their version exceeds the length of the honest blockchain, it becomes the accepted version. With majority control, the attacker can prevent certain transactions from being confirmed, exclude or delay specific transactions, and more.

There are several ways to prevent a 51% attack on a blockchain. One way is to encourage more independent miners and mining pools, which helps spread out the hashing power. This makes it harder for one entity to control the majority. The blockchain designer can also require more confirmations for transactions to be considered secure. This gives the network more time to detect and respond to a potential attack. Regular updates and network upgrades can also change the

consensus algorithm, making it more difficult for an entity to accumulate a majority of the hashing power. Another approach is to gradually switch to using Proof-of-Stake (PoS) or other consensus mechanisms that are less vulnerable to 51% attacks.

Sybil Attack

In a Sybil attack, a single attacker creates multiple fake nodes to take control of a significant portion of the network. This can enable the attacker to manipulate the consensus, or disrupt the intended behavior of a smart contract.

In decentralized networks that rely on consensus mechanisms like proof-of-work or proof-of-stake, attackers can influence the decision-making process by controlling a majority of nodes. This is especially concerning in systems with financial incentives, such as cryptocurrencies, where an attacker could use their control to exploit the system for financial gain. For example, they could manipulate transaction outcomes, game reward systems, or launch attacks to undermine the value of the cryptocurrency. If the network involves storing and retrieving data, controlling multiple nodes can allow an attacker to manipulate the data, which could lead to the corruption or alteration of the stored data. This could jeopardize the reliability of the entire system. Finally, attackers can partition the network by strategically controlling nodes. This can be used to isolate specific nodes or regions from the rest of the network, disrupting communication and creating isolated sub-networks.

To mitigate Sybil attacks, there are several strategies that can be implemented. For instance, a reputation system can be used to identify and discount the influence of suspicious nodes. Consensus mechanisms such as proof-of-work and proof-of-stake can also be employed to make the attack economically infeasible. Furthermore, requiring proof of a real-world resource or social validation mechanisms for the creation of a node can also be an effective approach to mitigate the risks of Sybil attacks.

Selfish Mining

Selfish mining is a strategy used by miners or mining pools with a lower hash rate. Instead of sharing the mined blocks with the network, they deliberately keep them hidden. This disrupts the normal block propagation process and gives the attacker an advantage in mining rewards. Selfish mining is mainly a risk to the mining process in proof-of-work blockchains.

Eclipse Attack

Selfish mining is a tactic employed by miners or mining pools with a lower hash rate. Instead of sharing the blocks they mine with the network, they intentionally keep them concealed. This upsets the standard block propagation process and grants the attacker an edge in mining rewards. Selfish mining is primarily a threat to the mining process in proof-of-work blockchains.

Distributed Denial of Service (DDoS) Attacks

A Distributed Denial of Service (DDoS) attack is a type of cyber attack that involves flooding a target system with a massive amount of traffic from multiple compromised computers or systems. This flood of traffic overwhelms the target system, making it unavailable or significantly reducing its

performance. In the context of blockchain, DDoS attacks can be particularly disruptive to the underlying blockchain network and decentralized applications (DApps) built on top of it. DDoS attacks can target nodes, mining pools, or other components of the blockchain network, causing disruptions that make it difficult for legitimate users to access or participate in the network.

Cryptographic Threats

The security of a blockchain is dependent on the effectiveness of the cryptographic protocols it employs. If there are any vulnerabilities in the applied cryptography tools, then those same issues will be present in the blockchain. A potential example would be the possibility of quantum computers undermining the security of current cryptographic algorithms used in blockchain systems.

6.2 Attack Methods on a Smart Contract

As discussed earlier, the security and reliability of the blockchain network is crucial for the proper functioning and safety of smart contracts built on it. Therefore, any vulnerability in the network can also affect the security of smart contracts. In addition to this, smart contracts can have errors and loopholes in their code, which can be exploited by attackers to launch various types of attacks, leading to unexpected results and potential loss of funds. Even minor logical errors in the smart contract code can cause significant damage, making it important to thoroughly test and review the code before deployment.

