SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY
INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Data Engineering and Analytics

Analysis of the SUAVE Architecture, Mechanisms and Use-Cases

Jonas Gebele

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY
INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Data Engineering and Analytics

Analysis of the SUAVE Architecture, Mechanisms and Use-Cases

Analyse der Architektur, Mechanismen und Anwendungsfälle von SUAVE

Author:	Jonas Gebele
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Burak Öz
Submission Date:	15.07.2024

I confirm that this master's thesis in data engineering and analytics is my own work and I have documented all sources and material used.

Munich, 15.07.2024 Jonas Gebele

Acknowledgments

The completion of this thesis was made possible through the support and guidance of several individuals. I am grateful to my advisor, Burak Öz, for accepting this thesis topic and providing valuable feedback throughout the research process. I thank Prof. Dr. Florian Matthes for the opportunity to conduct this thesis under his chair. Acknowledgment is due to my family for their support during my academic studies. Their contributions have been significant in the realization of this work.

Abstract

This thesis analyses SUAVE (Single Unifying Auction for Value Expression), a novel blockchain protocol designed to address challenges around Maximal Extractable Value (MEV) extraction, privacy preservation, and scalability in blockchain systems. The research analyzes SUAVE's architecture, development process, and potential applications, focusing on its use of Trusted Execution Environments (TEEs) for confidential computing and verifiable off-chain computation.

The thesis employs a comprehensive analysis of SUAVE's technical documentation, source code, and proposed use cases. It examines the protocol's implementation roadmap, core components, and development toolkit for SUAVE applications (SUAPPs). Additionally, the research explores potential applications in areas such as Sybil attack mitigation, decentralized finance, and privacy-preserving machine learning.

Findings reveal SUAVE's potential to decentralize processes that currently rely on trusted centralized entities for temporal confidentiality. The protocol's architecture demonstrates promise in executing confidential computations while maintaining blockchain integrity. However, the research also identifies areas requiring further development, including TEE security, consensus mechanisms, and governance processes.

This thesis concludes that SUAVE represents a significant advancement in blockchain technology, offering innovative solutions to centralization in the MEV supply chain and enhancing privacy and scalability. The study contributes to the understanding of emerging TEE-based blockchain infrastructures and their potential impact on decentralized systems, while also highlighting areas for future research and development in the field.

Contents

1 Introduction

1.1 Problem Statement and Motivation

The blockchain ecosystem, particularly Ethereum, has been grappling with a phenomenon known as Maximal Extractable Value (MEV) that has emerged as a significant challenge to the principles of decentralization and fairness. MEV refers to the maximum value that can be extracted from block production in excess of the standard block reward and gas fees by including, excluding, or reordering transactions within a block. This concept, first articulated in the "Flashboys 2.0" paper by Daian et al., has evolved from a theoretical concern to a practical reality that shapes the economic landscape of blockchain networks [1].

Initially, MEV was primarily exploited by a small group of sophisticated actors known as MEV searchers. These individuals or entities employ complex algorithms and high-frequency trading strategies to identify and capitalize on profitable transaction ordering opportunities. While this activity has created new revenue streams for miners and searchers, it has also introduced a host of issues that threaten the integrity and usability of blockchain networks [1].

The implications of MEV extend far beyond simple profit-making. It has led to increased network congestion as searchers compete to include their transactions, often resulting in gas price wars that drive up transaction costs for all users. This not only degrades the user experience but also raises concerns about the accessibility of blockchain networks, potentially excluding users who cannot afford the inflated fees [1] [2].

Moreover, the pursuit of MEV has introduced significant centralization pressures within the blockchain ecosystem. Actors with superior computational resources, low-latency network connections, or privileged access to transaction information gain a substantial advantage in MEV extraction. This trend towards centralization stands in stark contrast to the decentralized ethos that underpins blockchain technology, potentially undermining one of its core value propositions [1] [2].

In response to these mounting challenges, a whole supply chain of MEV extraction has emerged, including specialized roles such as searchers, builders, and relays. Flashbots[1], a research and development organization, has been at the forefront of efforts to address MEV-related issues. Their initial solutions aimed to democratize MEV extraction and mitigate its negative externalities. However, these early approaches, while beneficial, have not fully resolved the underlying issues. Instead, they have introduced new

[1] `https://www.flashbots.net`

concerns about centralization at different points in the MEV extraction supply chain, particularly in block production and orderflow access [3].

It is within this context that SUAVE (Single Unifying Auction for Value Expression) has been proposed as a more comprehensive solution. SUAVE aims to fundamentally reconfigure the roles of the mempool and block builder in existing blockchains, promising a decentralized approach of handling MEV. More than just addressing centralization issues, SUAVE is designed as a platform for decentralizing various points in the MEV extraction supply chain as MEV applications. Crucially, SUAVE seeks to replace trust in centralized entities within this supply chain with trust in cryptographic systems. By leveraging advanced cryptographic techniques, SUAVE aims to provide the security and integrity previously offered by centralized forces, but in a decentralized and trustless manner [4].

SUAVE is an evolving protocol with the potential to foster innovation in areas previously constrained by centralization. However, as with any nascent technology, its effectiveness in real-world scenarios and its ability to truly decentralize MEV extraction and related processes remain subjects of critical inquiry. This thesis aims to explore these aspects, contributing to our understanding of SUAVE's potential impact on the blockchain ecosystem. Additionally, we will investigate possible use cases enabled by SUAVE's capabilities.

1.2 Research Objectives

This thesis aims to critically examine SUAVE's architecture, development toolkit, and potential applications through the following research objectives:

1. **Analyze SUAVE's architecture and mechanisms:** This objective involves a thorough examination of SUAVE's core components, their interactions, and the underlying principles that govern their operation. The analysis will explore how these elements work together to create a protocol with distinctive capabilities, including SUAVE's approach to confidential computing and its implementation of TEEs.

2. **Evaluate the toolkit and resources for SUAVE application development:** This objective focuses on assessing the development environment and tools provided for creating SUAVE Applications (SUAPPs). The evaluation will consider the functionality, user-friendliness, and effectiveness of these tools in enabling developers to harness SUAVE's unique features, encompassing the overall development process for SUAPPs.

3. **Explore potential applications enabled by SUAVE:** This objective aims to investigate the new types of applications that SUAVE's unique properties could enable. It involves analyzing how SUAPPs can address existing challenges and create novel solutions in areas such as decentralized finance (DeFi), privacy-preserving computations, and cross-chain interactions. The exploration will consider the

technical feasibility, potential benefits, and limitations of implementing SUAPPs in various scenarios, focusing on the distinctive capabilities that SUAVE as a protocol brings to these applications.

Through these research objectives, this thesis seeks to provide a comprehensive understanding of SUAVE as an evolving protocol, examining its architectural design, development ecosystem, and the new possibilities it presents for decentralized applications.

1.3 Thesis Structure

Chapter 2 lays the groundwork by introducing Ethereum fundamentals, including its virtual machine, smart contracts, and decentralized exchanges. It also explains the concept of MEV and the Proposer-Builder-Separation (PBS) model. This background is essential for understanding the context and challenges that SUAVE aims to address.

Chapter 3 explores various Privacy Preserving Smart Contract (PPSC) schemes, analyzing their implications, restrictions, and trade-offs for confidential computation in blockchain environments. This comparative study informs the understanding of SUAVE's design decisions and architectural choices, particularly its use of TEEs. The chapter provides critical context for the technical foundations upon which SUAVE is built.

Chapter 4 forms the core of the thesis, presenting an in-depth analysis of SUAVE's design, implementation roadmap, and key mechanisms. It covers the Rigil and Sirrah implementations, exploring unique features such as the SUAVE Chain, MEVM, and Kettles, providing a comprehensive understanding of SUAVE's operation and potential.

Chapter 5 focuses on the practical aspects of SUAVE application development. It examines the development environment, SUAVE Standard Library, and precompile development process, offering insights into the tools and resources available for creating SUAVE applications.

Chapter 6 demonstrates SUAVE's versatility by exploring potential use cases, including combating Sybil attacks in airdrops, integrating centralized exchange liquidity with DeFi, and enabling machine learning on confidential data. This chapter illustrates the practical implications and potential impact of SUAVE technology.

Chapter 7 presents a discussion of the findings, synthesizing the insights gained from the analysis. It critically evaluates the strengths and limitations of SUAVE in addressing the challenges identified.

2 Background

This chapter provides essential background information on blockchain technology, with a focus on Ethereum and its ecosystem. It covers fundamental concepts, key innovations, and emerging challenges in the field. Understanding these elements is crucial for appreciating the context in which SUAVE operates and the problems it aims to solve. The topics discussed here form the foundation for the analysis of SUAVE in subsequent chapters.

2.1 Ethereum Basics

With ubiquitous internet connections in most places of the world, global information transmission has become incredibly cheap. Technological innovations like Bitcoin have proven the feasibility of creating a decentralized value-transfer system that operates on a global scale with minimal costs. Such systems represent a specialized form of a cryptographically secure, transaction-based state machine [5].

Proposed by Vitalik Buterin in late 2013, Ethereum builds upon the idea set forth by Bitcoin by offering a more generalized technological platform. Unlike Bitcoin, which is designed primarily for peer-to-peer money transfers, Ethereum serves as a foundation for all types of transaction-based state machine concepts. It integrates a Turing-complete programming language, enabling the development of sophisticated contracts and decentralized applications directly on the blockchain. This capability facilitates not just transactions but also complex contractual agreements and autonomous agent operations within a trustful object messaging compute framework [6].

The inception of Ethereum was influenced by several prior works, including Dwork and Naor's concept of cryptographic proof of computational expenditure [7], Nakamoto's introduction of Bitcoin[1], and projects like Namecoin[2], Mastercoin (now OmniLayer)[3], and Colored Coins[4], which extended Bitcoin's protocol for various applications. Ethereum's primary goal is to facilitate secure and transparent interactions between parties without mutual trust. This is achieved through a state-change system that is defined by a rich programming language and an architecture that autonomously enforces agreements. Such a system ensures incorruptibility and transparency, qualities often absent in traditional systems managed by humans [6].

[1] https://bitcoin.org/bitcoin.pdf
[2] https://www.namecoin.org
[3] https://www.omnilayer.org
[4] https://bitcoinwiki.org/wiki/colored-coin

At its core, Ethereum operates as a transaction-based state machine, commencing from a genesis state and progressing through the execution of transactions to define the current state. This state is recognized as the official current state of Ethereum, with transactions grouped into blocks that are cryptographically linked to form a blockchain. To encourage computation within the network, Ethereum utilizes a native currency called Ether (ETH), where the smallest unit is Wei (1 ETH = 10^{18} Wei). Transactions and computational services within Ethereum are paid for with "gas", a unit that measures the amount of computational effort required [8]. The Ethereum world state is maintained as a mapping between addresses and account states, managed in a modified Merkle Patricia tree. This structure enables secure and efficient verification of previous states by altering the root hash. Each account state comprises the nonce, balance, storage root, and code hash, facilitating integrity and immutability [8]. In Ethereum, a transaction is a cryptographically-signed instruction by an external actor. These transactions delineate specifics such as type, nonce, gas limit, recipient address, value, and signature components. Ethereum supports various types of transactions including legacy, EIP-2930, and EIP-1559, each catering to specific functionalities [8]. Ethereum initially adopted proof-of-work (PoW) to secure consensus but successfully transitioned to proof-of-stake in 2022. The proof-of-stake (PoS) model not only decreases energy consumption significantly but also employs a more economically incentivized structure that enhances network security by aligning validator incentives with network health [9].

Through its Turing-complete language, Ethereum not only processes transactions but also executes complex contracts and decentralized applications. This broader capability allows developers to encode arbitrary state transition functions directly into the blockchain, enabling a vast range of applications from digital registries and autonomous organizations to complex financial instruments [6].

2.2 Ethereum Virtual Machine

The Ethereum Virtual Machine (EVM) serves as the computational core of Ethereum, facilitating the execution of smart contracts across its decentralized network. As a quasi-Turing-complete machine, the EVM processes smart contracts with high efficiency and security. The gas mechanism plays a crucial role in the EVM's operation by limiting the computational resources available for each transaction, thereby mitigating potential abuse [8].

The EVM operates on a stack-based architecture capable of holding up to 1024 elements, each 256 bits wide. This design is optimal for cryptographic operations and enables the execution of complex smart contracts. The EVM's isolated execution environment ensures that transactions are processed securely without any unauthorized access to the external system, preserving the integrity of operations across the network.

Data within the EVM is managed through two types of storage:

- **Volatile Memory:** Cleared at the end of every transaction, this temporary storage

facilitates transaction processing.

- **Non-Volatile Storage:** Maintains state across transactions, crucial for persistent data and smart contracts.

Each operation in the EVM incurs a cost in gas, proportional to its computational complexity. This gas system prevents excessive computation by imposing a cap on the amount of work a transaction can perform, safeguarded by the gas limit specified in each transaction. This structure ensures network efficiency and protects against infinite loops and other forms of resource abuse [8].

2.2.1 Ethereum Execution Clients

An Ethereum node consists of two main components: the execution client and the consensus client. The execution client is primarily responsible for transaction processing and state management, supporting the core functionalities of the EVM [10].

Historically, execution clients were sufficient to operate a full Ethereum node. However, with Ethereum's transition from PoW to PoS, the architecture evolved. Execution clients now require integration with consensus clients to manage blockchain operations effectively. This integration allows execution clients to focus on transaction execution and state changes, while consensus clients handle block production and maintain consensus protocols [11].

The two clients communicate via a local RPC connection using the engine API. This setup allows efficient management of transactions and blocks [11]:

- **Execution Clients:** Manage local transaction pools and connect to a peer-to-peer network for transaction dissemination. They create and execute transaction payloads, ensuring compliance with Ethereum's rules.
- **Consensus Clients:** Operate on a separate P2P network, handling block and consensus information. They also interact with validators that propose blocks and attest to block validity, which is critical for the network's security and integrity.

Figure 2.1 illustrates the architecture of an Ethereum node after the transition to Proof-of-Stake, showing the interaction between the execution and consensus clients.

Go-Ethereum (Geth)

Go-Ethereum, commonly known as Geth, is a popular implementation of an Ethereum execution client. Geth integrates the EVM to execute smart contract code, playing a vital role not just in transaction processing but also in managing the blockchain's state and its interaction with the network [13].

Geth operates in conjunction with consensus clients, a collaboration that became particularly vital following Ethereum's shift from PoW to PoS. Geth handles the execution of transactions and smart contracts, while consensus clients are responsible for

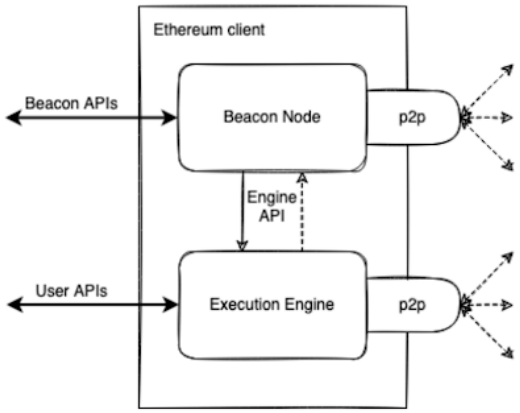

Figure 2.1: Ethereum Node Architecture after PoS Transition [12]

network consensus, block validation, and chain finalization, ensuring network security and operational efficiency [8].

As of June 2024, Geth is the predominant Ethereum client used by node operators, with 3,319 nodes (47.80% of all nodes) [14].

2.2.2 Precompiled Contracts in Ethereum

Precompiled contracts in Ethereum are crucial optimizations within the EVM designed to perform computationally intensive tasks efficiently. These contracts are positioned at fixed addresses within the blockchain, enabling direct access to essential cryptographic functions that would otherwise be too gas-expensive if implemented in standard smart contracts [8].

Precompiled contracts are located at specific addresses, starting from *0x01* through to *0x09*, with each address corresponding to a particular functionality. For example, address *0x01* is the precompiled contract for the ECDSA recovery function (`ecrecover`), which is used to recover the public key associated with a digital signature. This function is critical for verifying user identities and transaction authenticity. The implementation details of this contract are encapsulated within the Ethereum execution client and can be accessed directly using a special address [8].

For instance, to verify a signature within a smart contract, one would use the following Solidity (see next section) code snippet:

This function leverages the `ecrecover` precompiled contract to perform signature verification, highlighting its practical application within Ethereum-based systems.

Precompiled contracts do not execute inside traditional smart contracts but are integral to the Ethereum client's architecture. They are specified in the Ethereum Yellow Paper and are part of the protocol's official specification, ensuring standardization across different client implementations. For detailed implementation references, see the Geth

```
function recoverSignature(bytes32 hash, uint8 v, bytes32 r, bytes32 s)
public view returns (address) {
    address signer = ecrecover(hash, v, r, s);
    require(signer != address(0), "Invalid signature");
    return signer;
}
```

Figure 2.2: Solidity function for signature recovery using `ecrecover`

client's codebase[5] or the Ethereum Yellow Paper [8].

The scope of precompiled contracts have expanded over various network upgrades. The Ethereum community continuously evaluates the addition of new precompiled contracts to address emerging cryptographic needs and network efficiency improvements. A comprehensive list of precompiled contracts as of the Cancun release is available at `https://www.evm.codes/precompiled`.

2.3 Smart Contracts

Smart contracts are autonomous programs that are written in code and deployed on the Ethereum blockchain, where they reside as immutable entities. Unlike traditional contracts, which require human intermediaries, smart contracts enforce their own terms through code execution. They are capable of receiving, holding, and sending digital assets according to the predetermined rules encoded within them. Smart contracts operate as independent Ethereum accounts with the ability to initiate transactions. These transactions can alter the state of the blockchain, allowing the contracts to perform a variety of functions from financial transactions to automated decision-making [6].

Smart contracts are primarily written in high-level languages such as Solidity [15] or Vyper [16], which are then compiled into EVM bytecode [17].

The smart contract lifecycle begins with development, where the contract is written in a high-level language. After compilation into bytecode, the contract is deployed to the Ethereum blockchain through a transaction. Once deployed, the contract has an address through which users can interact with it by sending transactions that trigger its functions [17].

Smart contracts live on the Ethereum blockchain as immutable code, meaning the deployed code cannot be altered. Interaction with a smart contract can change its state [8].

[5] `https://github.com/ethereum/go-ethereum/blob/master/core/vm/contracts.go`

2.3.1 Solidity

Solidity is the high-level programming language for crafting smart contracts on the Ethereum platform. With a syntax reminiscent of JavaScript, Solidity is a statically typed language that supports advanced features such as inheritance, libraries, and complex user-defined types [15].

Solidity supports a range of functionalities, including managing Ether transactions, emitting events, and facilitating contract interactions. This capability enables the development of diverse applications, from simple token contracts to complex decentralized autonomous organizations (DAOs)[17].

Solidity's widespread use is indicative of its integral role in the Ethereum ecosystem. The language's extensive feature set establishes Solidity as a key tool for smart contract development on Ethereum.

2.4 Decentralized Exchanges

As the adoption of cryptocurrencies continues to rise, a diverse array of trading platforms has emerged to meet investor demand. Conventional centralized cryptocurrency exchanges, managing a daily trade volume exceeding $40 billion as of June 2024 [18], operate similarly to traditional financial exchanges. These platforms maintain centralized control over all transactions and the custody of assets. They utilize a Continuous Limit Order Book (CLOB) architecture, where a central entity manages a ledger of all open buy and sell orders. This method ensures efficient, real-time matching of buyers and sellers, with trades executed sequentially as they are placed. The requirement for centralized exchanges to secure all assets and fiat currencies demands a high degree of trust from their users, given their role as custodians [1] .

In contrast, decentralized exchanges (DEXs) offer a paradigm shift by operating on a blockchain-based infrastructure that decentralizes control over transactions. Unlike their centralized counterparts, DEXs do not retain users' funds; instead, trades are executed directly between users' wallets via smart contracts. This setup not only enhances security but also increases trust, as it eliminates the risk of manipulation or mismanagement by a central authority. Smart contracts on platforms such as Ethereum ensure that these operations are autonomous, running exactly as programmed without any possibility of downtime, censorship, fraud, or third-party interference. Typically, users engage with these smart contracts directly through their cryptocurrency wallets, further reinforcing the decentralized nature of these exchanges [19] [20].

DEXs can employ various operational models, but the most notable are continuous-limit order books and Automated Market Makers (AMMs). Some DEXs maintain continuous-limit order books similar to traditional exchanges but manage them via smart contracts. Platforms like Etherdelta [21] and certain applications of the 0x [22] protocol allow traders to hold their assets on-chain, with the smart contract substituting for the traditional role of the exchange operator. Orders are maintained off-chain, and traders are responsible for matching by presenting new orders along with signed

counterorders to the smart contract. Other platforms, such as IDex [23] and Paradex [24], perform matching off-chain and merely use the blockchain for finalizing transactions [19].

However, the most innovative approach among DEXs is the use of AMMs. This model departs from traditional order books, opting instead for liquidity pools managed by smart contracts. These pools maintain balanced reserves of various tokens and facilitate peer-to-peer trading at rates determined by a predefined mathematical formula—the swap invariant. This system enables dynamic price discovery, as liquidity providers contribute equal values of two different tokens, receiving liquidity tokens in return. These tokens symbolize their stake in the pool, ensuring they are rewarded for providing liquidity. Traders interact directly with the liquidity pool, with prices automatically adjusted based on the changing ratios of tokens within the pool [1] [20].

AMMs enhance trading by utilizing a Constant Product Market Maker (CPMM) model, expressed mathematically as $x \times y = k$, where x and y represent the quantities of two different crypto tokens in the liquidity pool, and k is a constant. This formula maintains a constant product of token quantities, allowing the pool to dynamically adjust prices according to fluctuations in supply and demand [20].

Traders and liquidity providers interact with the liquidity pool via the AMM's smart contracts. Liquidity providers deposit an equivalent value of two tokens, receiving liquidity tokens in return, which represent their share of the pool. Traders engage with these liquidity pools directly, with prices adjusted automatically by the AMM according to the current token ratios [20].

A critical aspect of trading on AMMs is the management of slippage. Slippage - differences between expected and actual execution prices - occurs primarily due to fluctuations in liquidity. When a trade is executed, a trader may experience expected slippage, where the price worsens as trading volume increases. Additionally, unexpected slippage can occur when the actual execution price deviates from the anticipated price due to changes in the blockchain state between the creation and execution of a transaction, often exacerbated by frontrunning. To mitigate such risks, traders typically establish a slippage tolerance, setting a maximum acceptable deviation from the expected price, which helps in managing the uncertainty inherent in these transactions [25].

2.5 Maximal Extractable Value

MEV refers to the total value that can be extracted from a blockchain by manipulating the order and inclusion of transactions within a block. MEV arises from the unique ability of certain participants to exploit transaction order for financial gain, often at the expense of ordinary users [1].

MEV can manifest in various forms. Actors, often referred to as "searchers", exploit the public nature of transaction pools (mempools). They monitor these pools for opportunities such as arbitrage, liquidations, and other time-sensitive trades. By strategically placing transactions (front-running or back-running), these actors can influence the

transaction order to their financial advantage, exploiting the state before it is confirmed on the blockchain [1].

To understand the extent of MEV's impact, it is essential to explore the different strategies employed by these actors to extract value. The common strategies include sandwiching, arbitrage, and liquidation, each taking advantage of market vulnerabilities in unique ways [26].

Sandwiching: This strategy involves monitoring the mempool for large pending transactions that can affect asset prices. MEV actors then place their own orders both before (front-run) and after (back-run) the large transaction, benefiting from the price movements caused by the significant trade [26].

Arbitrage MEV: MEV actors exploit price differences between assets across different exchanges. They may replicate pending arbitrage transactions seen in the mempool, placing higher fee transactions to front-run the arbitrage, securing profits before the original transaction can execute [26].

Liquidation MEV: In DeFi lending platforms, loans that become undercollateralized may be liquidated. MEV actors look for such opportunities to repay the debt on behalf of the borrower and receive the collateral at a discount, profiting from the price difference between the debt amount and the collateral value [26].

The extraction of MEV not only leads to financial gains for certain participants but also introduces several negative externalities to the blockchain ecosystem [1]:

- **Network Congestion:** High-frequency trading strategies employed by MEV actors can lead to a surge in transaction volume, overwhelming the network. Additionally, failed transactions from competing MEV searchers also consume block space, further congesting the network.

- **Increased Transaction Costs:** MEV actors often bid higher gas prices to prioritize their transactions, which results in overall increased transaction costs for regular users. This is commonly seen in Priority Gas Auctions (PGAs), where bots compete against each other by bidding up transaction fees (gas) to capture pure revenue opportunities.

- **Consensus Instability:** Profitable MEV opportunities can tempt validators to deviate from honest behavior, potentially leading to consensus instability and reduced security. High-MEV regimes generally increase the risk of 'Time-bandit attacks', where validators might rewrite blockchain history to profit from MEV opportunties.

The DeFi ecosystem, particularly DEXs, has been notably affected by MEV. In DEXs, trades are executed by smart contracts on the blockchain, providing an ostensibly transparent and tamper-proof environment. However, the inherent latency in blockchain's consensus mechanisms allows for MEV through transaction ordering manipulation. This has led to prevalent front-running, where transactions are observed and preemptively exploited, undermining the fair and equitable operation of these platforms [1].

The quantification of MEV has been challenging due to its diverse manifestations across different blockchains and protocols. Tools like "MEV-Explore[6]" estimate significant MEV extraction on platforms like Ethereum, with over $700 million extracted up to the merge in September 2022. Post-merge, the Flashbots Transparency Dashboard[7] reports that Realised Extractable Value (REV) on Ethereum has reached 526,207 ETH, highlighting the ongoing significant impact of this phenomenon [27].

Efforts to mitigate MEV have included both technical solutions, such as redesigning blockchain architectures to reduce information asymmetry and economic incentives, and community-led initiatives aimed at creating fairer trading environments. Proposals like the introduction of Proposer Builder Separation (PBS) in Ethereum's transition to PoS represent significant steps towards minimizing the risks associated with MEV by decoupling the process of block proposal from block execution [27].

2.6 Proposer Builder Seperation

The concept of PBS has been developed within Ethereum's ecosystem to address centralization risks that have been exacerbated by the dynamics of MEV. This structural modification is aimed at enhancing network security and maintaining the integrity of block production within Ethereum's inherently decentralized architecture [28].

As Ethereum has evolved, the demand for block space has significantly outstripped supply, leading to notable increases in transaction fees. This economic shift has incentivized block producers to maximize fee collection per block. The ability to include high-value private transactions and employ sophisticated trading strategies has created disproportionate advantages for certain producers, particularly those with superior resources or exclusive access to transaction information. These advantages have the potential to dominate the blockchain's economic landscape [28].

This heterogeneity in capability and information among block producers introduces significant centralization pressures, which contradict the decentralized ethos of blockchain technology and pose substantial risks to the network's security and fairness. Historical insights suggest that without intervention, block production could become a specialized market dominated by a few, highly capable entities. This could lead to a network that is governed by the interests of major financial institutions, resulting in a staking pool market dominated by entities with significant capital or the highest return on investment, thereby undermining the broader community's participatory role [28] [29].

To counteract these centralization forces, PBS introduces a separation of roles in the block production process:

- **Builders:** These entities specialize in assembling blocks, optimizing the order and selection of transactions within a block to maximize its value, primarily through

[6] `https://explore.flashbots.net`
[7] `https://transparency.flashbots.net`

MEV. Builders compete to have their blocks selected by validators, relying on sophisticated algorithms and private transaction flows to construct potentially lucrative blocks [3].

- **Proposers (Validators):** In the PBS model, validators retain their fundamental role in the consensus mechanism but are no longer responsible for block construction. Instead, they select from the blocks offered by various builders based on the profitability of the bids attached to these blocks. This separation allows validators to remain neutral and reduces their capability to extract excessive MEV, promoting a more equitable and decentralized ecosystem [3].

2.6.1 MEV-Boost

MEV-Boost represents a practical implementation of the PBS concept, serving as an intermediary layer to facilitate interactions between builders and validators. Builders submit their blocks to MEV-Boost, which then presents these blocks to validators. The operational dynamics of MEV-Boost are pivotal in decentralizing the block construction process and ensuring equitable participation across the Ethereum network. This system fosters a specialized market for block construction, where builders compete to assemble blocks that include valid transactions from private order flows, the public mempool, and their own resources [30].

Figure 2.3 provides an overview of the MEV-Boost integration with the Ethereum network. This diagram illustrates the interaction between various components, which are further detailed below.

- **Relays:** These act as neutral intermediaries, as shown in the bottom-center of Figure 2.3, collecting and organizing bids from builders. They connect to both internal and external builders and may enforce varied censorship policies to comply with regulations like OFAC standards [30].

- **Auction Mechanism:** Builders engage in a first-price sealed-bid auction, attaching bids to their blocks. This mechanism incentivizes validators to select blocks that offer the highest payments, fostering a competitive market that enhances the economic efficiency of block production and reduces the influence of dominant builders. The interaction between builders and relays in this auction process is depicted in the bottom-right of Figure 2.3 [30].

- **Searchers:** Participants using advanced algorithms or bots to identify lucrative MEV opportunities, who submit transaction bundles (is introduced in the next section) to builders via private channels, are represented in the top-right section of Figure 2.3 [30].

The operational dynamics of MEV-Boost, as illustrated in Figure 2.3, are pivotal in decentralizing the block construction process and ensuring equitable participation across the Ethereum network. The system creates a specialized market for block construction,

where builders compete to assemble blocks that include valid transactions from private order flows, the public mempool, and their own resources. Builders participate in a concise, 12-second ascending bid auction to determine the fees they are willing to pay validators for block inclusion. Relays act as trusted intermediaries in these auctions, safeguarding the process by ensuring that only the highest bidding blocks reach validators, thereby promoting fairness and transparency [30].

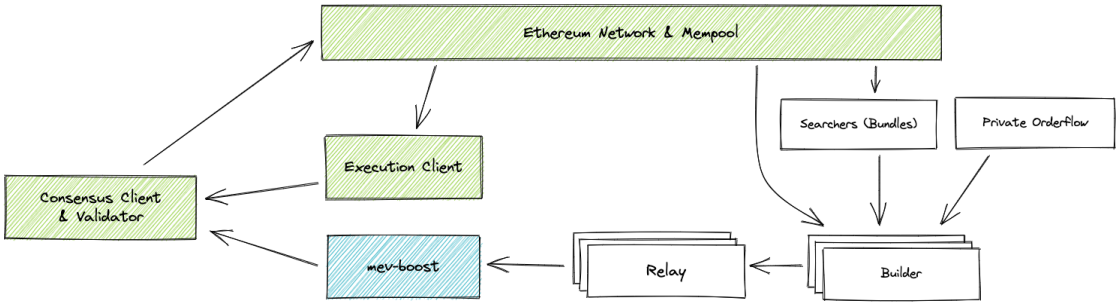

Figure 2.3: MEV-Boost Integration Overview [31]

Validators have the choice to propose either their own constructed blocks or the highest bidding block from the auction, optimizing cost efficiencies related to block construction and order flow security. This mechanism not only saves costs but also supports a merit-based block selection process [30].

Furthermore, MEV-Boost significantly enhances Ethereum's decentralization by enabling any participant to become a builder, thus promoting greater competition and diversity among block constructors. This inclusivity broadens participation in Ethereum's economic activities beyond traditional validators, leading to a fairer, market-driven approach to block selection. By democratizing blockchain operations, MEV-Boost effectively shifts the focus from merely proposing blocks to fostering a more competitive and collaborative environment for block construction and validation, thereby mitigating centralization risks at the validator level [30].

2.7 Flashbot Bundles

Flashbots, a research organization established to address the negative impacts of MEV, introduced transaction bundles in 2021. These bundles ensure transactions are executed in a specified order and may include the searcher's transactions alongside others from the mempool, aimed particularly at targeting specific blocks without revealing strategies to potential adversaries [32].

Bundles are submitted to the Flashbots Builder API via the `eth_sendBundle` method, which demands a precise structure including lists of transactions, their target block numbers, and the validity period defined by minimum and maximum timestamps. This method also requires a signature, included in the '*X-Flashbots-Signature*' header, to authenticate and ensure the integrity of the submission [31].