Below, we discuss in detail some of the best-known vulnerabilities associated with smart contracts. A general reference for this section can be found in [19].

6.2.1 Reentrancy Attacks

Smart contracts often interact with each other by calling functions in other contracts. When contract *A* calls a function in contract *B*, this is known as an external call.

A procedure is considered re-entrant if it can be interrupted during its execution, resumed, and completed without any errors. Reentrancy happens when a function in a contract that makes external calls can be maliciously re-invoked before the current execution of state changes is completed. This enables attackers to repeatedly execute certain functions and tamper with the contract's state in unintended ways. A reentrancy attack is, therefore, a manipulation of the order of execution and the way state changes and gas are handled.

As an example within a centralized system, consider an online banking system that checks account balance only at the initialization step. This would allow a user to initiate several transfers without submitting any of them. The banking system would confirm that the user’s account holds a sufficient balance for each transfer. If there was no additional check at the time of the actual submission, the user could then submit all transactions and potentially exceed the balance of their account.

6.2.1.1 REAL-WORLD EXAMPLE—THE DAO HACK

The DAO [14] was a project for decentralized investment crowdfunding, based on smart contracts on the Ethereum blockchain. The smart contract allowed a funding project to withdraw Ether if it received adequate support from the DAO community. However, there was a flaw in the transfer mechanism, whereby the ether would be transferred to the external address before updating its internal state and noting that the balance had already been transferred.

The DAO contract included a fallback function, which is a function that is executed when a contract receives Ether but the transaction doesn't match any of its existing function signatures (or whenever a transaction attempts to call a method that the smart contract does not implement). The fallback function was designed to allow DAO token holders to withdraw their Ether.

In 2016, an attacker exploited this vulnerability to drain approximately 3.6 million Ether (ETH) from The DAO [1] by using a malicious contract to repeatedly call the DAO's fallback function before the state changes (e.g., updating the balance) were finalized. This incident led to a contentious hard fork, resulting in the creation of Ethereum (ETH) and Ethereum Classic (ETC) as separate blockchains with the former following the fork and the latter maintaining the original, unaltered blockchain.

To prevent reentrancy attacks, code developers should ensure that state changes are made before interacting with external contracts and include a Boolean variable in the code to serve as a reentrancy guard.

6.2.2 Integer Overflow and Underflow

An integer overflow occurs when the result of an arithmetic operation exceeds the maximum representable value for the given integer data type. For example, the maximum representable value by 8-bit unsigned integer is 255. If one tries to add 1 to 255, the result, 256, results in an overflow, and the value wraps around to 0. An integer underflow occurs when the result of an arithmetic operation goes below the minimum representable value for the given integer data type. For example, the minimum representable value by 8-bit signed integer is -128. If one tries to subtract 1, the result, -129, results in an underflow, and the value wraps around to 128.

Improper handling of arithmetic operations can result in integer overflow or underflow, leading to unexpected values, manipulation of transactions, and system crashes. For example, if a spender convinces the account owner to send a transfer that decreases the spender’s allowance below 0, this would lead to an integer underflow and the spender receives an astronomical allowance that will likely enable them to withdraw the entire balance of the account.

6.2.2.1 Real-world Example: BatchOverFlow

In April 2018, a security breach known as "BatchOverflow" [6] occurred on the ERC-20 token contract of the BEC (BeautyChain) token. The BEC token contract's transferFrom function was designed to enable third parties (such as exchanges) to transfer tokens on behalf of a user with their approval. However, the function had an integer underflow issue when handling the value parameter. The attacker exploited this vulnerability by creating a malicious contract and manipulating the value

parameter to underflow, resulting in a large value. As a result, the smart contract credited a significant number of tokens to the attacker's account, as it failed to check for the underflow.

To prevent integer underflows and overflows, it's essential to validate the results of arithmetic operations. For instance, when adding two unsigned integers a and b, the result a + b should be greater than or equal to a. If there's an overflow, this property is violated. Some programming languages and frameworks have libraries or functions that perform arithmetic operations securely, preventing overflows and underflows.