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "eth_sendBundle",
  "params": [
    {
      "txs": ["0x123abc...", "0x456def..."],
      "blockNumber": "0xb63dcd",
      "minTimestamp": 0,
      "maxTimestamp": 1615920932
    }
  ]
}
```

Figure 2.4: Example JSON payload for eth_sendBundle method.

Executing bundles within their targeted block is fraught with challenges such as unexpected transaction failures, competition from higher bidder's bundles, precise timing requirements that miss narrow execution windows, all of which can thwart a bundle's inclusion [31].

3 Privacy Preserving Smart Contracts

Blockchain technology is renowned for its transparency, which ensures the integrity of transactions but also introduces significant privacy concerns. Traditional blockchain systems, such as Bitcoin, publicly disclose every transaction and the state of each account [5].

Privacy-focused advancements began with Zerocoin [33], which evolved into Zerocash by Sasson et al [34]. Zerocash enhanced privacy by allowing transactions to obscure amounts and paths using non-interactive zero-knowledge proofs (NIZKs). These proofs have become essential in developing subsequent privacy-preserving features in blockchain applications [35].

However, adapting these privacy mechanisms to smart contracts introduces complex challenges. Unlike regular transactions, smart contracts involve multifaceted operations and state transitions that are difficult to obscure without compromising functionality. While Zerocash addressed transaction privacy, applying similar principles to smart contracts—encompassing both code and state that must remain publicly verifiable—requires a nuanced approach [35].

The necessity for privacy in smart contracts has driven the development of specific solutions across various sectors, including healthcare [36], data marketplaces [37], and energy [38], where privacy is paramount. These solutions utilize cryptographic techniques such as commitments, encryption, and hashing to protect sensitive data. However, they are often custom-designed for specific applications and not easily transferable to other contexts [35].

To address these limitations, there is a growing trend towards creating adaptable and universally applicable privacy-preserving smart contract (PPSC) frameworks. These frameworks aim to equip developers with tools that facilitate the integration of privacy features into smart contracts, minimizing the need for specialized knowledge. This shift towards more generalized frameworks represents a significant advancement in the field, seeking to seamlessly incorporate privacy into smart contract design [35].

Exploring PPSC schemes further reveals significant variations in design and the challenges faced, depending on the cryptographic techniques employed. The subsequent sections will delve into these techniques, examining their integration into smart contracts and the trade-offs they involve regarding efficiency, security, and privacy.

3.1 PPSC Schemes

In the model defined by KACHINA for PPSCs, the state of a smart contract is divided into a public state σ and a private state ρ_u for each user u. The public state and a cryptographic reference to each user's private state—either as a hash or ciphertext—are stored on the blockchain. This setup ensures the integrity and verifiability of the state while keeping the details of the private state confidential and managed off-chain by individual users [39].

State transitions in PPSCs involve either single-party or multi-party operations. In a single-party transition, a user u or a TEE executor applies a transition function Δ, changing the state from (σ, ρ_u) to (σ', ρ'_u) and generating a cryptographic proof π. Block producers verify this proof and then update the blockchain to reflect the new public and private states. These transitions are suitable for scenarios where the actions of one user do not affect the private states of others. Conversely, multi-party transitions involve simultaneous updates to multiple users' private states, requiring complex coordination and posing significant implementation challenges [39].

To concretize the discussion of PPSCs, consider the implementation of a silent auction. In this scenario:

- The public contract state σ includes the addresses of all participants.
- Each participant's private state ρ_u confidentially holds their bid.

During the auction, bids are processed individually through a single-party transition Δ_{bid}, which takes the participant's address u and bid *bid* as inputs. This transition updates both the public contract state σ and the participant's private state ρ_u along with a proof π, subsequently broadcast to block producers for validation [39]:

Δ_{bid} : Takes the participant's address u and bid *bid* as inputs,
updates the public contract state σ and the participant's private state ρ_u
with a proof π.

At the auction's conclusion, a multi-party transition Δ_{reveal} publicly reveals the highest bid by updating the public state σ [39]:

Δ_{reveal} : Reveals the highest bidder and their bid by updating
the public state σ.

PPSCs can be implemented through two principal approaches, each leveraging distinct technologies to enhance privacy and security. These approaches are broadly categorized as crypto-based and TEE-based PPSC schemes. Crypto-based PPSC schemes employ cryptographic tools such as non-interactive zero-knowledge proofs (NIZK), secure multi-party computation (MPC), and homomorphic encryption (HE). These technologies

enable the secure and confidential execution of smart contracts by protecting transaction details from being exposed on the blockchain, ensuring that sensitive data is handled discreetly among involved parties [35]. In contrast, TEE-based PPSC schemes utilize TEEs to safeguard transaction execution. Systems like Intel Software Guard Extensions (SGX) offer secure processing environments that isolate contract execution from the main operating system. This isolation, supplemented with remote attestation, verifies the security and integrity of the execution environment to all network participants [35].

These two approaches address different aspects of security and privacy challenges within blockchain environments. The following sections will delve into the operational mechanisms, benefits, and inherent challenges associated with each type of PPSC scheme.

3.1.1 Commitment Schemes

Commitment schemes can be utilized for PPSCs, enabling parties to commit to a value without revealing it until a specified time. This mechanism is crucial in applications where data integrity and confidentiality must be temporarily maintained [40].

A commitment scheme typically operates in two phases [40]:

- **Commit Phase:** The sender selects a message m and a random value r, computes the commitment $\text{Comm}(m, r)$, and sends it to the receiver. This ensures the sender's commitment to the value m without revealing it.
- **Reveal Phase:** The sender reveals m and r to the receiver, who verifies the commitment using the provided information to confirm it matches the commitment made during the commit phase.

In smart contracts, commitment schemes are particularly useful for scenarios like auction systems, where bids need to be kept confidential until the end of the auction to ensure fair bidding processes. Advanced commitment schemes like Pedersen and ElGamal provide additional properties that enhance their utility in such applications [40].

The Pedersen commitment scheme is favored in environments requiring additive homomorphism. It allows for the aggregation of committed values (e.g., summing bids) without revealing individual values. This is especially useful in auctions where only the total or highest bid might need to be disclosed at the end, rather than each individual bid [41].

ElGamal commitment offers multiplicative homomorphism, beneficial in scenarios where products of committed values are needed. This feature can be used in financial computations on the blockchain, such as calculating compounded interest rates on committed investments without disclosing individual rates [42].

For applications with lower confidentiality demands, simple hash commitments using one-way hash functions may be adequate. These commitments are straightforward to compute and verify, providing a basic level of privacy and data integrity. They are suitable for environments where security requirements are minimal, and computational overhead needs to be reduced.

3.1.2 Secure Multi-Party Computation

Secure MPC is a cryptographic technique that allows multiple parties to compute a function over their private inputs, ensuring that no individual input is revealed to other participants. MPC empowers each participant to contribute to a collective computation without exposing their private data. This is achieved by dividing secret data into shares that are distributed among the participants. For instance, to securely compute the sum $A + B$ without revealing the individual values, A and B can be split into parts $[A]_1, [A]_2, [A]_3$ and $[B]_1, [B]_2, [B]_3$ respectively. Each participant computes a partial sum $[C]_x = [A]_x + [B]_x$, and the final result C is reconstructed only when the shares are combined, preserving the privacy of A and B [35].

The application of MPC in smart contracts facilitates functions such as auctions, where bids are computed to find the highest without revealing individual bids, or financial contracts that require the aggregation of private financial data. Notable efforts to incorporate MPC in PPSCs include projects like Keep Network [43] (now part of Threshold Network [44]) and Enigma (now Secret Network [45]).

MPC is theoretically capable of performing any computation expressed through additions and multiplications, making it an exciting approach for privacy-preserving computations. However, the protocol inherently requires extensive communication among participants, leading to high latency and reduced scalability. Furthermore, the integrity of the computation critically depends on the honesty of all participating nodes. A single dishonest node can corrupt the entire result, rendering the computational effort of other participants useless and potentially compromising the outcome of the smart contract [35].

An advancement in the field of MPC is the development of Public Auditable MPC (PA-MPC), which allows multiple parties to jointly evaluate a function and provide proof to convince verifiers of the correctness of the computed result without requiring any knowledge of the participants' private inputs. This feature adds a layer of transparency and security, essential for applications with significant economic or political implications [46].

By design, the computational work in an MPC setup can be separated into an offline phase, where most cryptographic operations are pre-computed, and a minimally intensive online phase, enhancing performance. Despite these optimizations, sMPC protocols remain slow and expensive due to the communication costs between nodes. All nodes must execute the program correctly using the secret shared data they were given. If a node replaces their output with a random value, it can disrupt the entire computation process, wasting the efforts of all participants involved [35].

3.1.3 Homomorphic Encryption

HE is an advanced cryptographic method that enables the execution of calculations on encrypted data, producing an encrypted result that, when decrypted, corresponds to the result of operations performed on the plaintext. This capability is crucial for privacy-

preserving smart contracts, which operate on sensitive data without compromising its confidentiality [35].

HE is categorized into two types based on the operations it supports [35]:

- **Partial Homomorphic Encryption (PHE):** PHE allows either addition or multiplication on ciphertexts. For example, the Paillier scheme, which is additively homomorphic, ensures that for any encrypted messages m_1 and m_2, the operation $\text{Enc}_k(m_1) + \text{Enc}_k(m_2)$ will yield $\text{Enc}_k(m_1 + m_2)$.
- **Fully Homomorphic Encryption (FHE):** FHE, introduced by Craig Gentry in 2009, supports both addition and multiplication, allowing for arbitrary computations on encrypted data. Although FHE is theoretically potent, its practical use is limited due to substantial computational overhead.

HE can be particularly useful in smart contracts for operations such as confidential transactions where user account balances are encrypted and updated through homomorphic operations without revealing the actual values. For instance, Zether uses HE to support confidential payments, while smartFHE extends these capabilities to allow arbitrary computations on encrypted inputs, enabling both public and private smart contracts [47].

While HE offers significant advantages for privacy-preserving computations, it also presents substantial challenges. First, the computational intensity of HE, especially FHE, necessitates extensive computational resources. This high demand can markedly slow down the processing speed of smart contracts and escalate transaction costs. Additionally, as the scope of smart contract applications widens and the number of participants increases, managing cryptographic keys becomes more complex. This complexity can complicate both the deployment and ongoing maintenance of smart contracts, potentially limiting wider adoption. Furthermore, despite theoretically supporting arbitrary computations, the practical application of FHE is frequently curtailed by its immense computational demands and the intricacies of the cryptographic protocols involved [35].

J.P. Morgan's Quorum platform, an enterprise-focused version of Ethereum, incorporates HE to enhance its privacy features. Building on this, Zether, a confidential payment mechanism compatible with Ethereum, extends the privacy capabilities by using encrypted account balances and cryptographic proofs to ensure transaction confidentiality. Zether supports an account-based model, similar to Ethereum, and maintains encrypted account balances, leveraging its homomorphic properties to perform confidential and anonymous transactions [48].

3.1.4 Zero-Knowledge Proofs

Zero-knowledge proofs (ZKPs) offer a sophisticated method for proving the validity of a statement without revealing any information about the statement itself. Originating from a seminal study in 1985, ZKPs enable a prover to demonstrate knowledge of a fact without exposing the actual information to the verifier. This unique characteristic

ensures that the verifier learns nothing beyond the fact that the statement is true, crucial for maintaining confidentiality in sensitive transactions [49].

Interactive ZKP

Interactive zero-knowledge proofs involve a series of exchanges between the prover and the verifier, structured around three key components: witness, challenge, and response. The sequence initiates with the prover presenting a *witness*—confidential information that serves as the foundation of the proof. During the *challenge* phase, the verifier poses questions related to the witness, to which the prover responds in the *response* phase, thereby confirming their knowledge of the witness without disclosing it [35].

Sigma Protocol: A practical implementation of an interactive ZKP in smart contracts is the Sigma Protocol. This protocol follows the *honest verifier zero-knowledge* (HVZK) principle and includes three phases [40]:

1. *Commitment Phase*: The prover commits to a secret value, setting the stage for the proof.

2. *Challenge Phase*: The verifier, unaware of the secret, issues a challenge to the prover.

3. *Response Phase*: The prover computes and returns a response based on the secret value, which the verifier evaluates using the initial commitment and challenge to determine the proof's validity.

Non-interactive ZKP (NIZK)

In contrast to interactive ZKPs, non-interactive zero-knowledge proofs require no ongoing communication between the prover and verifier. The prover generates a standalone proof that can be independently verified by anyone, leveraging cryptographic methods that depend on complex mathematical problems to ensure both the validity of the proof and the confidentiality of the underlying information. NIZKs are particularly useful in decentralized systems like blockchains, where proofs must be publicly verifiable [40].

Types of NIZKs and Their Characteristics [40]:

- **zk-SNARKs (Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge):** These are favored for their efficiency and brief communication needs, making them ideal for blockchain applications. They involve a setup phase for key generation, a proving phase where the prover creates a concise proof, and a verification phase where the proof's validity is checked. The main drawback is the need for a trusted setup, which poses security risks if compromised.

- **Bulletproofs:** These do not require a trusted setup, enhancing their security. However, they involve more complex computations and produce larger proofs, which could impact performance.

- **zk-STARKs (Zero-Knowledge Scalable Transparent ARgument of Knowledge):** Known for their scalability and transparency, zk-STARKs support high-throughput applications and provide post-quantum security. They do not require a trusted setup, thus alleviating related security concerns, although they generate larger proofs than zk-SNARKs, increasing verification time.

Practical Implementations of NIZK Protocols for PPSCs: Hawk and zkHawk are two examples of how NIZKs can be integrated into PPSCs to enhance transactional confidentiality and the integrity of contract execution within blockchain environments.

Hawk is a decentralized framework that enhances privacy by enabling smart contracts to execute without revealing transactional data on the blockchain. It utilizes zero-knowledge proofs to validate the correctness of transactions under strict privacy constraints. Hawk's architecture separates contract logic into public and private components: the public part operates on the blockchain, handling non-private data and verifying transaction validity, while the private part, managed off-chain, deals with sensitive computations and data. A specialized manager is employed to oversee these off-chain operations, tasked with proving the privacy and correctness of executions through ZKPs without exposing the data involved [50].

Building on Hawk's foundation, **zkHawk** addresses its precursor's limitations by removing the dependency on a singular trusted manager. Instead, it introduces a MPC protocol that distributes the execution of private contracts across multiple parties. This decentralization significantly reduces risks related to single points of failure and bolsters the system's defense against collusion and privacy breaches. zkHawk employs ZKPs to verify that all parties involved in the MPC comply with the contract's rules without revealing their individual inputs, thereby ensuring the decentralized integrity and confidentiality of the transactions [51].

3.1.5 Trusted Execution Environments

While cryptographic schemes such as NIZKs, MPC, and HE enable PPSCs, they often come with significant performance overhead and are limited to relatively simple computations. TEEs offer a more performant and general-purpose alternative for implementing PPSC [52].

TEEs provide a secure and isolated operating environment separate from the main operating system, ensuring protection of critical processes and sensitive data from unauthorized access and tampering. This fully isolated environment prevents other software applications, the operating system, and even the host owner from interfering with or learning the state of applications running within the TEE [52].

TEE-based PPSC Schemes

TEE-based PPSC schemes introduce hardware manufacturers as trusted parties, utilizing trusted execution environments such as Intel SGX. A key feature of TEEs is remote attestation, which provides proofs to convince other parties of the trustworthiness and integrity of the executing environment, eliminating the need for NIZK proofs [52].

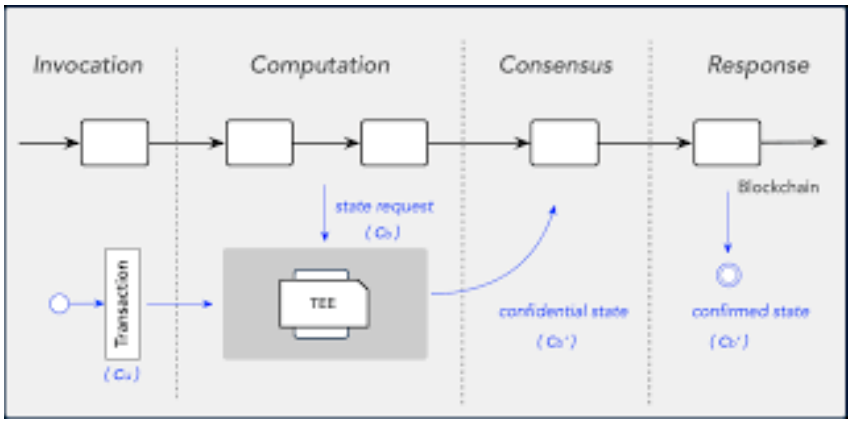

Figure 3.1: TEE-based PPSC Scheme Workflow [35]

The process of executing confidential smart contracts within a TEE involves four main steps [35]:

1. **Invocation:** The contract is activated by loading its state and operational code to distributed nodes. A user initiates a state change by sending an encrypted transaction (T_x with ciphertext cu) to the TEE.
2. **Computation:** The TEE decrypts the received data (cu) and the current encrypted contract state (cb) fetched from the blockchain. It then executes the contract logic confidentially and produces an output (m_b). This output is encrypted with the user's public key to obtain c'_b, which is then sent to the blockchain network.
3. **Consensus:** The consensus mechanism verifies the encrypted state (c'_b) from the transaction during the mining of new blocks. When a majority of consensus nodes receive the same block and re-execute included transactions, the state c'_b with its carrier T_x is deemed confirmed and becomes immutable.
4. **Response:** The blockchain returns the final state c'_b and its corresponding transaction T_x to the user. Only the user who owns the private key can decrypt c'_b to obtain the final plaintext result.

Ekiden: A TEE-Blockchain Hybrid System

Ekiden represents a sophisticated system designed to address critical gaps in blockchain technology by combining it with TEEs. It uniquely separates computation from con-

sensus, enabling efficient TEE-based PPSC and high scalability. At the core of Ekiden's architecture is the separation of computation and consensus roles [52]:

- **Compute Nodes:** These perform smart contract computation over private data off-chain in TEEs, then attest to their correct execution on-chain. By operating in TEEs, Ekiden imposes minimal performance overhead.
- **Consensus Nodes:** These maintain the blockchain ledger and need not use trusted hardware. Ekiden is agnostic to consensus-layer mechanics, only requiring a blockchain capable of validating remote attestations from compute nodes.

This separation allows Ekiden to scale consensus and compute nodes independently, markedly improving system scalability and efficiency over traditional on-chain processes. Benchmark studies with Tendermint as the consensus layer show that Ekiden can process transactions up to 600 times faster, with 400 times less latency, at costs approximately 1000 times lower than those of Ethereum's mainnet [52].

Ekiden is implemented with Intel SGX, but it's designed to work with any comparable TEE technology that offers attestation capabilities. The system comprises three primary entities [52]:

1. **Clients:** End users who interact with smart contracts using secret inputs. They are responsible for initiating contract creation or execution by submitting these inputs to the compute nodes.
2. **Compute Nodes:** Execute contracts within TEEs and generate cryptographic attestations validating the correctness and integrity of state transitions.
3. **Consensus Nodes:** Maintain blockchain integrity by validating state updates based on attestations from compute nodes.

A key feature of Ekiden is its protocol for atomic delivery of two messages generated by TEE execution: m1 (output to the caller) and m2 (state update to the blockchain). This ensures that both messages are delivered, maintaining consistency between client outputs and blockchain state [52].

Key Management in TEE-Blockchain Systems

A critical challenge in TEE-blockchain systems is maintaining confidentiality while persisting state on the blockchain. While encryption solves this issue, it introduces the problem of securely managing encryption keys. The common approach of replicating keys across multiple TEEs creates a tension between exposure risk and availability: higher replication improves resilience to state loss but increases the attack surface for key exfiltration through side-channel attacks [52].

Given the lack of definitive full-system side-channel mitigation, key management protocols must be designed against a stronger adversarial model that assumes an attacker can compromise a small fraction of TEEs. The Ekiden system proposes an advanced protocol to address these challenges [52]:

- **Key Management Committees (KMCs):** A set of nodes form a committee to manage keys, distributing trust and reducing single points of failure.
- **Two-tier Key System:** Long-term keys are generated using Distributed Key Generation (DKG) and secret-shared among KMC members. Short-term, epoch-based keys are derived from these long-term secrets.
- **Sybil-resistant Identities:** Participating hosts must have identities resistant to Sybil attacks, often implemented through security deposits.
- **Breach Isolation:** A privacy budget is enforced for each compute node to limit potential damage from compromised nodes.

In operation, when initializing a contract, the KMC generates a long-term key via DKG. For each epoch, compute nodes request short-term keys from KMC members, who compute and send key shares. The compute node constructs the full short-term key after collecting a threshold number of shares [52].

To prevent widespread key compromise, the system implements breach isolation through a query counter for each compute node, reset each epoch. Key-manager nodes only fulfill queries if this counter is below a set threshold, effectively quarantining potential confidentiality breaches [52].

It's worth noting that Ekiden's approach is one of several in the field of TEE-blockchain systems. Other TEE-based blockchain networks have developed alternative key management strategies, which will be analyzed in subsequent sections.

Challenges and Limitations of TEE-Based PPSC Schemes

While TEE-blockchain hybrid systems offer powerful guarantees, their integration introduces significant challenges. The following presents challenge, identified in the Ekiden TEE PPSC system paper by Raymond Cheng et al. [52]:

Timer Failures: TEEs generally lack trusted time sources, complicating blockchain integration. The available trusted relative timer only provides a "no-earlier-than" time notion, as malicious hosts can delay communication between enclaves and the timer. This makes maintaining an up-to-date blockchain view challenging, necessitating minimal reliance on TEE timers in hybrid protocols.

Side-Channel Attacks: Despite strong isolation guarantees, TEEs are vulnerable to data leakage through side-channel attacks. Current defenses are largely application-specific, and developing generalized protections remains an open challenge in TEE security research.

Availability Failures: TEEs cannot ensure their own availability, as hosts can terminate enclaves arbitrarily or lose them during power cycles. TEE-blockchain systems must tolerate such failures while ensuring that crashed TEEs only delay execution rather than compromising security or consistency. This requires careful system design and robust failure handling mechanisms.

Integrity Attacks: In naive TEE-blockchain designs, blockchain integrity attacks can threaten TEE-protected content confidentiality. An attacker could potentially circumvent TEE-enforced privacy budgets by providing a forged blockchain, allowing arbitrary queries through execution rewinding. This vulnerability highlights the complex interplay between blockchain integrity and TEE confidentiality [53]. However, it's important to note that Ekiden has specifically addressed this issue in its design, implementing mechanisms to prevent such attacks and ensure the integrity of the system even in the presence of potentially forged blockchains [52].

3.2 Intel Software Guard Extensions (SGX)

Intel SGX is a set of security-related instruction codes integrated into modern Intel CPUs. SGX provides a TEE called an enclave, which allows applications to protect sensitive data and code from unauthorized access or modification, even in the presence of privileged malware or a compromised operating system [54].