6.2.3 TRANSACTION-ORDERING DEPENDENCE (TOD) AND FRONT RUNNING

Recall that the order of processing transactions on Ethereum is determined by miners. Before a transaction is mined, it is visible in the "mempool" to all miners. Furthermore, in decentralized systems like Ethereum, pending transactions are visible to the public before they are confirmed and added to a block. These transactions often include information about the sender's intention to perform a trade, such as buying or selling a specific number of tokens.

Front running is an unethical practice where someone takes advantage of confidential information about upcoming transactions to gain an unfair advantage. This occurs when someone takes advantage of the time delay between submitting a transaction and it being included in a block. The attacker can change the order of transactions to their benefit, which could impact the execution of a smart contract.

For example, consider a situation where an investor submits a buy order for a certain asset at a certain price. Before this transaction is included in a block, an attacker who has access to confidential information about the investor's trade submits their own buy order at a slightly higher price. This causes the investor's transaction to be executed at a higher price than they intended, resulting in a loss for them and a profit for the attacker.

Front running remains a challenging issue to mitigate in decentralized environments. Efforts to diminish front running include the development of decentralized exchanges (DEXs) and protocols that incorporate mechanisms to minimize the impact of this practice, such as batched transactions or using algorithms that obscure the true intentions of users until the last possible moment.

6.2.3.1 Real world example: Bancor Front-Running Attack (June 2018):

Bancor [5] is a decentralized liquidity protocol that operates on the Ethereum blockchain. It enables users to trade tokens through smart contracts without relying on a traditional order book. The pricing of tokens is determined algorithmically based on a formula that takes into account the token balances in smart contract reserves. However, in 2018, an attacker took advantage of the algorithmic pricing mechanism by placing multiple large transactions in quick succession. This manipulation of token prices was done to profit from the price changes caused by their own transactions, resulting in a significant disruption to the Bancor network. The time delay between the initiation of a transaction and its inclusion in a block was exploited, which affected the prices of some tokens.

6.2.4 Timestamp Dependence

Smart contracts often rely on the timestamp of a block to carry out certain actions or make decisions. For instance, a contract may have a function that unlocks particular features or releases funds after a certain period of time. Exploitation of the timestamp in smart contracts happens when the timestamp is manipulated within the blockchain.

The timestamp of a block in a blockchain is set by the miner who successfully mines the block. The timestamps must be increasing and cannot be too far into the future, or they will be rejected by the network nodes. Although the timestamp should represent the approximate time when the block was mined, it is not entirely tamper-proof. Miners have some flexibility in setting the timestamp, and it is expected to be within a certain range of the current time. Thus, miners who have incentives may manipulate timestamps. They may manipulate timestamps for financial gain, such as front running, actively excluding transactions of other investors, or delaying the execution of time-sensitive functions.

Some mitigation strategies against this vulnerability class are to use relative time measurements like block numbers or block intervals along with timestamps, and to integrate with external oracles providing accurate and tamper-resistant timestamps.

6.2.5 Denial of Service (DoS) Attacks

A Denial of Service (DoS) attack on smart contracts involves malicious actors attempting to disrupt the normal operation of a smart contract or of the entire blockchain network. DoS attacks aim to overwhelm the resources of the targeted system, making it temporarily or permanently unavailable.

There are several routes an attacker may take to achieve DoS. The most prominent method is through gas exhaustion, where an attacker deploys a smart contract that performs resource-intensive computations, creating a situation where users must pay an exorbitant amount of gas fees to interact with the contract. If a contract's logic is too complex or resource-intensive, it may exceed the gas limit, causing the transaction to fail. Another method to implement a DoS is through reentrancy. An attacker can deploy a contract that exploits vulnerabilities in another contract, causing it to enter an infinite loop or consume excessive gas during execution. This can lead to a denial of service by consuming all available gas in the network.

Some methods of mitigating DoS attacks on smart contracts include the implementation of gas limits, the use of secure coding patterns, and thorough testing and audits to identify and address potential vulnerabilities. Additionally, it is essential to implement network-level solutions like congestion control mechanisms and gas price controls, which help to protect the broader blockchain network from DoS attacks.

6.2.6 Faulty Random Number Generation

Smart contracts require the use of random numbers for various purposes, like the generation of cryptographic keys, nonces, unique identifiers, salts, and initialization vectors, for selecting random

winners in lotteries and contests, shuffling lists, in gaming and gambling, and for determining the order of transactions in dApps. Thus, the proper generation of random numbers is crucial for ensuring fairness, security, and unpredictability. Using insecure random number generation methods can allow attackers to predict outcomes, leading to manipulations and exploits.

However, by design, all computations on Ethereum must be deterministic, otherwise, the consistency of the state of the blockchain may be compromised. This means that if the same transactions are included in a block and the block is mined by the same miner, the resulting block hash will be the same for all nodes. Thus, pseudo-random number generation is used instead. Developers often attempt to use information related to the state of the blockchain for pseudorandom number generation. However, some block information, like block hash, can be partially predictable before a block is mined. If this information is the basis for randomness, an attacker might attempt to manipulate the block hash to predict the random number.

6.2.6.1 Real World Example: Etheroll

The "Etheroll" incident [25] on the Ethereum blockchain in 2017 exemplifies this vulnerability. Etheroll was a decentralized application (dApp) on the Ethereum blockchain that allowed users to participate in a dice game, with outcomes determined by a random number generator. The dApp utilized a random number generator based on the block hash of the Ethereum block in which the bet was placed. The attacker was able to influence the transactions included in a block, thereby increasing the likelihood of specific block hashes. The attacker thereby successfully predicted the block hashes, manipulated the random number generator to their advantage, and was able to consistently win bets, resulting in financial gains, leading to financial losses for Etheroll and its users.

Since the Etheroll incident, various solutions and improvements have been proposed to enhance the security of random number generation in smart contracts. More robust randomness generation methods are encouraged, such as decentralized oracles, commit-reveal schemes, or cryptographic techniques. Some notable developments are Chainlink VRF (Verifiable Random Function), DECO (Decentralized Coin Toss), Kyber Network's Katalyst protocol upgrade, and the use of external Oracles and APIs.

6.2.7 External call vulnerabilities

Many smart contract vulnerabilities stem from the ability of smart contract functions to be triggered by other smart contracts, by what are known as external calls. These calls can involve sending funds, invoking functions in other contracts, or interacting with external services.

Sometimes, a smart contract might make external calls without implementing proper checks or validations. This can create security issues, particularly if the external calls involve interacting with potentially malicious or untrusted contracts. For instance, if a contract assumes that an external call was successful without verifying the return status, it may result in unintended consequences. It is important to ensure that proper validations and checks are implemented when making external calls to prevent such security risks.

6.2.7.1 Real World Example: King of Ether

One notable real-world example of a smart contract exploit resulting from a lack of external call validation is the "King of the Ether" incident of 2016 [16]. King of the Ether [17] was a decentralized game on the Ethereum blockchain where participants could become the "king" by sending Ether to a smart contract. The participant with the highest bid would become the new king, and they could increase their bid to maintain their status.

The smart contract used a mechanism where the current king could transfer the crown (ownership) to another Ethereum address. However, the smart contract did not properly validate whether the recipient address was a valid contract or an externally owned account. Additionally, the function involved did not revert the state for failed transactions. When the contract attempted to compensate the previous king, it would mistakenly execute its fallback function. This allowed a new king to assume the throne without compensating the old king.

An attacker took advantage of a flaw in the validation process and created a contract that could receive the crown. The malicious contract then launched a reentrancy attack by invoking the fallback function of the original King of the Ether contract before the state was updated. As a result, the attacker's contract was able to continuously call back into the original contract, repeatedly claiming the crown and draining Ether from the contract.