3.2.1 Background

Modern software applications frequently process sensitive user data such as passwords, encryption keys, and health records. In most cases, only authorized recipients should be allowed to access this private information. While the operating system is responsible for enforcing security policies to prevent unintentional data leakage, a significant vulnerability persists - a malicious party with administrative privileges can access all system resources and extract secrets directly from memory. To defend against such attacks, Intel designed SGX to offer a higher level of protection [55].

3.2.2 Enclave

An enclave is a protected area in the application's address space that provides confidentiality and integrity even in the presence of privileged malware. Intel SGX uses enclaves to create a trusted execution environment within the processor [54].

Properties and Functionality

Enclaves have several key properties [54]:

- **Isolated Execution**: Enclave code and data are stored in the Processor Reserved Memory (PRM), a subset of DRAM that cannot be directly accessed by other software, including system software and System Management Mode code.
- **Memory Encryption**: The enclave memory is encrypted using industry-standard encryption algorithms with replay protection. The memory encryption key changes every power cycle and is stored within the CPU, inaccessible to software.
- **Access Control**: Enclave code can only be entered through specific entry points, and enclave data can only be accessed by code running inside the enclave.
- **Measurement**: SGX maintains a cryptographic hash (MRENCLAVE) of the enclave's initial state, including its code and data, which is used for attestation purposes.

The Enclave Page Cache (EPC), a subset of PRM, stores the contents of enclaves and associated data structures. The EPC is managed by the system software but protected by the CPU. Each 4KB EPC page is assigned to a specific enclave, and its allocation is tracked in the Enclave Page Cache Map (EPCM) [54].

Enclaves use a designated area in their virtual address space called the Enclave Linear Address Range (ELRANGE) to map their code and sensitive data stored in EPC pages. Memory accesses inside ELRANGE are guaranteed to obey the virtual memory abstraction, while those outside are not protected [54].

Enclave Lifecycle

The lifecycle of an enclave involves several stages, each initiated by specific CPU instructions [54]:

1. **Creation (ECREATE)**: The system software uses the ECREATE instruction to initialize a new enclave. This creates the SGX Enclave Control Structure (SECS) for the enclave.
2. **Loading (EADD)**: The EADD instruction is used to load initial code and data into the enclave. This includes both Thread Control Structure (TCS) pages and regular pages.
3. **Measurement (EEXTEND)**: The EEXTEND instruction updates the enclave's measurement (MRENCLAVE) used in the software attestation process.
4. **Initialization (EINIT)**: The EINIT instruction finalizes the enclave build process. After EINIT, the MRENCLAVE measurement is complete, and the enclave is ready to execute user code.
5. **Execution**: Once initialized, enclave code can be executed through specific entry points defined in the Thread Control Structure (TCS).

6. **Termination**: When the enclave is no longer needed, it can be terminated, and its resources are freed.

Throughout its lifecycle, an enclave interacts with the untrusted application through well-defined interfaces. Calls into the enclave (ECALLs) and calls out of the enclave (OCALLs) are defined using the Enclave Definition Language (EDL) and are mediated by proxy functions generated by the Intel SGX SDK [54].

Limitations

While SGX enclaves offer robust security features, they also come with certain limitations. One significant constraint is memory size. Initially, SGX supported enclaves up to only 128MB in size. Although later versions have increased this limit, it still restricts the amount of code and data that can be protected within an enclave, requiring careful memory management and potential partitioning of larger applications [56].

Hardware dependency is another limitation. SGX is specific to Intel processors, which limits its availability and can complicate deployment in heterogeneous computing environments. This processor-specific nature of SGX means that applications designed to use enclaves may not be portable to non-Intel platforms without significant modifications [56].

Development complexity is a considerable challenge when working with SGX. Creating SGX-enabled applications often requires substantial redesign of existing software. Developers must carefully partition their applications into trusted and untrusted components, manage the interaction between these components, and handle the limitations of the enclave environment. This process can be time-consuming and may require specialized expertise in secure programming practices [56].

Despite SGX's strong protections against direct attacks, side-channel vulnerabilities remain a concern. While enclaves protect against direct memory access, they may still be susceptible to certain side-channel attacks, such as cache timing attacks or page fault analysis. These vulnerabilities can potentially leak sensitive information from within the enclave, requiring additional mitigation strategies and careful design considerations to ensure the overall security of the system [55].

3.2.3 Attestation

Attestation in Intel SGX proves a particular piece of software is running within a genuine SGX enclave on a trusted platform. It supports both local and remote verification.

Local Attestation

Local attestation allows enclaves on the same platform to verify each other's identity and authenticity. This is necessary for secure inter-enclave communication on a single machine. For instance, an application might split its operations across multiple enclaves

for better isolation, and these enclaves would need to authenticate each other before exchanging sensitive data [54].

Remote Attestation

Remote attestation is essential for establishing trust between an enclave and an external party. This process is crucial for scenarios where a remote service needs to verify that it's interacting with a genuine SGX enclave before provisioning secrets or sensitive data [54].

Two key types are particularly important for remote attestation [54]:

1. Attestation Key (EPID Key): This asymmetric key is used to sign quotes for remote attestation. It's based on Intel's Enhanced Privacy ID (EPID) scheme. The EPID key is not directly embedded in the CPU during manufacturing. Instead, each CPU receives a unique Provisioning Key, which it uses to participate in a complex provisioning protocol with Intel. Through this protocol, the CPU proves its authenticity and receives its EPID private key.

2. Provisioning Key: This key is derived from the Root Provisioning Key, which is embedded in the CPU during manufacturing. The derivation process involves both hardware-specific and software-specific components, ensuring that each CPU's Provisioning Key is unique. This key is crucial in the initial communication with Intel to obtain the EPID key, serving as proof of the CPU's authenticity.

The remote attestation process typically involves the following steps [54]:

1. The remote party issues a challenge to the enclave, usually including a nonce for freshness.

2. The enclave generates a report structure containing its identity (MRENCLAVE), attributes, and user-defined data.

3. This report is passed to a special Quoting Enclave (QE), which verifies the report and converts it into a quote.

4. The QE signs the quote using the platform's EPID key.

5. The signed quote is sent to the remote party.

6. The remote party forwards this quote to Intel's Attestation Service (IAS) for verification.

7. If the IAS confirms the quote's validity, the remote party can trust that it's communicating with a genuine SGX enclave.

This process helps the remote party trust the execution and integrity of the enclave in several ways [56]:

Firstly, the quote contains a measurement (MRENCLAVE) of the enclave's initial state, including its code and data. This allows the remote party to verify that the enclave is running the expected code, unmodified.

Secondly, the use of the EPID key for signing ensures that the quote comes from a genuine Intel SGX-enabled processor, without identifying the specific CPU. This provides assurance that the code is running within the hardware-protected environment that SGX promises.

Thirdly, the inclusion of the remote party's challenge (usually a nonce) in the quote proves the freshness of the attestation, preventing replay attacks.

Lastly, the verification by Intel's Attestation Service provides an additional layer of trust. It confirms that the signing key is valid and belongs to a non-revoked SGX processor, and that the processor is up-to-date with security patches.

All these factors combined allow the remote party to establish trust in the integrity and confidentiality of the enclave's execution, even if the rest of the system (OS, hypervisor, etc.) is potentially compromised [55].

Trust Assumptions and Security Considerations

The remote attestation process places significant trust in Intel as a company. The entire attestation process depends on Intel's key provisioning service and attestation service being secure and honest. There's an assumption that Intel correctly manages the EPID group keys and accurately verifies quotes without mistakenly validating compromised or simulated environments [55].

This trust extends to Intel's Attestation Service (IAS), a critical component in the remote attestation process. The security model assumes that the IAS correctly verifies the authenticity of quotes, maintains the integrity of its signing key, and cannot be compelled to produce false attestations. Any compromise of the IAS could potentially undermine the entire remote attestation system [55].

Additionally, the security of the EPID cryptographic scheme itself is crucial; any weaknesses in this scheme could undermine the privacy or integrity guarantees of the attestation process [55].

3.2.4 Sealing

Sealing is a crucial mechanism in Intel SGX that allows enclaves to securely store sensitive data outside the enclave for later use. This is particularly important when an enclave needs to preserve its state across events such as application closure, platform shutdown, or enclave destruction [54].

The sealing process involves encrypting the data using a key that is unique to the enclave and the platform. This key is derived from the Root Sealing Key, which is only known to the enclave. SGX provides two sealing policies [54]:

1. **Seal to Current Enclave**: This binds the sealed data to the specific enclave version (MRENCLAVE). Only the exact same enclave can unseal the data.

2. **Seal to the Enclave Author**: This binds the sealed data to the enclave author's identity (MRSIGNER) and the Product ID. It allows data sharing between different versions of enclaves from the same author.

The sealing process involves encrypting data with a unique key derived from the enclave and platform, then storing this encrypted data outside the enclave. Unsealing reverses this process. This mechanism allows enclaves to securely preserve sensitive data or state across enclave and system restarts, enabling the maintenance of long-term secrets [54].

3.2.5 Security Considerations of SGX

While Intel SGX provides strong security guarantees, it's important to note that it has not been immune to vulnerabilities. Several security incidents have demonstrated that SGX's protections can be bypassed under certain circumstances, raising concerns about the overall security of SGX-based systems.

A comprehensive overview of SGX vulnerabilities can be found at `https://sgx.fail`. This resource documents various attacks that have been discovered over the years, highlighting the ongoing challenges in maintaining the security of SGX enclaves.

One notable incident directly affected Secret Network, a blockchain platform that relies heavily on SGX for its privacy features. In August 2022, Secret Network was found to be vulnerable to the xAPIC[1] and MMIO[2] vulnerabilities, which were publicly disclosed on August 9, 2022. These vulnerabilities could potentially be exploited to extract the consensus seed, a master decryption key for private transactions on the Secret Network. This posed a significant risk, as exposure of the consensus seed would enable the complete retroactive disclosure of all Secret-4 private transactions since the chain's inception. In response, Secret Network implemented mitigations, including a Registration Freeze on October 5, 2022, to prevent new potentially vulnerable nodes from joining the network.

Furthermore, research presented in the paper "SGXonerated: Finding Privacy Flaws in TEE-based Smart Contract Platforms Without Breaking the TEE" reveals additional vulnerabilities in TEE-based smart contract platforms, including Secret Network. The paper highlights issues such as access pattern leakage, insufficient state consistency mechanisms, and potential backdoors introduced through software update processes. These findings demonstrate that even without breaking the TEE itself, there are still

[1] `https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/stale-data-read-from-xapic.html`

[2] `https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/processor-mmio-stale-data-vulnerabilities.html`

significant privacy and security risks in the implementation and integration of SGX within blockchain systems [53].

These incidents and research findings underscore the importance of not relying solely on the security guarantees of SGX. They highlight the need for additional layers of protection, careful system design, and ongoing security audits in SGX-based systems, especially in high-stakes applications like blockchain platforms.

3.3 Survey of State of the Art TEE-based PPSC Systems

This section examines prominent blockchain platforms that leverage TEEs to implement PPSCs. By analyzing these systems, we gain valuable insights into the current state of the art, identify common approaches and challenges, and establish a context for understanding the innovations proposed by SUAVE. Our analysis focuses on two notable implementations: the Oasis Protocol and Secret Network.

3.3.1 Oasis Protocol

he Oasis Protocol introduces a novel approach to privacy-preserving smart contracts through its ParaTime (Parallel Runtime) architecture. The network's flagship feature is its confidential ParaTime, which leverages TEEs to enable confidential smart contract execution [57].

In a confidential ParaTime, nodes utilize TEEs, primarily Intel SGX, as secure enclaves for smart contract execution. This process ensures data confidentiality by encrypting data before it enters the TEE, decrypting and processing it within the secure environment, and re-encrypting the results before they exit. This mechanism prevents node operators or developers from accessing unencrypted data [57].

The Oasis Protocol's ParaTime architecture allows for modular and scalable privacy-preserving computation. Each ParaTime operates independently, with its own state and execution model, while sharing the network's consensus layer. This separation enables parallel processing of transactions across multiple ParaTimes, enhancing overall network throughput. The confidential ParaTime, leveraging Intel SGX, provides hardware-based isolation for contract execution, ensuring that even the node operator cannot access unencrypted data. Furthermore, the protocol's flexibility allows for the integration of other privacy-enhancing technologies, positioning the Oasis Network as a versatile platform for developing confidential smart contracts and decentralized applications with stringent privacy requirements [57].

3.3.2 Secret Network

Secret Network is a blockchain platform leveraging Intel SGX to provide privacy-preserving smart contracts, known as "Secret Contracts." It enables confidential computation with encrypted input, output, and state, while maintaining the benefits of a decentralized network [58].

In Secret Network's implementation, each validator node employs an Intel SGX CPU, combining consensus and computation layers. Private metadata is encrypted before being sent to validators and is only decrypted inside the TEE, remaining inaccessible to the validators themselves [58].

The network employs a dual core architecture. The Untrusted Core runs the Cosmos SDK and Tendermint, while the Trusted Core executes Secret Contracts and handles SGX-specific mechanisms like Remote Attestation and Sealing [58].

A critical component of Secret Network's security is the consensus encryption seed, stored exclusively inside the TEE of each validator node. This seed is crucial for decrypting private transactions [58].

To reduce potential attack vectors, Secret Network uses SGX-SPS (Server Platform Services). Each full node creates an attestation report proving their CPU uses the latest firmware upgrades before registration. The network verifies this report on-chain to ensure node operators cannot decrypt data without authorization [58].

4 SUAVE Roadmap, Architecture and Mechanisms

Building upon the foundation of PPSC schemes discussed in Chapter 3, this chapter examines SUAVE, a TEE-based PPSC scheme that offers a programmable and general-purpose approach to addressing privacy and efficiency challenges in blockchain systems.

SUAVE, proposed in 2022 by Flashbots who lead its development, represents an innovative platform designed for fast and private off-chain computation. It aims to create a decentralized infrastructure for MEV extraction and block building, while offering a programmable environment for a wide range of applications [4].

This chapter explores SUAVE's implementation roadmap, its evolving architecture, and the key mechanisms underpinning its functionality.

4.1 Implementation Roadmap

The SUAVE protocol implementation follows a strategic, star-themed roadmap. This roadmap is structured around two parallel development tracks, each named after celestial bodies and designed to address specific aspects of the protocol's evolution. The implementation strategy is divided into [59]:

1. **Centauri Track:** This track focuses on exploration and enhancing developer experience. It serves as a specification for the secure instantiation of SUAVE. The Centauri track progresses through three distinct testnets, each building upon the previous one to incrementally increase functionality. These testnets are named after stars in the Alpha Centauri system: Rigil, Toliman, and Proxima.

2. **Andromeda Track:** Running parallel to Centauri, the Andromeda track concentrates on secure instantiation using SGX and advanced cryptography. Its primary emphasis is on enhancing privacy and minimizing trust requirements. This track advances through three key milestones, each progressively improving system hardening and reducing reliance on trusted parties.

The roadmap incorporates strategic convergence points between these tracks, termed "Proto-Collisions." These junctures represent critical phases where functionalities developed in the Centauri testnets can be implemented and executed within the secure, trust-minimized framework established by the Andromeda track [59].

The ultimate goal of this dual-track approach is to reach the "Collision" point. This collision point will result in the Veritate testnet, featuring a network of permissionless

nodes that combine the advanced functionality developed in the Centauri track with the reduced reliance on trusted parties achieved in the Andromeda track [59].

4.1.1 Rigil

Rigil, named after Rigil Kentaurus (Alpha Centauri A), is the initial testnet in the Centauri track of SUAVE development. It represents the first step in a series of testnets based on stars in the Alpha Centauri system, to be followed by Toliman (B) and Proxima Centauri (C).

In the implementation roadmap, Rigil introduces several key components that will be detailed in the next section [59]:

- The first version of the MEVM (Modified Ethereum Virtual Machine)
- Basic precompile governance
- Initial set of MEV-specific precompiles
- An initial confidential data storage system

Rigil provides a developer-focused sandbox for creating SUAPPs in a decentralized and confidential manner. It features a Flashbots-hosted test network for rapid prototyping, using Goerli ETH for gas and operating with a proof-of-authority consensus mechanism. This integration occurs via a dedicated bridge to the Rigil network, which, while not explicitly covered in this thesis, is crucial for enabling asset transfers for gas and MEV applications [59]. The primary goal of Rigil is to gather feedback on developer experience and harden the overall SUAVE software stack. It is designed as a temporary network, intended to be decommissioned after the launch of the subsequent testnet, Toliman. Following Rigil, the implementation roadmap progresses to [59]:

- Toliman: This testnet will introduce advanced confidential storage, improved DoS resistance, and an expanded set of precompiles. Speculatively, based on the roadmap in the SUAVE specifications, Toliman is expected to introduce a dedicated data availability system [60].
- Proxima: The final testnet in the Centauri track will focus on SUAVE chain consensus and performance improvements. Speculatively, based on the roadmap, Proxima may introduce multiple chains with a distributed consensus mechanism, potentially serving the Data Availability use case through bulletin chains.

The technical architecture of Rigil, including SUAVE Kettles, Confidential Data Storage, SUAVE Chain, and MEVM, as well as the advancements in subsequent testnets, will be discussed in detail in the following sections.

4.1.2 Sirrah

Sirrah marks the commencement of the Andromeda track in the SUAVE implementation roadmap. It's important to note that public information about Sirrah is limited, and much of what follows is inferred from context and the SUAVE specifications available on GitHub. According to these specifications, Sirrah is expected to introduce several key components: [59]

- An SGX-based MEVM
- An attestation protocol
- Transport Layer Security (TLS) implementation
- Initial state proofs of data to be placed inside the enclave

These technical foundations represent significant advancements in the secure instantiation of SUAVE, focusing on enhancing privacy and minimizing trust requirements. The specifics of these components and their implementations will be explained in detail in the following sections.

The roadmap suggests that the subsequent milestone, Mirach, will build upon Sirrah's foundations by introducing [59]:

- Key rotation mechanisms
- A key manager node
- An upgrade process, probably for SGX-based operations

It's crucial to emphasize that this information is largely speculative and inferred from publicly available sources. The actual implementation may differ or include additional features not mentioned here. As the SUAVE project progresses, more concrete information about these developments may become available.

4.1.3 Veritate

Veritate represents the culmination of both the Centauri and Andromeda tracks, marking the "Collision" point in the SUAVE implementation roadmap. While detailed information is limited, the roadmap hints at two significant developments for this stage [59]:

- The transition of SUAVE into a permissionless network
- Harmonious integration of a key management protocol utilizing MPC, likely for the secure management of trusted execution environments

These developments suggest that Veritate aims to achieve the ultimate goal of SUAVE: a fully decentralized, secure, and efficient platform for MEV applications. However, as with previous stages, the specifics of this implementation remain speculative and subject to change as the project evolves.

4.2 Rigil Architecture

This section analyzes the architecture of the Rigil Network, the initial testnet implementation of the SUAVE protocol. We focus on Rigil as it offers the first concrete implementation of the SUAVE protocol's concepts. Our analysis draws from available documentation and source code, providing insights into the system's design and abstracted functionality [59].

It's important to note that while our goal is to explain SUAVE functionality as outlined in the roadmap, we must primarily reference SUAVE Rigil, as it's the only version available at this point. We will attempt to extend our discussion to future SUAVE implementations where possible, recognizing that many functionalities are abstracted in Rigil [59].

The Rigil implementation includes several key features that are significant for the current stage but may evolve in future protocol versions. These include a Proof-of-Authority (PoA) consensus mechanism, as SUAVE consensus remains an active area of research; the absence of SGX nodes, with SUAVE Kettles still under development; and limited data availability guarantees, reflecting ongoing research into Heterogeneous Data Availability. These aspects will be discussed further in the following sections, as they represent current abstractions or placeholders for future protocol implementations still under discussion [59].

The Rigil architecture comprises four primary components working in tandem. SUAVE Kettles form a network of actors providing confidential computation for SUAPPs. Confidential Data Storage serves as a secure repository for private data. The SUAVE Chain functions as a public ledger for storing intentionally leaked information, transaction data, and SUAPP logic. Finally, the MEVM, a modified Ethereum Virtual Machine, exposes confidential computation and storage APIs to developers [59].

In the following subsections, we will examine each of these components in detail, exploring their roles and interactions within the broader SUAVE ecosystem. This analysis will help illuminate both the current state of the SUAVE protocol as implemented in Rigil and potential future developments.

4.2.1 SUAVE Chain

The SUAVE chain is a critical component of the SUAVE protocol, designed to achieve and maintain consensus while serving as a platform for storing and broadcasting data.

In its initial development phase, the SUAVE chain operates on a PoA consensus protocol called Clique[1], utilizing a network of permissioned nodes. This design choice enables rapid experimentation and iteration during the protocol's early stages, with the understanding that it may evolve in future testnets [59].

The network is configured with specific parameters, including a Network ID of `16813125`, `5` validator nodes, and a block time of `3` seconds. Clique restricts block

[1] `https://eips.ethereum.org/EIPS/eip-225`

minting to a predefined list of trusted signers, allowing clients to verify each block header against this list [59].

The SUAVE chain supports two primary forms of messages [59]:

1. Standard (legacy) Ethereum transactions: Used for transferring SUAVE-ETH and deploying smart contracts.

2. ConfidentialComputeRequests: A new EIP-2718 message type for interacting with SUAVE smart contracts.

While standard transactions follow familiar Ethereum protocols, ConfidentialComputeRequests[2] employ a unique signing scheme designed to keep confidential inputs off-chain. The ConfidentialComputeRecord[3], signed by the sender, contains only the hash of the confidential inputs, allowing for verification without exposing sensitive data on-chain [59].

Confidential Compute Requests and Records

The SUAVE chain introduces two key structures for handling confidential computations: ConfidentialComputeRecord and ConfidentialComputeRequest.

ConfidentialComputeRecord The ConfidentialComputeRecord serves as a record of computation and is an integral part of both the Confidential Compute Request and Suave Transaction. Its structure is defined as follows [61]:

Listing 4.1: ConfidentialComputeRecord Struct

```
type ConfidentialComputeRecord struct {
    Nonce uint64
    GasPrice *big.Int
    Gas uint64
    To *common.Address
    Value *big.Int
    Data []byte
    KettleAddress common.Address
    ConfidentialInputsHash common.Hash
    ChainID *big.Int
    V, R, S *big.Int
}
```

This structure combines elements of a standard Ethereum transaction (Nonce, GasPrice, Gas, To, Value, Data) with SUAVE-specific fields [61]:

[2] https://github.com/flashbots/suave-geth/blob/main/core/types/confidential.go/
[3] https://github.com/flashbots/suave-geth/blob/main/core/types/confidential.go/

- `KettleAddress`: The address of the Kettle (execution node) that will process this request.
- `ConfidentialInputsHash`: A hash of the confidential inputs, allowing verification without exposing the actual data.
- `ChainID, V, R, S`: Fields used for transaction signing and chain identification.

ConfidentialComputeRequest The ConfidentialComputeRequest structure enables users to request computation over their data via the MEVM using the `eth_sendRawTransaction` method [61]:

Listing 4.2: ConfidentialComputeRequest Struct

```
type ConfidentialComputeRequest struct {
    ConfidentialComputeRecord
    ConfidentialInputs []byte
}
```

This structure embeds the ConfidentialComputeRecord and adds the actual confidential inputs. After processing, the request's ConfidentialComputeRecord is embedded into the SuaveTransaction's ConfidentialComputeRequest field, serving as an on-chain record of computation [59].

The integrity of the computation results is guaranteed by the Kettle's signature. In future implementations, this could be extended to include more advanced integrity proofs, such as zero-knowledge proofs.

SuaveTransaction While not directly visible to end-users, the SuaveTransaction is a crucial internal structure in the SUAVE chain. It serves as the final repository for compute results and intentionally leaked data from confidential compute requests. The SuaveTransaction includes [59]:

- The ConfidentialComputeRecord from the original request
- The result of the confidential computation
- The Kettle's signature, providing an integrity guarantee for the computation results

This structure ensures that the results of confidential computations are properly recorded and verified on the SUAVE chain, maintaining the integrity of the system while preserving the confidentiality of sensitive inputs.

Gas and Transaction Fees

The SUAVE chain employs a gas pricing mechanism similar to Ethereum pre-Cancun hardfork, where gas prices adjust based on network demand. While nodes track gas

usage for Confidential Compute Requests, they currently only charge a small flat fee, with no cap on off-chain compute. SUAVE transactions are initially expressed as Legacy transaction types but are converted to the EIP-1559 base fee model under the hood [59].

4.2.2 MEVM

The MEVM is an extended version of the standard EVM, specifically designed to support SUAVE's new capabilities for confidential computation and interaction with specialized APIs, while maintaining compatibility with existing Ethereum smart contracts [59].

The MEVM operates in two distinct modes: a regular mode equivalent to the standard EVM environment, and a confidential mode accessed through a new transaction type called `ConfidentialComputeRequest`. This confidential mode can be invoked using standard JSON-RPC methods with an additional `IsConfidential` argument [59].

Suave Execution Backend

The `SuaveExecutionBackend`[4] is a crucial component of the MEVM that enables confidential execution capabilities. It provides access to confidential APIs and confidential inputs. This backend is exclusively available during confidential execution, creating a secure environment for sensitive operations. The `SuaveExecutionBackend` serves as the bridge between the standard EVM functionality and the SUAVE-specific features, allowing smart contracts to interact with confidential data and specialized APIs in a controlled manner [61].

Confidential Store

The `ConfidentialStore`[5] is an essential part of the SUAVE chain, designed to facilitate secure and privacy-preserving transactions and smart contract interactions. It functions as a key-value store where users can safely store and retrieve confidential data. Access to the Confidential Store is strictly controlled, with both reading and writing restricted to the allowed peekers [59].

It's important to note that the current implementation of the Confidential Store is not final and may evolve to meet future requirements and incorporate more advanced privacy mechanisms.

[4] `https://github.com/getclave/suave-geth-ethglobal-istanbul/blob/main/core/vm/suave.go#L35`

[5] `https://github.com/getclave/suave-geth-ethglobal-istanbul/blob/main/core/vm/suave.go#L19`

Precompiles

The MEVM introduces several new precompiles[6] that serve as the interface between smart contracts and SUAVE's confidential computation capabilities. These precompiles are implemented using a `SuavePrecompiledContractWrapper`, which allows access to confidential APIs during execution.
Key precompiles include [59]:

ConfidentialInputs *Address: 0x0000000000000000000000000000000042010001*
This precompile provides access to the confidential inputs associated with a confidential computation request. It takes no input parameters and returns the confidential data in bytes format. The function `confidentialInputs()` allows smart contracts to access and process confidential data within the MEVM environment, enabling secure operations on sensitive information.

ConfidentialStore *Address: 0x0000000000000000000000000000000042020000*
The ConfidentialStore precompile manages the storage of data in the confidential store. It requires the caller to be listed in the `AllowedPeekers` for the associated data record, ensuring data privacy. The `confidentialStore()` function allows smart contracts to securely store sensitive data, with access controls enforced at the protocol level. This enables developers to create applications that can safely manage confidential information.

ConfidentialRetrieve *Address: 0x0000000000000000000000000000000042020001*
This precompile facilitates the retrieval of data from the confidential store. Like ConfidentialStore, it mandates that the caller is present in the `AllowedPeekers` list, maintaining data confidentiality. The `confidentialRetrieve()` function enables smart contracts to access stored confidential data, ensuring that only authorized contracts can read sensitive information. This is crucial for maintaining privacy in SUAVE applications.

4.2.3 Kettle

Kettles are the primary actors in the SUAVE network, responsible for processing confidential compute requests and managing confidential data. They integrate the SUAVE chain, Confidential Data Store, and MEVM to handle Confidential Compute Requests, execute confidential transactions, and maintain the Confidential Data Store [59].

While Kettles are not permissioned, participation in the confidential data store synchronization protocol is. It's crucial to note that in the Rigil testnet, Kettles do not operate within TEEs. This current limitation means the security model relies on the honesty of Kettle operators, as a malicious Kettle could potentially censor requests or alter results [59].

[6] `https://github.com/flashbots/suave-geth/blob/b328d64689930a40eae0a6e834805f3feab6b58f/core/vm/contracts_suave.go`

Domain-Specific Services

Domain-Specific Services allow Kettles to utilize separate processes for operations like external API calls, improving efficiency. Kettles are not required to publicly commit to their domain-specific services, resulting in varied support across the network [59]. This flexibility may be beneficial, as not every node might want to commit to external calls, possibly due to legal considerations in different jurisdictions.

Confidential Computation Process

The Confidential Computation Process in a Kettle orchestrates the handling of confidential requests through various components [59]:

1. Developers deploy contracts to the Kettle. If confidentiality functionality or offchain capability is required, developers use SUAVE-provided precompiles in their Solidity code.
2. Users send Confidential Compute Requests to the Kettle's RPC endpoint.
3. The RPC endpoint triggers the MEVM, which interacts with the Suave PoA Chain, Confidential Store, and precompiles to execute the request. The MEVM can decrypt the data for confidential computations.
4. After processing, the MEVM creates a SUAVE transaction containing the results and a signature of integrity. The confidential data itself is stored separately in the Confidential Data Store.
5. If required and supported by the node, Domain-Specific Services handle external API calls or specialized computations.
6. Finally, the confidential compute request result is returned to the request originator.

SUAVE JSON-RPC

The SUAVE JSON-RPC extends the standard Ethereum JSON-RPC interface, enabling seamless integration of confidential compute capabilities while preserving existing Ethereum functionality. This extension introduces specialized methods to handle SUAVE's features [61]:

- **eth_sendRawTransaction**: Enhanced to process signed ConfidentialComputeRequests, allowing for confidential contract interactions.
- **eth_call**: Augmented with confidentiality options, permitting immediate execution of confidential computations without blockchain transactions. It introduces:
  - An IsConfidential flag to activate MEVM's confidential mode
  - An optional ExecutionAddress parameter for specifying the compute executor

- **eth_executionAddress**: A new method that reveals the Kettle's available execution addresses, enabling users to select specific computational resources.

4.3 Sirrah Trusted Execution Model

Sirrah represents a significant milestone in the SUAVE implementation roadmap, serving as a proof-of-concept for the integration of TEEs into the SUAVE protocol. Unlike the Andromeda track where confidential storage and computation are abstracted, Sirrah demonstrates a concrete implementation using Intel SGX [62].

The Sirrah model extends blockchain functionality with confidential computing using TEEs, specifically Intel SGX. It creates a system that runs adjacent to an ordinary blockchain, enhancing its smart contract environment with TEE-based secure computing [62]. This approach builds upon concepts explored in platforms like Secret Network and Oasis, as well as research projects such as Ekiden [52].

At its core, Sirrah's approach involves modifying an ordinary blockchain so that the smart contract VM runs inside a TEE. While a complete codebase for Sirrah was not available at the time of writing this thesis, Flashbots has provided a description of its Proof of Concept that outlines the core principles and architecture of this model [62]. It's important to note that the information presented here is based on publicly available sources and may be subject to change as the SUAVE project evolves.

The following subsections will explore the key aspects of Sirrah's TEE integration and its remote attestation protocol, providing insights into how these components contribute to the overall goal of creating a secure and privacy-preserving blockchain environment.

4.3.1 TEE Integration

The integration of TEEs in blockchain systems necessitates addressing two primary challenges: ensuring secure computation and maintaining compatibility with the underlying network. In the context of Sirrah, this involves creating an environment where the EVM operates within a context reflecting the current blockchain state, while minimizing the Trusted Computing Base (TCB) [62].

Initial investigations explored the implementation of a complete Geth execution client within Intel SGX[7]. This approach, while technically feasible, presented significant challenges. The substantial storage requirements conflicted with TCB minimization goals, and the process was resource-intensive and time-consuming [63]. Furthermore, the risk of information leakage emerged as a critical concern, with potential threats to data confidentiality through memory access patterns within the SGX enclave [63] [64].

To address these limitations, the Sirrah proof of concept uses a light client solution. This approach reduces the TCB footprint and storage requirements while maintaining access to verified state. Light clients download block headers containing summary

[7] `https://github.com/flashbots/geth-sgx-gramine`

information, requesting additional data from full nodes as needed and verifying it against state roots in the block headers [62].

The implementation utilizes Helios[8], a Rust-based Ethereum light client built upon the Rust Ethereum Virtual Machine (REVM). Helios is integrated as a client library within the kettle to further reduce the TCB. The execution layer employs a Remote State DB to interface with Helios, retrieving execution states and performing real-time verification using Merkle proofs [62].

To align with SUAVE network requirements, Sirrah adapts Helios' consensus mechanism from PoS to the Clique PoA. This modification results in a simplified light client consensus layer focusing primarily on block signature verification, with an assumption of fully trusted validators. The syncing process is reduced to fetching the latest block header from the untrusted RPC [62].

The integration is based on REVM[9], selected for its performance characteristics. REVM is modified for SGX enclave execution and enhanced with additional precompiles to create the MEVM required for Sirrah's functionality [62].

Gramine[10] serves as an essential component in this architecture, facilitating the adaptation of existing code for SGX enclave execution. A key function of Gramine is encapsulating binary dependencies into a single root hash via a manifest file. This approach enables the execution of existing applications within SGX with minimal source code modifications, significantly simplifying the process of adapting complex software systems for secure enclave environments [62].

The Sirrah concept, currently in development, demonstrates an approach to combining light client technology with TEE capabilities. It presents a potential method for secure and efficient blockchain execution in trusted environment.

4.3.2 Remote Attestation

Remote attestation is a crucial feature of TEEs, enabling the verification of a TEE's integrity and authenticity. In the context of Intel SGX, this process involves generating a chain of certificates signed by the root of trust (Intel), which can be verified to confirm that a given message was indeed produced by a genuine TEE [56].

The Sirrah concept proposes to implement remote attestation capabilities directly within the smart contract environment. This approach would introduce a new precompile, allowing Solidity code to generate remote attestations. Conceptually, this can be likened to implementing Gramine functionality for Solidity, as it would expose a low-level TEE interface directly to the smart contract environment [62].

A key aspect of this proposed implementation is that while generating an attestation would require an SGX processor, the verification process could occur entirely on-chain. The remote attestation itself would essentially be a certificate chain, with the verifier

[8] `https://github.com/a16z/helios`
[9] `https://github.com/flashbots/revm`
[10] `https://github.com/gramineproject/gramine`

implemented in Solidity. The proof of concept for Sirrah suggests utilizing the Automata-V3-DCAP repository[11], an open-source Solidity library designed for verifying these attestations [65].

It's important to note the evolution of SGX attestation methods. Initially, Intel used the Enhanced Privacy ID (EPID) protocol, which required interaction with Intel's Attestation Service (IAS) for each verification. EPID attestation can be verified using tools like `gramine-sgx-ias-verify-report`[12], which checks the IAS report and verifies that the quote contains expected values [65]. The newer Data Center Attestation Primitives (DCAP) approach aims to reduce this reliance on Intel's services. DCAP uses a more traditional Public Key Infrastructure (PKI) with X.509 certificate chains, allowing for local verification without constant communication with Intel [66].

This shift from EPID to DCAP presents both challenges and opportunities for blockchain-based attestation [66]:

- DCAP reduces dependence on Intel's services, reducing latency in attestation verification.
- However, DCAP introduces complexities in handling certificate chains within the constraints of smart contract execution.
- Implementing DCAP verification in Solidity could provide a portable reference implementation, potentially simplifying the adoption of DCAP in blockchain contexts.
- On-chain storage of TCB information and efficient verification of the certificate chain within smart contracts present new technical challenges.

The proposed implementation of remote attestation in Sirrah would involve several key components [65]:

1. A precompile enabling Solidity code to generate remote attestations.
2. On-chain verification of the attestation using Solidity-based verifiers.
3. Integration of a Solidity library (such as Automata-V3-DCAP) for attestation verification, adapted to handle EPID or DCAP protocol.

This approach, if successfully implemented, would enable blockchain applications to directly verify the integrity and authenticity of TEE-based computations in decentralized systems.

However, it's important to note that this remains a conceptual model, and practical implementation may face various challenges.

[11] `https://github.com/automata-network/automata-dcap-v3-attestation`

[12] `https://gramine.readthedocs.io/en/stable/manpages/gramine-sgx-ias-verify-report.html?highlight=gramine-sgx-ias-verify`

4.3.3 Key Management

The Sirrah Trusted Execution Concept introduces a Key Manager for key management. This approach eliminates the need to generate a new key for each operation, allowing multiple applications and Kettles to use a single, shared key for confidential computations. The proposed Key Manager offers the following features [62]:

1. **Efficient Key Distribution:** The Key Manager can distribute keys to other Kettles, allowing multiple Kettles to join the network.

2. **Optimized Attestation:** Verification of a raw SGX attestation, which is expensive in terms of Solidity gas, needs to occur only once per Kettle. After initial registration, cheaper ordinary digital signatures can be used for subsequent verifications.

3. **Shared Attestation:** A single instance of the Key Manager contract can be used by multiple applications. This allows many applications to use the shared attestation, with each application potentially receiving its own derived key.

This Key Manager design is stated to be a simplified model intended as a starting point for further development. It currently lacks advanced features like enclave upgrades or key revocation, which could be explored in future Solidity implementations [62].

4.4 Mechanisms

The SUAVE protocol introduces several key mechanisms and concepts that are fundamental to its design. This section presents preliminary discussions of these mechanisms and their potential implications for the protocol.

The following subsections will explore the core mechanisms of SUAVE, including the Offchain-Onchain Programming Model, Key Management strategies, Consensus mechanisms, Heterogeneous Data Availability, and Precompile Governance. It's important to note that many of these concepts are still evolving, and their final implementation may differ from the current proposals.

4.4.1 Offchain-Onchain Programming Model

The Offchain-Onchain Programming Model is a key pattern observed in the SUAVE protocol, particularly in the Rigil testnet. This model is expected to persist in future testnet versions and potentially in the final implementation. It facilitates the integration of confidential, off-chain computations with on-chain interactions [67].

The model consists of two main components [67]:

1. **Offchain Computation:** Confidential computations triggered by a Confidential Compute Request, processing sensitive data without exposing it to the public blockchain.

2. **Onchain Interaction:** Selective publication of results or specific outputs from the offchain computation to the blockchain.

The output of these computations is significant for the attestation process, providing a mechanism to verify the integrity of off-chain operations.

A typical implementation of this model in Solidity might look like [61]:

Listing 4.3: Offchain-Onchain ExampleSuapp Contract