6.2.8 Vulnerable Library Usage

Certain functionalities of Ethereum contracts, such as on-chain data structures, token contract interfaces, and multi-signature wallets, can be reused in other contracts. These collections of functionalities, also known as libraries, are deployed to the blockchain once and then referenced multiple times by other smart contracts known as "client" contracts. It is important to note that if a library has a security vulnerability, all instances of the client contract using that library are exposed to risk. Additionally, because of the immutability of the blockchain, it is not possible to fix a vulnerable library by redeploying it to the same address. Some client contracts may also be hard-wired to the flawed library version, without an option to switch to a different library version.

6.2.8.1 Real World Example: the Parity multi-signature wallet hacks

There are two well-known examples of library exploits, namely the Parity multi-signature wallet hacks of 2017 [21]. The first vulnerability enabled unknown parties to access wallet contracts and transfer funds from the wallets, allowing for unexpected wallet ownership transfers. The second exploit occurred after the library was updated. Even after it was deployed, the update remained uninitialized, which made it possible to initialize the library and set its owner.

The library's stateful nature was a major cause of this exploit. However, it is unnecessary for a library to have its own state, as the client's state is updated whenever a library is called. Therefore, designing libraries to be stateless is a security best practice. Moreover, using established libraries and frameworks can significantly reduce the risk of smart contract library vulnerabilities.

7 Writing secure smart contracts

As discussed in the previous section, attackers may exploit several routes to compromise the functionality and security of a smart contract. While each attack vector may have individual approaches for an optimal mitigation strategy, developers may adopt a set of fundamental overarching approaches to avoid security vulnerabilities and to ensure the correct functioning of the code in accordance with the requirements of the project.

1. Understand previous attacks and existing sources of vulnerabilities and follow the latest best practices and standard design patterns to avoid them.
2. The smart contract code should be written cleanly and documented properly and thoroughly
3. The code should be thoroughly tested through a multi-layer approach prior to deployment, and all issues arising from tests should be appropriately addressed. This may include the use of automated analysis tools, as well as performing manual internal and external audits. Some well-known smart contract analysis tools are Oyente, Maian, Mythril, and Securify. Manual audits are essential to verify the logic of complex contract code.
4. Every dApp should implement strong authentication principles.

For a more in-depth explanation of these best practices, the interested reader is referred to [19, 29].

8 References

[1] A $50 Million Hack Just Showed that the DAO was All Too Human. (n.d.). *Wired*. Retrieved on 21 January 2024 from https://wired.com/2016/06/50-million-hack-just-showed-dao-human

[2] Antonopoulos, A. M. (2018). Mastering Ethereum: Building Smart Contracts and DApps. O'Reilly Media.

[3] Accredited Standards Committee X9 Incorporated. (2005). *Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA).* Accredited Standards Committee X9 Incorporated.

[4] Atzei, N., Bartoletti, M., & Cimoli, T. (2017). *A Survey of Attacks on Ethereum Smart Contracts SoK*. *Proceedings of the 6th International Conference on Principles of Security and Trust*, 164–186. Retrieved on 21 January 2024 from https://eprint.iacr.org/2016/1007.pdf

[5] Bancor V3. Retrieved on 21 January 2024 from https://docs.bancor.network/about-bancor-network/bancor-v3

[6] BatchOverflow Exploit Creates Trillions of Ethereum Tokens, Major Exchanges Halt ERC20 Deposits. (n.d.). *Crypto Slate*. Retrieved on 21 January 2024 from https://cryptoslate.com/batchoverflow-exploit-creates-trillions-of-ethereum-tokens-major-exchanges-halt-erc20-deposits

[7] Buterin, V. (2013). *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. Retrieved from https://github.com/ethereum/wiki/wiki/White-Paper

[8] Consensys. *Known Attacks.* Retrieved on 21 January 2024 from https://consensys.github.io/smart-contract-bestpractices/known_attacks

[9] Consensys. *Recommendations for Smart Contract Security in Solidity.* Retrieved on 21 January 2024 from https://consensys.github.io/smart-contract-best-practices/recommendations

[10] Deloitte Insights. (2016, June 08). *Upgrading Blockchains: Smart Contract Use Cases in Industry.* Retrieved on 21 January 2024 from https://www2.deloitte.com/insights/us/en/focus/signals-for-strategists/using-blockchain-for-smart-contracts.html

[11] Drescher, D. (2017). *Blockchain Basics: A Non-Technical Introduction in 25 Steps*. Apress.