```
contract ExampleSuapp is Suapp {
    event OffchainEvent(uint256 num);
    function onchain() public emitOffchainLogs {}
    function offchain() public returns (bytes memory) {
        // Confidential computation here
        emit OffchainEvent(1);
        return abi.encodeWithSelector(this.onchain.selector);
    }
}
```

The `offchain` function represents the confidential computation, while the `onchain` function acts as an interface to the public blockchain, allowing selective publication of results [67]. This model allows applications to [59]:

- Process sensitive data securely off-chain
- Perform computations without burdening the blockchain
- Selectively publish results or proofs for verification
- Maintain confidentiality while interacting with public blockchain infrastructure

4.4.2 Consensus

The Rigil testnet and Sirrah both utilize the EIP-225: Clique PoA consensus protocol. This choice facilitates rapid development and testing with quick block times. Clique PoA operates with trusted signers minting blocks, maintaining the signer list within block headers for efficient light client implementations [59]. Sirrah adapts the Helios light client for this consensus, simplifying the layer by focusing primarily on block signature verification [62].

However, the consensus mechanism for the final SUAVE implementation is still under discussion. Public discussions on this topic are at a very early stage, with limited information available. It's possible that decisions are being made internally within the development team. The community is exploring options that address several key requirements: ensuring data availability for confidential requests and managing the MEVM state.

One promising concept under consideration is the idea of Sharded Inclusion Lists, also known as Bulletin Board Chains. This approach separates data availability from

consensus, using fixed-size references to reduce message sizes. It allows for parallel processing of different data streams [60].

According to the roadmap, the Toliman testnet should introduce the first consensus mechanism with a bulletin board chain. However, it's not yet clear when this will be implemented. As the project is still in its early stages, many aspects of the consensus mechanism remain to be defined and developed [59].

4.4.3 Precompile Governance

The governance process for adding precompiles to the SUAVE protocol is in its early stages and is expected to follow a relatively informal procedure for the Rigil testnet. The process typically involves the following steps:

1. Discussion of the proposed precompile idea in a forum post
2. Opening a pull request (PR) with the implementation
3. Receiving feedback and undergoing review
4. Potential merging and deployment in the next network upgrade

The implementation of new precompiles typically involves a pull request to the 'suave-geth' GitHub repository. However, it's important to note that there is currently no formalized system of SUAVE Improvement Proposals (SIPs) or clear guidelines on how the community or the Flashbots development team would make decisions on proposed changes. The lack of a formalized decision-making process reflects the evolving nature of the project and the ongoing discussions about how best to manage its development.

5 Analysis of the Development Process for SUAVE Applications

The development of applications for the SUAVE protocol requires a unique set of tools and methodologies. This chapter provides a comprehensive examination of the development process for SUAPPs, focusing on the tools, environments, and practices that enable developers to harness SUAVE's capabilities.

5.1 Development Environment

5.1.1 suave-geth

Suave-geth, version v0.2.1 at the time of writing, is the node implementation for the Rigil testnet of the SUAVE protocol. It is a modified version of the geth client that extends functionality to support SUAVE protocol features. Importantly, suave-geth is designed to be compatible with various blockchain networks, including Ethereum testnets, allowing for flexible deployment and testing scenarios [61].

The primary function of suave-geth is to serve as an execution client, processing confidential computations and managing SUAVE-specific operations. Its key features include support for confidential smart contract execution, extended precompiles for MEV functionalities, and dedicated execution capabilities for handling sensitive data and advanced operations [67].

Developers have three main options for setting up suave-geth [61]:

1. Install the latest release binary
2. Build from source
3. Use Docker

The most convenient method is to use the `suave-up` script provided by the SUAVE development team. This bash script not only automates the installation process but also manages updates for suave-geth. It detects the user's system architecture, downloads the appropriate binary, installs it in a designated directory, and configures the user's shell profile. The script works across different Unix-based systems and supports multiple shell environments. By using this script, developers can easily keep their suave-geth installation up to date with the latest features and improvements [61].

To use the script, developers can simply run [61]:

```
curl -L https://suaveup.flashbots.net | bash
```

For those preferring to build from source or use Docker, the suave-geth repository provides detailed instructions for these methods. After installation, developers can start a local SUAVE development network for testing and debugging applications in a controlled environment. This capability is crucial for developing and testing SUAVE applications before deployment on public networks [61].

SUAVE-Specific Command Line Functionality

Suave-geth extends the standard geth functionality with several SUAVE-specific commands and options. These extensions are crucial for developers working with SUAVE applications. Key among these are the `forge` and `spell` commands [67].

The `forge` command is an internal tool for MEVM operations. It includes subcommands to check if the remote Suave node is enabled and to reset the confidential store.

The `spell` command is a helper facility that simplifies contract deployment and interaction with confidential compute requests. It offers two main subcommands [61]:

- `deploy`: Deploys a contract to the SUAVE network.
- `conf-request`: Sends a confidential request to a contract.

These commands support various options, including specifying artifact directories, kettle addresses, private keys for signing, and RPC endpoints. The `conf-request` subcommand also allows for the inclusion of confidential inputs.

The `spell` command can automatically resolve and use contract artifacts from a specified directory (default is "out"), which is particularly useful for contracts compiled with Forge. It also provides detailed logging about transaction status, including hash, block number, and attempts to decode emitted events for more readable output [61].

For example, to deploy a contract:

```
suave-geth spell deploy MyContract.sol:MyContract
```

And to send a confidential request:

```
suave-geth spell conf-request --confidential-input <input> \
<contract_address> 'functionName()'
```

It's important to note that when passing data with the `-confidential-input` option, it should be a hex-encoded string without the `0x` prefix [67].

5.1.2 Development Network

The Rigil Development Network provides developers with a local testing environment that closely mimics the behavior of the Rigil testnet. This network is initialized using a

specific genesis file, `rigil.json`, which configures the initial state of a private Ethereum blockchain named "suave-rigil" [61].

The genesis configuration sets the chain ID to 16813125 and enables various Ethereum upgrades from block 0, including Homestead, EIP150, EIP155, EIP158, Byzantium, Constantinople, Petersburg, Istanbul, and Berlin. It also activates SUAVE-specific features from the genesis block. The Clique PoA consensus mechanism is defined with a block time of 4 seconds and an epoch of 30000 blocks, specifying three initial signers. The configuration sets a low initial difficulty of "1" and a high gas limit of 30,000,000 per block. It preallocates substantial balances to five specific accounts and lists a bootnode to facilitate network connectivity among peers [61].

Developers can start a local development network using the command [67]:

```
suave-geth --suave.dev
```

This command initializes the local blockchain with the specified configurations, setting up an environment optimized for development and testing. When starting the local development network, it automatically funds test accounts, deploys necessary smart contracts (including the MEV-share contract), and prepares the environment for testing various interactions, including confidential transactions and MEV-related functionalities [61].

Suave-geth offers several configuration options for developers. They can specify the confidential storage backend using either a Pebble database or Redis. The suave dev mode can be enabled for a development-friendly environment. Developers can also configure a remote Ethereum endpoint as the backend, set up bundle and block signing keys, and define whitelisted external addresses [61].

Additionally, suave-geth supports various networks including Ethereum mainnet, testnets like Görli, Rinkeby, and Sepolia, as well as the SUAVE-specific Rigil network. This flexibility allows developers to test their applications across different network environments, ensuring compatibility and smooth deployment transitions [61].

5.1.3 Testnet Faucet

The SUAVE Faucet is a tool for developers working on the Rigil testnet, providing a means to obtain test ETH for development and testing purposes. Based on the eth-faucet project by Flashbots[1], which was inspired by the chainflag eth-faucet[2], this web application is designed to distribute small amounts of Ether in test networks [67].

To access the faucet, developers can visit `https://faucet.rigil.suave.flashbots.net` and authenticate using their Twitter or GitHub accounts. This authentication process, combined with rate-limiting implemented via Redis, helps prevent abuse and ensures fair distribution of test ETH.

To obtain test ETH on the Rigil testnet, developers simply need to sign in to the faucet using their preferred authentication method and submit a request. The faucet

[1] `https://github.com/flashbots/eth-faucet`

[2] `https://github.com/chainflag/eth-faucet`, a faucet designed for private networks.

then processes these requests, distributing a predetermined amount of test ETH to the specified address.

It's worth noting that while the faucet is essential for the Rigil testnet, it is not needed when working with the local development network, as the latter comes with pre-funded accounts to facilitate immediate testing and development [61].

5.1.4 External Calls

SUAVE applications have the capability to make external HTTP requests, enabling them to fetch data from various sources and incorporate it into their confidential computations. This feature significantly expands the potential of Suapps, allowing them to interact with external data providers and services [67].

To enable external calls in a local development environment, developers need to start suave-geth with an additional flag [67]:

```
suave-geth --suave.dev --suave.eth.external-whitelist='*'
```

Suapps can utilize this functionality in several ways:

- Fetching blockchain data from other networks (e.g., Ethereum mainnet)
- Interacting with external APIs, such as financial data providers
- Integrating AI services, like ChatGPT, directly within smart contracts

The SUAVE standard library provides utilities like `Gateway.sol` for blockchain interactions and `ChatGPT.sol` for AI integrations. These utilities use the `Suave.doHttpRequest()` precompile to execute external calls securely within the SUAVE environment [67].

When working with Foundry projects, developers should include the following in their `foundry.toml` file to enable external calls [67]:

```
[profile.suave]
whitelist = ["*"]
```

This powerful feature allows developers to create Suapps that combine on-chain logic with off-chain data and services, opening up new possibilities for decentralized applications.

5.2 SUAVE Standard Library

The SUAVE Standard Library (suave-std) is a Solidity library developed to support the creation of Suapps. This library provides a collection of tools and utilities that interface with SUAVE precompiles, facilitating interaction with the SUAVE environment and enabling the development of applications that utilize privacy-preserving features [68].

The library's functionalities can be categorized into four main areas:

- **Core Utilities**: Functions for SUAVE environment interaction, including context management, random number generation, and basic Suapp operations.
- **Blockchain Interaction**: Tools for Ethereum and cross-chain network interactions, supporting transaction handling, block construction, and MEV-related operations.
- **External Service Integration**: Components that enable communication with external services, such as API endpoints and JSON-RPC endpoints.
- **Cryptographic Operations**: Utilities for cryptographic functions, including key management, signing operations, and secure data handling within the SUAVE framework.

The design of the SUAVE Standard Library emphasizes modularity, allowing for selective import and use of specific components. The library's architecture allows for extensibility, enabling developers to augment and customize functionalities for specific use cases. Integration with Ethereum development tools such as Foundry is also a feature of the library's design [68].

Notable components of the library include Context.sol, which provides functions for interacting with the SUAVE context in the MEVM, and Gateway.sol, which facilitates interactions with contracts on other blockchain networks. These components, among others, are designed to utilize the precompile-enabled features unique to SUAVE, such as confidential computing and cross-chain interactions [68].

5.2.1 Client Software Development Kit

The SUAVE project provides two Software Development Kits (SDKs): one implemented in Go and another in TypeScript. This thesis focuses exclusively on the Go SDK, which was utilized in the subsequent chapter's implementation. The Go SDK, part of the suave-geth project, offers a comprehensive set of tools for deploying contracts, sending transactions, and managing interactions within the SUAVE ecosystem [67].

Central to the SDK's architecture is the `Framework` type, which encapsulates the SUAVE environment configuration. This configuration is defined through the `Config` struct with the following key options [61]:

- `KettleRPC`: The RPC endpoint for the SUAVE Kettle (default: "http://localhost:8545").
- `FundedAccount`: A pre-funded account for the SUAVE devnet.
- `L1RPC`: The RPC endpoint for the L1 chain (default: "http://localhost:8555").
- `FundedAccountL1`: A pre-funded account for the L1 devnet.
- `L1Enabled`: A boolean to enable or disable L1 functionality.

The `Framework` type serves as the main entry point for interacting with the SUAVE environment. It provides access to both the SUAVE and L1 chains (if enabled) through the `Suave` and `L1` fields respectively, allowing developers to seamlessly interact with both SUAVE-specific functionalities and standard L1 operations within a unified framework. The SDK includes a `ReadArtifact` function for processing contract artifacts, which contain compiled bytecode and ABIs essential for contract deployment and interaction [61].

Core functionalities provided by the SDK include contract deployment, confidential request transmission, account funding, transaction signing, and RPC client access. These capabilities enable developers to engage with the full spectrum of SUAVE's features, from fundamental transaction processing to sophisticated confidential computations.

5.3 Precompile Development

The development of precompiles is a crucial aspect of extending the SUAVE protocol's functionality. SUAVE utilizes a structured approach for developing custom precompiles, which involves several key steps.

Initially, each precompile is defined in a YAML file, specifying its name, address, input parameters, and output format. This YAML specification serves as the foundation for the precompile's interface. From this specification, two sets of bindings are auto-generated: a Solidity binding (SuaveLib) for client-side interactions, and a Go binding for server-side implementation. More specifically, the Go binding is used within the MEVM itself to handle the Solidity calls to the SUAVE precompiles. These generated bindings abstract away the complexities of encoding/decoding and error management, providing a standardized communication format between Solidity and Go runtimes. Finally, developers implement the actual precompile logic in Go, following the generated interface [61].

To illustrate the precompile development process, we can examine an example from the suave-geth repository: the creation of a simple "add" precompile. This process involves three main steps:

First, we add a new entry to the YAML specification file [61]:

```
functions:
  - name: add
    address: "0x0000000000000000000000000000000042010009"
    input:
      - name: a
        type: uint64
      - name: b
        type: uint64
    output:
      fields:
        - name: output1
```

```
        type: uint64
```

This specification defines the precompile's name, address, input parameters (two uint64 values), and output (a single uint64 value).

Next, we generate the code bindings by running [61]:

```
$ go run suave/gen/main.go --write
```

This command creates the necessary Solidity and Go bindings based on the YAML specification. In the generated Go skeleton, a new `Add` function is created in the `SuaveRuntime` interface:

```
func (b *suaveRuntime) Add(a uint64, b uint64) (uint64, error) {
    return a+b, nil
}
```

This implementation simply returns the sum of the two input parameters. While this example is basic, it demonstrates the process of extending the SUAVE MEVM with custom functionality. In real-world applications, these precompiles can be much more complex, tailored to specific MEV or confidential computing needs [61].

In the context of the Andromeda development path, which focuses on SGX implementation, there's a move towards using the REVM. This implementation leverages Gramine features through the filesystem to provide enhanced security and functionality. Key precompiles in this context include functions for random number generation using Gramine's /dev/urandom, volatile memory operations, SGX attestation using Gramine's remote attestation feature, secure key generation, and HTTPS requests [66].

These Rust-based precompiles implement crucial functionality required for confidential computation and storage, making extensive use of Gramine's capabilities for attestation and secure operations. The Suave.localRandom precompile uses the RDRAND instruction via Gramine's /dev/urandom. Suave.volatileSet/Get uses a static HashMap in local memory for temporary storage. Suave.attestSgx utilizes Gramine's remote attestation feature, while Suave.sealingKey employs Gramine's key management capabilities. Additionally, Suave.doHTTPRequest uses /etc/ssl/ca-certificates.crt for HTTPS certificate validation [69].

6 Analysis of SUAPPs Use-Cases

SUAVE, an evolving protocol in the blockchain ecosystem, represents a significant shift in how we approach blockchain infrastructure and applications. While still in development, its potential impact is already evident from the vision articulated by its creators at Flashbots:

> "We designed SUAVE to be the mempool and block builder for all blockchains."
>
> — Flashbots, "The Future of MEV is SUAVE" [4]

> "The ambition of the MEVM is to offer every primitive of the MEV supply chain as a precompile, allowing any centralized MEV infrastructure today to be transformed into a smart contract on a decentralized blockchain"
>
> — Flashbots, "MEVM: SUAVE, Centauri, and Beyond" [70]

These statements position SUAVE as an integral part of the blockchain stack, envisioned as a plug-and-play mempool. While this perspective often leads to SUAVE being primarily associated with MEV applications, the potential use cases extend far beyond, leveraging not only the privacy but also the integrity of TEE execution that SUAVE provides.

The scope of possible use-cases is largely determined by the precompiles available in the MEVM. This chapter will explore both currently conceptualized use cases and potential applications that could be enabled by proposed or future precompiles. Importantly, SUAVE has the potential to enable applications where end-users may be unaware they're interacting with the protocol. It could serve as an invisible layer, facilitating secure and private information exchange, and ensuring the integrity of code execution without users needing to understand the underlying mechanics.

Initially, SUAVE was targeted at applications with specific requirements [67]:

- Computation on confidential data, such as auctions
- Access to relevant off-chain data, like trading strategies conditional on centralized exchange prices
- Computationally intensive operations that are too expensive to perform on-chain

These capabilities, while achievable through centralized services, align with the decentralized ethos of cryptocurrencies when implemented via SUAVE. The network aims to provide properties like low latency, privacy, credible computation, and composability,

enabling a wide range of applications that previously required centralized infrastructure [67].

In the following sections, we will explore various use cases that demonstrate the potential of SUAVE. We begin with an analysis of how SUAVE can combat Sybil attacks in blockchain airdrops, showcasing its ability to enhance security and fairness in token distribution. Next, we examine the integration of CEX liquidity into DeFi, illustrating SUAVE's capacity to bridge traditional and blockchain-based financial systems. Finally, we delve into the application of machine learning on confidential data, highlighting SUAVE's potential to revolutionize privacy-preserving data analysis in various domains.

6.1 Combating Sybil Attacks in Blockchain Airdrops

Airdrops have emerged as a significant token distribution mechanism in the blockchain ecosystem, serving dual purposes: as a marketing strategy to increase project visibility and as an alternative distribution method that circumvents regulatory challenges associated with Initial Coin Offerings (ICOs). Essentially, airdrops involve the distribution of free tokens to early supporters or potential users of a blockchain project or protocol [71].

While airdrops effectively bootstrap communities and create initial token holders, they are inherently vulnerable to Sybil attacks. In such attacks, malicious actors create multiple pseudonymous identities to gain a disproportionate share of the distributed tokens. This vulnerability presents a significant challenge for blockchain projects aiming to ensure equitable token distribution among genuine users [72].

The fundamental difficulty in mitigating Sybil attacks stems from the pseudo-anonymous nature of blockchain networks. Establishing real-world identity on the blockchain without compromising user privacy presents a complex challenge. Various strategies have been employed by blockchain projects to counter Sybil attacks, including requiring ownership of high-value Non-Fungible Tokens (NFTs), mandating participation in network activities such as staking, or implementing proof-of-personhood systems. However, these methods often prove to be predictable, potentially exclude new or economically disadvantaged users, or introduce additional privacy concerns [73].

This thesis proposes a novel approach to combat Sybil attacks in blockchain airdrops by leveraging TEEs and social media identity verification. The SUAVE architecture, with its capability to execute confidential off-chain HTTP requests, provides a promising framework for implementing this solution. The proposed methodology operates as follows:

1. Users authenticate through a social media platform (e.g., Twitter) using OAuth 2.0 Authorization Code Flow with PKCE [74].
2. The SUAPP front-end manages the initial authentication process, receiving the access token from the social media platform after the OAuth flow is completed.
3. The access token is transmitted via an encrypted confidential compute request to the TEE.

4. Within the secure confines of the TEE, several critical operations occur:
    a) The TEE uses the access token to make HTTP requests to the social media API, retrieving user data while maintaining confidentiality.
    b) The retrieved data is analyzed within the TEE's secure environment.
5. The TEE analyzes various metrics that are difficult to fake, such as:
    - Follower and following counts: These numbers indicate the user's network size and engagement on the platform.
    - Tweet count: This metric reflects the user's activity level and longevity on the platform.
    - Account creation date: Older accounts are generally considered more trustworthy.
    - Location: This can be used to comply with regional regulations.
    - Verification status: Verified accounts have undergone additional identity checks by the platform, providing an extra layer of legitimacy.
    - Interaction Analysis: The TEE could analyze the user's interaction patterns, such as the diversity of accounts they engage with, to further validate authenticity.
6. Based on these metrics, the TEE computes an "airdrop allocation score" for the user, determining the quantity of tokens the user should receive in the airdrop.
7. Secure Data Storage: The calculated score and any sensitive data are stored in an encrypted format within the TEE. Only a minimal hint about airdrop eligibility or potential token amount is emitted.
8. Ephemeral Token Handling: The access token is neither stored nor refreshed after use, minimizing the window of vulnerability and reducing the potential impact of any security breaches.

This approach relies on the temporary confidentiality provided by Intel SGX. Even if a vulnerability were to be discovered later, the risk is minimized because the tokens are not refreshed.

The proposed methodology allows for two distinct approaches to implementing airdrop criteria:

1. Make the criteria public and allow users to audit the SUAPP contract before participating.
2. Obfuscate the criteria by using private Solidity libraries, as demonstrated in the suave-examples GitHub repository[1]. The criteria and airdrop amounts can be disclosed after a set deadline or post-facto.

[1] `https://github.com/flashbots/suapp-examples`

Each approach offers distinct advantages and trade-offs, and the choice depends on the specific goals and trust model of the airdrop campaign.

It's important to note that this approach requires users to trust the SUAPP with their social media access, similar to the trust assumptions when linking social media accounts to third-party applications. However, users can inspect the data payload of the confidential compute request and verify that only the access token is being transmitted to the TEE.

By integrating social media identity verification with SUAVE's confidential computing capabilities, this approach offers a promising solution to combat Sybil attacks in blockchain airdrops. It provides a framework for more nuanced and community-oriented token distribution strategies while maintaining a balance between security and user privacy. Future research could explore the integration of this approach with other identity verification methods and the development of more sophisticated scoring algorithms.

6.2 Bringing CEX Liquidity DeFi and making it Composable

As described, DEXes that employ the CPMM model are predominant mechanisms for token exchange within blockchain ecosystems like Ethereum. These token exchanges allow for perfect composability with other on-chain interactions if conducted within the same block, facilitating atomic operations across trades. However, composability is confined within the blockchain network. Cross-blockchain interactions or transactions involving Layer 2 networks such as rollups necessitate transaction confirmations for secure operations, which limits immediate composability. Similarly, trades on CEXes suffer delays due to off-chain processing and confirmation times, impeding seamless interaction with other blockchain systems.

Addressing these challenges, SUAVE provides a platform that integrates new paradigms and functionalities, redefining the interplay between centralized and decentralized finance. Conceptualized as a decentralized block builder, SUAVE excels in processing signed L1 transactions, performing confidential computations, and enabling verifiable off-chain computations for smart contracts. This unique capability allows SUAVE to function as a bridge that not only extends the realm of DeFi but also integrates and leverages CEX liquidity effectively.

Enhanced Verifiable Off-Chain Pricing Mechanics

In our DEX concept, we initially address the question of establishing a market-reflective price at the moment a user initiates a token exchange. Numerous factors influence the price received by the user, including direct and indirect market impacts such as slippage, MEV activities like front-running, and delays until the transaction is mined. These factors together determine the realized price, which may not always fully account for broader market movements. CEXes not only provide deeper liquidity but also reflect

market changes more swiftly than their decentralized counterparts. In determining the most appropriate price for a DEX transaction, we consider real-time execution prices available on CEXes by examining the depth of their order books. We focus on executable limit orders at any given moment, noting that conditional orders such as One-Cancels-the-Other (OCO) or post-only orders [75] are excluded for simplicitly but might still hold significance on realizable market prices.

SUAVE's capabilities in verifiable off-chain computation allow us to refine the accuracy of DEX pricing significantly. By harnessing off-chain data, SUAVE can mitigate the various adverse impacts on price that are commonly encountered in traditional AMMs. This method moves beyond the typical reliance on internal liquidity reserve ratios for determining prices. Instead, SUAVE ensures the integrity of data fetched from CEX APIs, enabling a more reliable and timely reflection of market conditions. Furthermore, SUAVE reduces other pricing impacts associated with block time delays.

While such a mechanism significantly reduces the dependency on trust compared to trading on a CEX, some trust assumptions remain, primarily related to the integrity of CEX API services and the integrity of network connections. In our reference implementation, we use API calls to Binance to fetch real-time data. To enhance the robustness of our data source, we could aggregate the median value from multiple API calls, adjusting these values based on current liquidity conditions. This method ensures a more stable and reliable price feed, safeguarding against anomalies and potential manipulation.

6.2.1 User Experience

The SUAPP provides a user experience akin to DEX like Uniswap but enhances it significantly by handling transactions through confidential compute requests. Users submit their L1 transactions or token transfers, which are signed and RLP-encoded, as confidential inputs. This method ensures that sensitive transaction details remain secure until they are processed.

The SUAVE node kettle confidentially decodes the submitted transactions and constructs the necessary withdrawal transactions for users to receive the requested funds. These transactions are then grouped into a bundle, which are forwarded to a protected mempool, like Flashbots Protect. The bundle ensures that transactions are not published individually, thus avoiding network risks and ensuring that the user's funds are transferred as part of a secure, atomic bundle execution.

For liquidity providers (LPs), the integration with SUAPP requires uploading their private keys to enable the platform to construct and execute withdrawal transactions immediately. This access allows the SUAPP to manage transactions directly from the LPs' reserves, ensuring secure trade executions for users.

The SUAPP also leverages the high trading volumes typical of market makers on CEXs. This volume advantage allows LP to offers access to more competitive fees to the SUAPP users. As a result, regular users of SUAPP can enjoy lower trading fees, which could be much more favorable than those found on traditional DEXes.

6.2.2 Technical Details and Implementation

Users begin their interaction with the SUAPP frontend by selecting the asset pair they wish to trade, specifying the trade direction, the amount, and the blockchain network on which they prefer to execute the transaction. This selection process, integral to user setup, is akin to what users experience on centralized exchanges (Step 1). Following this, the platform displays active LPs with their associated L1 blockchain addresses to which users will direct their funds for the trade (Step 2).

The next step involves the SUAPP frontend executing an API call to a centralized exchange to retrieve real-time pricing for the chosen asset pair. This price fetching is crucial for ensuring that users can make informed decisions based on current market conditions (Step 3). After reviewing the price, the user finalizes the transaction details and proceeds to sign the RLP-encoded transaction, which is then encrypted and sent as a confidential compute request to the SUAVE network (Steps 4 and 5). This encrypted submission safeguards the transaction's confidentiality until it will be packed inside a bundle and is ready for execution.

In an off-chain precompile execution, the SUAPP obtains the current market price by doing an order book request of a designated CEX selecting the best bids or asks necessary for the amount of the trade. The platform then utilizes the securely stored private key of the LP to initiate a transaction that sends the appropriate amount of the alternate asset from the liquidity reserves, determined by the previously fetched exchange rate (Step 8).

All related transactions are bundled together to ensure atomic execution, as highlighted in Step 11 of the sequence diagram. The SUAPP in theory also allows the inclusion of additional transactions in the bundle at arbitrary positions, enabling users to implement advanced trading strategies such as arbitrage. This feature is particularly advantageous for exploiting arbitrage opportunities across different DEXs and CEXs but also benefiting from the shorter block times of SUAVE compared to traditional L1 block times.

The implementation of this SUAPP, which we named Snapshot Finance due to its use of CEX orderbook snapshots for price determination, was tested on the local SUAVE development network, with the orchestration of L1 transactions demonstrated through interactions with the Sepolia testnet. Two key transactions illustrate this process. In the first transaction[2], a user sends 0.01 ETH to the Liquidity Provider's address on Sepolia. Immediately following this, in the same block, a second transaction[3] is executed. This transaction, constructed within the SUAVE kettle using the LP's private key, transfers 38.120893 USDC tokens back to the user's address. The full implementation details can be found in the GitHub repository[4], showcasing the practical application of SUAVE in creating a bridge between centralized and decentralized liquidity sources.

[2] `https://sepolia.etherscan.io/tx/0xc5aeb994a7ed6078fbe765d0b6866f5ac9edfcee245adb14efbbc748f0f34a69`
[3] `https://sepolia.etherscan.io/tx/0x8e59b7d7d0c4e622c6c6aded931655f5213f2f56aadd4a17b54738c49caa1f2e`
[4] `https://github.com/jonasgebele/snapshot-finance`

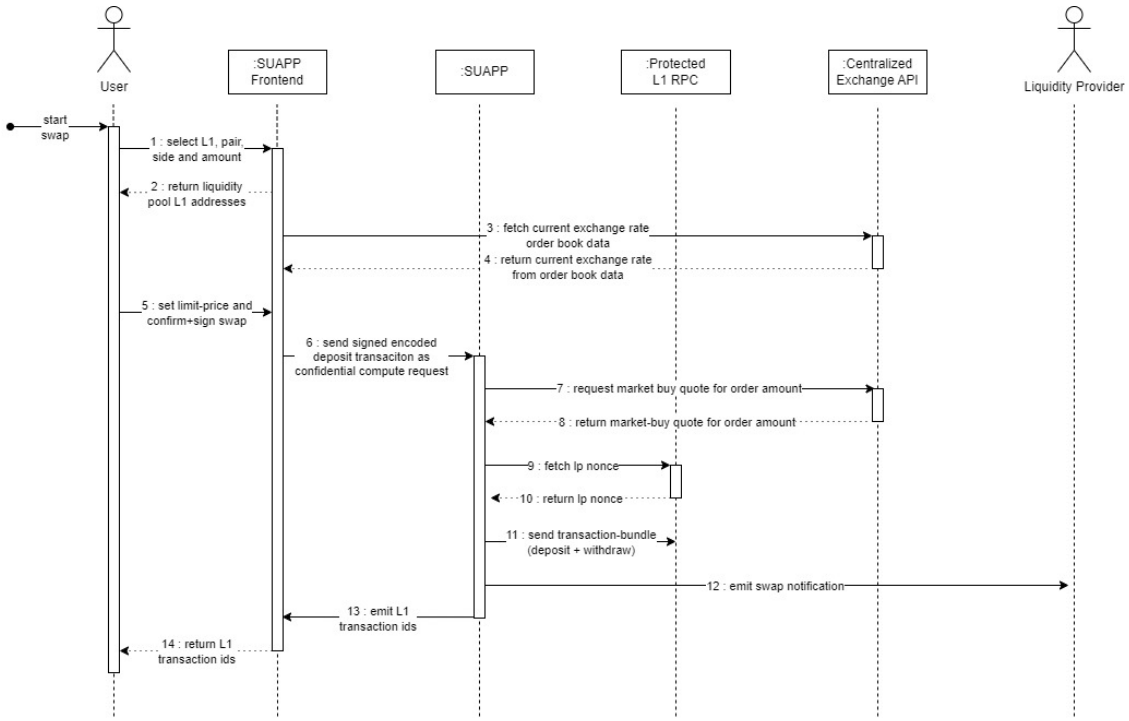

Figure 6.1: Sequence Diagram

Listing 6.1: Contract functoin to execute a swap on Snapshot Finance

```
function makeSuaveDexSwap():
    rlpEncodedInputTxn = getConfInputs()
    txnDetails = decodeRLP(rlpEncodedInputTxn)

    exchangeRate = FetchExchangeRate(txnDetails.amount, CEX_API)
    if exchangeRate < threshold:
        revert("Unacceptable execution rate")

    swapOutputAmount = calculateSwapAmount(txnDetails, exchangeRate)
    privateKey = retrieveLiquidityProviderPrivateKey()

    signedWithdrawTxn = buildWithdrawTxn(txnDetails, privateKey)
    bundleHash = publishTxnBundle(signedWithdrawTxn, RPC_ENDPOINT)

    Emit SwapExecutionEvent()

    return bundleHash
```

The `makeSuaveDexSwap` function, as shown in the pseudo code, initiates by retrieving and decoding the transaction input details through `getConfInputs` and `decodeRLP`

respectively. It then fetches the current exchange rate via `FetchExchangeRate`. If the exchange rate falls below a predefined threshold, the function reverts, ensuring the swap occurs only at acceptable rates. Upon successful validation, the swap output amount is calculated, and the liquidity provider's private key is retrieved. A withdrawal transaction is constructed and signed with the private key, followed by publishing this transaction bundle to the L1 via `publishTxnBundle`. Finally, the function emits a swap execution event and returns the transaction bundle hash, signifying the completion of the swap process.

6.2.3 Role and Incentives of Liquidity Providers

LPs in the SUAPP essentially assume the role of market makers, akin to their function on centralized exchanges. This role involves profiting from the spread of an order book, a common business model employed by various trading firms. In the context of the SUAPP, this involves participating in a system that uses a simulated and replicated order book, allowing LPs to benefit from the price spreads generated by trades. By averaging over multiple order books, LPs have the opportunity to optimize their execution strategies off-chain. This optimization allows for more profitable outcomes by selecting the most advantageous times and marketplaces to execute trades. Essentially, this setup enables them to earn the spread of the averaged order book, mirroring the profit mechanism available to traditional market makers on centralized platforms.

A significant advantage for liquidity providers in SUAPP compared to those on centralized exchanges is the access to exclusive order flow. This exclusivity means that LPs face no competition from other market makers when capitalizing on spreads from specific orders. Such an environment can potentially offer higher profits and more control over trade execution.

Further enhancing the attractiveness of such an SUAPP for liquidity providers is the potential for integrating automated trading capabilities directly within the platform. For instance, the ability to immediately settle trades on centralized exchanges via a signed API calls managed within the smart contract is possible. Additionally, there is the possibility of integrating a fee model similar to those found in CPMM systems like in Uniswap [20]. In such a model, a portion of each trade amount could be allocated as a fee to the LP, incentivizing them to maintain and provide liquidity within the platform.

6.2.4 Risk Assessment

LPs in SUAPP face significant risks related to the security of confidential storage within the TEE, which guards their private keys. This environment, often reliant on system assumptions such as Intel's SGX, is critical for creating the withdrawal transactions to the user. However, it also presents vulnerabilities; if the TEE is compromised, the private key would be exposed which could lead to the immediate draining of LP funds. Unlike auction use cases where only confidentiality might be temporarily compromised, in such a SUAPP, the exploitation could make all assets held by the LP vulnerable to theft.

In addition to security concerns, LPs as market makers depend on the timely and accurate transmission of market data from the SUAVE node. This data flow is crucial for effective market making but may suffer from latency issues that are generally worse than those experienced with CEXs like Binance, which utilize dedicated websocket connections for near-instantaneous data streaming. Delayed order execution notifications could lead to ineffective trading strategies, impacting the profitability and operational efficiency of LPs.

LPs in SUAPP face front running risks due to delays in CEX price feeds, which lag behind real-time transactions. This risk increases if substantial trading volume on DEXes affects market prices before these changes are reflected on CEXs. Such discrepancies can lead to significant losses if large front-running transactions are added to the bundle, manipulating market prices. LPs must manage this risk by selecting asset pairs wisely, avoiding those susceptible to such price manipulation where DEX volumes significantly influence market prices.

Users of SUAPP are at risk from potential collusion and exploits involving block builders. Although SUAPP's design aims for atomic inclusion of transactions to maintain security, malicious builders could manipulate transaction payloads. They might execute transactions that favor liquidity providers while ignoring withdrawal transactions, effectively leading to potential financial losses. This threat, while serious, is mitigated by the reputational damage that would deter such behavior, making it detectable and highly disincentivized. Maintaining strict security protocols and vigilant monitoring is crucial to detect and prevent such manipulative activities, ensuring transaction integrity and user trust.

6.3 Machine Learning on Confidential Data

The increasing generation of sensitive data across various domains presents significant challenges in handling and utilizing this information effectively. Many users are reluctant to upload their personal data to centralized servers due to privacy concerns, especially in domains like healthcare. This reluctance often results in valuable data remaining unutilized or accessible only to device manufacturers.

SUAVE's off-chain capabilities, leveraging TEEs, enable computation on confidential data, including machine learning applications. This provides a framework for handling sensitive information without compromising individual privacy. Users of personal devices that collect sensitive data might contribute to a collective but private data storage system, where their information can be utilized for beneficial purposes without exposure to centralized services.

SUAVE's ability to perform confidential computations off-chain, while maintaining blockchain security guarantees, allows for the development of machine learning models on aggregated data without exposing individual records. This approach enables users to benefit from collective insights while retaining control over their data. The capability extends to any domain where privacy-preserving data analysis is crucial.

In the following subsections, we propose a system design leveraging SUAVE's capabilities for shared machine learning on confidential data. We will explore the technical aspects of implementation within the SUAVE framework and compare this approach with existing privacy-preserving machine learning methodologies like federated learning.

6.3.1 Confidential Data Processing Framework

In designing a system for machine learning on confidential data using SUAVE, we must address several key considerations to ensure data integrity, user privacy, and system reliability.

First, in a permissionless system like applications on public blockchains, ensuring that only valid data is uploaded is crucial. Sophisticated users might attempt to circumvent application restrictions and upload corrupted data, either through duplicates or intentionally wrong values, to manipulate the dataset. To mitigate this risk, the easiest approach could be for device manufacturers to whitelist certain customer IDs and map them to wallet addresses of confidential compute requests, allowing only one upload or data replacement per customer ID. This method, while straightforward, presents a starting point for data integrity. Further research into more efficient and novel ways to prevent data spamming attacks from sophisticated actors is necessary to enhance the system's robustness.

The system relies on the trust assumptions of Intel SGX to ensure data confidentiality. Users' data remains confidential within the TEE-based blockchain's confidential storage, encrypted and inaccessible even to those operating the storage infrastructure. This confidentiality is crucial in building trust and encouraging user participation.

To make the system viable, it should incorporate an incentive structure that motivates users to contribute their data. By storing data in the confidential storage, users could gain the right to perform inference on the collective dataset, accessing insights from both their own and aggregated data. This approach provides a compelling reason for users to participate, as they receive valuable broader insights in return for their contribution.

When a user initiates an inference request, the system could trigger a workflow that either retrains the model or performs inference on the existing model. The results of these operations would be encrypted within the TEE, ensuring that only the requesting user can decrypt and access the insights.

It's important to note that extracting information from trained models remains a potential privacy concern. Techniques like model inversion and membership inference attacks can potentially reveal sensitive information about the training data. Addressing these vulnerabilities is an ongoing area of research, with methods such as differential privacy showing promise. However, for the scope of this thesis, we acknowledge these concerns without delving into their specific implementations, as they represent a complex area of study in their own right.

This decentralized approach stands in contrast to centralized services that often store and process data primarily to gain control over it or to sell services back to users. Instead,

our proposed system creates an environment where users are incentivized to upload their personal data to gain valuable insights, all while maintaining confidentiality and not having to trust a third party with their sensitive information.

6.3.2 Implementation Considerations

The conceptual implementation of our machine learning system on confidential data using SUAVE involves three main functionalities:

1. **Users Uploading Their Data:** This process involves storing the user's data in the confidential storage within the TEE. The implementation should allow users to upload their data securely, remove it if desired, and grant them the privilege to request inference on the existing model.

2. **Building the Model:** A function call could trigger the training of a simple model such as linear regression or logistic regression, using the data collected in the confidential storage. After training, the model parameters are stored back in the confidential storage for future use.

3. **Performing Inference:** Users can request inference on their data using the trained model. This process involves retrieving the model from confidential storage, applying it to the user's data, and returning the with the users public key encrypted result.

From a SUAVE perspective, the implementation relies on two primary services:

1. Offchain Machine Learning Computation: This is implemented as a precompile within the SUAVE node. It handles the actual training and inference processes, operating on data within the TEE.

2. Confidential Data Storage: This service, provided by the TEE-based blockchain system, ensures that all user data and model parameters remain encrypted and secure throughout the process.

From an EVM perspective, several challenges arise in handling the data types required for machine learning, particularly due to Solidity's limitations. Solidity does not natively support floating-point numbers. To address this limitation, we implement a convention for representing floating-point numbers as byte arrays. This allows for flexible data handling while staying within the EVM's capabilities.

The precompile will accept the data encoded in byte arrays and handle the parsing and conversion. It transforms these byte arrays into appropriate numerical representations and then performs the necessary computations for linear regression or logistic regression. This separation of concerns leverages the strengths of both the EVM for smart contract logic and off-chain computation for numerical processing.

After computation, the precompile can return the model parameters or the inference values as strings and return them to the user or store them in the confidential storage

again. This approach ensures that all numerical operations are handled efficiently off-chain while maintaining the security and integrity of the blockchain system. By implementing these functionalities and addressing the EVM challenges, we create a system that maintains data confidentiality, enables machine learning on collective data, and provides users with valuable insights while preserving their privacy.

6.3.3 Evaluation Against Federated Learning

Federated learning and our TEE-based approach represent different strategies for privacy-preserving machine learning, each with distinct characteristics and use cases.

Federated learning is typically employed when data must remain distributed across multiple devices or organizations. In this method, only model updates (typically gradients) are shared with a central server, minimizing data exposure. However, it requires complex infrastructure to coordinate the learning process across distributed nodes [76].

Our TEE-based approach, in contrast, offers a simpler implementation for small datasets, allowing users to retain ownership of their data while enabling decentralized processing. It's particularly suitable for scenarios where strong confidentiality guarantees are crucial, and the simplicity of setup is valued. Key differences between the two approaches include [76]:

1. Infrastructure Complexity: Federated learning requires a sophisticated setup to manage distributed learning, while our TEE-based system offers a more straightforward implementation.

2. Computational Capabilities: Federated learning can leverage accelerators like GPUs across its network, whereas our TEE approach is limited to CPU processing and smaller datasets.

3. Data Control: Both approaches prioritize user control over data, but our TEE system provides stronger confidentiality guarantees within its processing environment.

4. Scale: Federated learning is suitable for large-scale, highly distributed data sources, while our TEE approach is more appropriate for smaller-scale applications with sensitive data.

5. Model Complexity: Federated learning can potentially handle more complex models, whereas our TEE approach is limited to simpler models due to hardware constraints and blockchain latency.

In summary, both approaches contribute to the decentralization of machine learning on private data, but in different ways. While federated learning offers a solution for large-scale, distributed privacy-preserving machine learning [76], our TEE-based approach introduces a novel paradigm for smaller, sensitive datasets. It enables users to

collectively contribute to machine learning models without relinquishing control of their data to a centralized authority. This approach not only preserves user privacy but also democratizes the process of deriving insights from collective data, potentially opening up new possibilities for collaborative, privacy-preserving analytics in various domains.

7 Discussion

The SUAVE protocol represents a notable advancement in blockchain technology, particularly in its approach to confidential computing and credible off-chain computation. These capabilities have the potential to significantly expand the scope of blockchain applications.

Our analysis suggests that SUAVE's development is progressing along two primary trajectories. One perspective focuses on utilizing SUAVE's confidentiality guarantees primarily for MEV applications, aiming to enhance and decentralize existing services within the blockchain ecosystem, particularly addressing the centralizing forces in Ethereum's MEV supply chain. Alternatively, SUAVE could evolve into a comprehensive blockchain platform with privacy-preserving and verifiable off-chain computation capabilities, enabling new classes of applications while indirectly addressing MEV-related centralization.

These approaches represent a spectrum of potential development paths for SUAVE, from becoming an additional layer in the blockchain stack facilitating L1 interactions to evolving into a standalone blockchain with native tokens and applications.

SUAVE's capabilities could significantly impact the broader blockchain ecosystem, offering alternatives to existing crypto business models, particularly in areas requiring trust in centralized entities, such as cross-chain bridges and oracle services. Its ability to interact with traditional financial services through API integrations could potentially bridge the gap between centralized and decentralized finance.

However, SUAVE faces limitations. As a global mempool and sequencing layer, it cannot enforce transaction inclusion on other blockchains. More critically, the security guarantees provided by Intel SGX raise concerns given its history of vulnerabilities. Developers should operate under the assumption that confidentiality breaches may occur. The system's reliance on Intel, particularly for attestation, is problematic and ideally should be independent.

Confidentiality alone may not ensure widespread adoption, as evidenced by other privacy-preserving blockchain systems. However, SUAVE's close integration with the Ethereum ecosystem might provide an advantage.

To enhance SUAVE's capabilities for verifiable off-chain use cases, integrating alternative virtual machines like Solana's could enable the use of Rust and its extensive libraries, leveraging existing toolchains for more diverse and efficient off-chain computations. However, if SUAVE remains focused on MEV applications and mempool management, the current MEVM might suffice.

The governance of precompiles and the process for introducing new ones is crucial. Community-driven governance could ensure SUAVE's development aligns with user

needs and supports a wider range of applications, determining its future functionalities and use cases.

In conclusion, SUAVE presents an innovative approach to blockchain privacy and new application types. Its future hinges on effectively addressing challenges in security, trust, and development strategy.

8 Conclusion

This thesis has conducted a comprehensive analysis of SUAVE, examining its current architecture, the development process of SUAPPs, and potential use cases. SUAVE represents an innovative approach in blockchain infrastructure, integrating confidential computing and verifiable off-chain computation through the use of TEEs.

The SUAVE implementation roadmap, structured around the Centauri and Andromeda tracks, demonstrates a strategic approach to balancing rapid development with the integration of advanced cryptographic and security mechanisms. Our analysis of SUAVE's architecture revealed its potential for executing confidential computations while preserving blockchain integrity and transparency. The current development toolkit for SUAPPs, while still in its early stages, provides a foundation for experimentation and innovation within the SUAVE ecosystem.

Our exploration of potential SUAPP use cases unveiled several promising applications. SUAVE's architecture may enable the decentralization of processes that currently rely on trusted centralized entities for temporal confidentiality. The range of applications - from mitigating Sybil attacks in blockchain airdrops to integrating centralized exchange liquidity with decentralized finance, and enabling privacy-preserving machine learning - illustrates SUAVE's potential to expand the scope of blockchain technology.

However, our research also identified areas requiring further development. The current reliance on Intel SGX for TEEs introduces potential vulnerabilities and centralization concerns that need to be addressed as the protocol matures. The consensus mechanism for SUAVE is still evolving. Furthermore, the governance process for introducing new precompiles and making protocol-level decisions would benefit from formalization to ensure transparent community involvement.

In conclusion, SUAVE presents a notable advancement in blockchain technology, offering potential solutions to challenges related to centralization in the MEV supply chain, privacy preservation, and scalability. Its approach to bridging centralized and decentralized systems could have significant implications for the blockchain ecosystem. As SUAVE continues to evolve, addressing the identified challenges and further exploring its capabilities across various blockchain applications will be crucial.

List of Figures

Bibliography

[1] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability". In: *IEEE Symposium on Security and Privacy (SP)* (2019).

[2] S. Yang, F. Zhang, K. Huang, X. Chen, Y. Yang, and F. Zhu. "Sok: MEV Countermeasures: Theory and Practice". In: *Conference on Advances in Financial Technologies* (2022).

[3] L. Heimbach, L. Kiffer, C. F. Torres, and R. Wattenhofer. "Ethereum's Proposer-Builder Separation: Promises and Realities". In: *IMC '23: Proceedings of the 2023 ACM on Internet Measurement Conference* (2023).

[4] Flashbots. "The Future of MEV is SUAVE". In: *Flashbots Writings Website* (2022).

[5] S. Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". In: (2008).

[6] V. Buterin. "Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application". In: *https://ethereum.org/en/whitepaper/* (2013).

[7] C. Dwork and M. Naor. "Pricing via Processing or Combatting Junk Mail". In: *Advances in Cryptology — CRYPTO' 92, 1993, Volume 740* (1993).

[8] G. Wood. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* (2014).

[9] E. Foundation. "Ethereum Proof-of-Stake Consensus Specifications". In: *https://ethereum.github.io/consens specs/* (2024).

[10] D. Grandjean, L. Heimbach, and R. Wattenhofer. "Ethereum Proof-of-Stake Consensus Layer: Participation and Decentralization". In: *CoRR* (2023).

[11] B. Edgington. "Upgrading Ethereum - A technical handbook on Ethereum's move to proof of stake and beyond". In: *https://github.com/benjaminion/upgrading-ethereum-book* (2023).

[12] C. Brennan. "My Journey to Being a Validator on Ethereum 2.0, Part 5". In: *Consensys Blog* (2022).

[13] "go-ethereum Documentation". In: *https://geth.ethereum.org/docs* (2024).

[14] bitfly GmbH. "Ethereum Mainnet Statistics". In: *https://ethernodes.org/* (2024).

[15] S. Authors. "Solidity Documentation". In: *https://docs.soliditylang.org* (2023).

[16] "Viper Documentation". In: *https://ethereum-viper.readthedocs.io/* (2024).

[17] A. M. Antonopoulos and G. Wood. "Mastering Ethereum". In: *O'Reilly Media, Inc.* (2019).

[18] "Cryptocurrency Prices by Market Cap". In: *https://coinmarketcap.com* (2024).

[19] D. Engel and M. Herlihy. "Composing networks of automated market makers". In: *3rd ACM Conference on Advances in Financial Technologies* (2021).

[20] H. Adams, N. Zinsmeister, and D. Robinson. "Uniswap v2 Core Whitepaper". In: *https://uniswap.org/whitepaper.pdf* (2020).

[21] "EtherDelta". In: *https://etherdelta.com (now https://buycoinnow.com)* (2018).

[22] "0x Protocol". In: *https://0x.org* (2024).

[23] "IDEX". In: *https://idex.io* (2024).

[24] "ParaDEX". In: *https://www.paradex.trade* (2024).

[25] L. Zhou, K. Qin, A. Cully, B. Livshits, and A. Gervais. "On the Just-In-Time Discovery of Profit-Generating Transactions in DeFi Protocols". In: *IEEE Symposium on Security and Privacy (SP)* (2021).

[26] B. Weintraub, C. F. Torres, C. Nita-Rotaru, and R. State. "A Flash(bot) in the Pan: Measuring Maximal Extractable Value in Private Pools". In: *Proceedings of the 22nd ACM Internet Measurement Conference* (2022).

[27] J. R. Jensen, V. von Wachter, and O. Ross. "WIP: Are Builders Using Multi-block MEV (MMEV) Opportunities?" In: *arXiv preprint arXiv:2303.04430* (2023).

[28] M. Bahrani, P. Garimidi, and T. Roughgarden. "Centralization in Block Building and Proposer-Builder Separation". In: *Published in arXiv.org* (2024).

[29] V. Buterin. "Endgame". In: *https://vitalik.eth.limo/* (2021).

[30] D. Mancino, A. Leporati, M. Viviani, and G. Denaro. "Exploiting Ethereum after "The Merge": The Interplay between PoS and MEV Strategies". In: *ITASEC: The Italian Conference on CyberSecurity* (2023).

[31] "Flashbots Documentation". In: *https://docs.flashbots.net/* (2024).

[32] Z. Li, J. Li, Z. He, X. Luo, T. Wang, X. Ni, W. Yang, X. Chen, and T. Chen. "Demystifying DeFi MEV Activities in Flashbots Bundle". In: *Proceedings of the 2023 ACM SIGSAC* (2023).

[33] I. Miers, C. Garman, M. Green, and A. Rubin. "Zerocoin Anonymous Distributed E-Cash from Bitcoin". In: *IEEE Symposium on Security and Privacy* (2013).

[34] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *IEEE Symposium on Security and Privacy* (2014).

[35] H. Qi, M. Xu, D. Yu, and X. Cheng. "SoK Privacy-Preserving Smart Contract". In: *High-Confidence Computing 4(12):100183* (2023).

[36] A. A. Omar, A. K. Jamil, A. Khandakar, A. R. Uzzal, R. Bosri, and N. Mansoor. "A Transparent and Privacy-Preserving Healthcare Platform With Novel Smart Contract for Smart Cities". In: *IEEE Access, vol. 9, pp. 90738-90749* (2021).

[37] N. Hynes, D. Dao, D. Yan, R. Cheng, and D. Song. "A demonstration of sterling: a privacy-preserving data marketplace". In: *Proceedings of the VLDB Endowment, Volume 11, Issue 12* (2018).

[38] S. Tan, X. Wang, and C. Jiang. "Privacy-Preserving Energy Scheduling for ESCOs Based on Energy Blockchain Network". In: *Privacy-Preserving Energy Scheduling for ESCOs Based on Energy Blockchain Network* (2019).

[39] T. Kerber, A. Kiayias, and M. Kohlweiss. "Kachina – Foundations of Private Smart Contracts". In: *IEEE Computer Security Foundations Symposium* (2021).

[40] P. Ç. D. Cnudde. "Analyzing Privacy-Preserving Smart Contracts". In: *Ku Leuven Master Thesis* (2023).

[41] T. P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Advances in Cryptology* (1991).

[42] H. Brenner, V. Goyal, S. Richelson, A. Rosen, and M. Vald. "Fast Non-Malleable Commitments". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015).

[43] "Keep Network". In: *https://keep.network/* (2024).

[44] "Threshold Network". In: *https://threshold.network* (2024).

[45] "Secret Network". In: *https://scrt.network* (2024).

[46] C. Baum, I. Damg, and C. Orlandi. "Publicly Auditable Secure Multi-Party Computation". In: *Security and Cryptography for Networks* (2014).

[47] B. B. S. A. M. Z. D. Boneh. "Zether: Towards Privacy in a Smart Contract World". In: *Financial Cryptography and Data Security* (2020).

[48] "JPMorgan Adds Privacy Features to Ethereum-Based Quorum Blockchain". In: *CoinDesk by Ian Allison* (2019).

[49] S. G. S. M. C. Rackoff. "The knowledge complexity of interactive proof-systems". In: *STOC 85: Proceedings of the seventeenth annual ACM symposium on Theory of computing* (1985).

[50] A. K. A. M. E. S. Z. W. C. Papamanthou. "Hawk: The Blockchain Model of Cryptography andPrivacy-Preserving Smart Contracts". In: *IEEE Symposium on Security and Privacy* (2016).

[51] A. B. M. C. H. Tewari. "zkHawk: Practical Private Smart Contracts from MPC-based Hawk". In: *3rd Conference on Blockchain Research and Applications for Innovative Networks and Services (BRAINS)* (2021).

[52] F. Z. W. H. R. C. J. K. N. H. N. J. A. J. A. M. D. Song. "Ekiden: A Platform for Confidentiality-Preserving,Trustworthy, and Performant Smart Contracts". In: *IEEE Security and Privacy, Volume 18, Issue 3* (2020).

[53] N. J.-L. Y. L. Y. J. H. M. T. Y. S. B. A. Miller. "SGXonerated: Finding (and Partially Fixing) Privacy Flaws inTEE-based Smart Contract Platforms Without Breaking the TEE". In: *Proceedings on Privacy Enhancing Technologies* (2024).

[54] S. Software and S. Lab. "Intel SGX 101". In: *https://sgx101.gitbook.io/* (2024).

[55] V. C. Devadas. "Intel SGX Explained". In: *SCA 2015 tutorial* (2015).

[56] "Intel Software Guard Extensions Developer Guide". In: *Intel Corporation* (2018).

[57] "Oasis Documentatoin". In: *https://docs.oasis.io/* (2024).

[58] "Secret Network Graypaper". In: *https://scrt.network/graypaper/* (2024).

[59] Flashbots. "SUAVE Protocol Specifications". In: *https://github.com/flashbots/suave-specs* (2024).

[60] dmarz on HackMD. "WIP: Bulletin Chains (Sharded Inclusion Lists)". In: *https://hackmd.io/BALXiGhq* (2023).

[61] Flashbots. "Suave Geth Source Code Repository". In: *https://github.com/flashbots/suave-geth* (2024).

[62] A. Miller. "Sirrah: Speedrunning a TEE Coprocessor". In: *https://writings.flashbots.net/* (2024).

[63] C. H. Paape. "Block Building inside SGX". In: *https://writings.flashbots.net/* (2023).

[64] A. K. M. M. A. S. S. K. Chen. "SGX-MR: Regulating Dataflows for ProtectingAccess Patterns of Data-Intensive SGXApplications". In: *Proceedings on Privacy Enhancing Technologies* (2021).

[65] socrates1024 (Andrew Miller). "Demystifying remote attestation by taking it on-chain". In: *https://collective.flashbots.net/* (2023).

[66] "Gramine documentation by Gramine Contributors". In: *https://gramine.readthedocs.io/en/stable/index.h* (2023).

[67] Flashbots. "SUAVE Developer Documentation". In: *https://suave-alpha.flashbots.net/* (2024).

[68] Flashbots. "Suave Standard library source code repository". In: *https://github.com/flashbots/suave-std* (2024).

[69] Flashbots. "Anrdomeda REVM source code repository". In: *https://github.com/flashbots/suave-andromeda-revm* (2024).

[70] R. Miller. "The MEVM, SUAVE Centauri, and Beyond". In: *Flashbots Writings Website* (2023).

[71] C. R. Goforth. "It's Raining Crypto: The Need for Regulatory Clarification When It Comes to Airdrops". In: *Indian Journal of Law and Technology* (2019).

[72] J. Chong. "The Three Tokenomics Problems and a Productivity-Linked Tokenomics Design". In: *Available at SSRN: https://ssrn.com/abstract=4071089* (2022).

[73] P. S. C. M. D. Pate. "Preventing sybil attack inblockchain using distributed behavior monitoring of miners". In: *10th International Conference on Computing, Communication and Networking Technologies* (2019).

[74] X. Corp. "X Developer Platform Documentation". In: *https://developer.x.com/en* (2024).

[75] "Binance Academy - Understanding the Different Order Types". In: *https://academy.binance.com/* (2024).

[76] L. L. Y. F. M. T. K.-Y. Lin. "A review of applications in federated learning". In: *Computers and Industrial Engineering* (2020).