[12] ERC20 Tokens: A Comprehensive Origin Story. (2018). *Blockgeeks*. Retrieved on 21 January 2024 from https://blockgeeks.com/guides/erc20-tokens

[13] Hertig, A. (n.d.). *How Do Ethereum Smart Contracts Work? CoinDesk*. Retrieved on 21 January 2024 from https://coindesk.com/information/ethereum-smart-contracts-work

[14] Hertig, A. (n.d.). *What is a DAO? CoinDesk*. Retrieved on 21 January 2024 from https://coindesk.com/information/what-is-a-dao-ethereum

[15] Katz, J., & Lindell, Y. (2014). *Introduction to Modern Cryptography* (2nd ed.). Chapman & Hall/CRC.

[16] King of Ether. (n.d.). *King of the Ether.* Retrieved on 21 January 2024 from https://kingoftheether.com/thrones/kingoftheether/index.html

[17] King of Ether. (February 2016). *Post-Mortem Investigation.* Retrieved on 21 January 2024 from https://kingoftheether.com/postmortem.html

[18] Luther, W., & Schwartz, R. A. (2019). *Decentralized Autonomous Organizations: Blockchain-Based Organizations from Code*. Journal of Management Information Systems, 36(3), 734-770. https://doi.org/10.1080/07421222.2019.1626341

[19] Ma, R., Gorzny, J., & Zulkoski, E. (2019). *Fundamentals of Smart Contract Security*. Momentum Press.

[20] Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. Retrieved on 21 January 2024 from https://bitcoin.org/bitcoin.pdf

[21] Parity Technologies. (2017). *A Postmortem on the Parity Multi-Sig Library Self-Destruct.* Retrieved on 21 January 2024 from https://parity.io/a-postmortem-on-the-parity-multi-sig-library-self-destruct

[22] Poston, H. E., III. (2022). *Blockchain Security from the Bottom Up: Securing and Preventing Attacks on Cryptocurrencies, Decentralized Applications, NFTs, and Smart Contracts*. Wiley. ISBN 9781119898634.

[23] Qian, P., He, J., Lu, L., Wu, S., Lu, Z., Wu, L., Zhou, Y., & He, Q. (25 Apr 2023). *Demystifying Random Number in Ethereum Smart Contract: Taxonomy, Vulnerability Identification, and Attack Detection*. Retrieved from https://arxiv.org/abs/2304.12645.pdf

[24] Ream, J., Chu, Y., & Schatsky, D. (2016, June 08). *Upgrading Blockchains: Smart Contract Use Cases in Industry. Deloitte Insights*. Retrieved on 21 January 2024 from https://www2.deloitte.com/insights/us/en/focus/signals-for-strategists/using-blockchain-for-smart-contracts.html

[25] Reutov, A. (n.d.). *Predicting Random Numbers in Ethereum Smart Contracts*. Retrieved 21 January 2024 from https://blog.positive.com/predicting-random-numbers-in-ethereum-smart-contracts-e5358c6b8620

[26] Silverman, J.H., Pipher, J., & Hoffstein, J. (2008). An Introduction to Mathematical Cryptography. Springer. https://doi.org/10.1007/978-0-387-77993-5

[27] Solidity. GitHub. Retrieved on 21 January 2024 from https://github.com/ethereum/solidity

[28] Solidity Front Running Attack (2023). Retrieved on 21 January 2024 from https://neptunemutual.com/blog/solidity-front-running-attack/

[29] Solorio, K., Kanna, R., & Hoover, D. (2019). *Hands-On Smart Contract Development with Solidity and Ethereum: From Fundamentals to Deployment*. O'Reilly Media.