SOVR Hybrid Engine V2: Product Requirements Document

The SOVR Hybrid Engine V2 represents a sophisticated decentralized finance infrastructure that bridges traditional finance with blockchain technology through a comprehensive smart contract ecosystem. This system enables hybrid liquidity management, synthetic fiat token operations, and programmable pool mechanics while maintaining robust security through role-based access control and cryptographic attestation mechanisms.

System Architecture Overview

The SOVR Hybrid Engine V2 comprises nine interconnected smart contracts, each serving distinct but complementary functions within the broader ecosystem. The architecture follows a modular design pattern that separates concerns between routing, liquidity management, reserve operations, oracle functionality, and governance. This separation ensures maintainability, upgradeability, and security while enabling complex DeFi operations.

Core Routing Layer
SOVRHybridRouter_v2 serves as the primary entry point for all swap and liquidity operations, integrating with Uniswap V3 protocols

Liquidity Management
Private and programmable pools enable dynamic pricing, position management, and sophisticated liquidity strategies

Synthetic Assets
sFIAT token contract provides role-based minting and burning capabilities for synthetic fiat representation

Attestation Layer
Credit event registry and oracle mechanisms ensure verifiable off-chain to on-chain settlement integrity

Technical Implementation Standards

All contracts within the SOVR Hybrid Engine V2 system adhere to industry-standard practices and utilize proven libraries for mathematical operations and security. The implementation leverages Solidity version 0.8.x with SPDX-License-Identifier specifications for licensing clarity. Critical dependencies include OpenZeppelin contracts for access control and reentrancy protection, Uniswap V3 core and periphery contracts for liquidity operations, and specialized libraries such as FullMath for precise arithmetic and Babylonian methods for square root calculations in pricing algorithms.

Solidity Standards

- Solidity version 0.8.x with compiler optimizations
- SPDX licensing identifiers on all contracts
- Comprehensive NatSpec documentation
- Event emission for all state changes

Security Practices

- ReentrancyGuard on all external functions
- Role-based access control (RBAC) patterns
- EIP-712 typed structured data signing
- Mathematical overflow protection via SafeMath

Contract 1: SOVRHybridRouter_v2

The SOVRHybridRouter_v2 contract functions as the system's primary orchestration layer, providing a unified interface for users to interact with various liquidity pools and execute token swaps. This router abstracts the complexity of direct pool interactions while implementing critical safety mechanisms including slippage protection, deadline enforcement, and reentrancy guards. The contract integrates seamlessly with Uniswap V3's SwapRouter through the exactInputSingle function, enabling precise single-hop token swaps with optimal gas efficiency.

01
Swap Execution
Utilizes Uniswap V3 SwapRouter exactInputSingle for atomic token exchanges with configurable slippage tolerance and deadline parameters

02
Liquidity Operations
Provides addLiquidity user flow with comprehensive safety checks, balance verification, and position validation before execution

03
TWAP Integration
Exposes time-weighted average price consultation helpers for informed trading decisions and price manipulation resistance

04
Security Layer
Implements reentrancy protection, input validation, and authorization checks across all public-facing functions

Key Functionalities

The router contract implements several critical functions that enable sophisticated DeFi operations. The swap functionality accepts parameters including input token address, output token address, swap amount, minimum output amount for slippage protection, recipient address, and transaction deadline. The addLiquidity function orchestrates multi-step operations including token approval, pool initialization verification, price range calculation, and NFT position minting. All functions emit detailed events capturing transaction parameters, execution results, and any state changes for off-chain monitoring and analytics purposes.

> **Reentrancy Protection:** All external functions utilize OpenZeppelin's ReentrancyGuard modifier with the nonReentrant decorator to prevent reentrancy attacks, which historically have resulted in significant financial losses across DeFi protocols.

Dependencies and Interfaces

Uniswap V3

- ISwapRouter
- INonfungiblePositionManager
- IUniswapV3Factory

OpenZeppelin

- ReentrancyGuard
- IERC20
- SafeERC20

Custom Contracts

- TWAPHelper
- SOVRPrivatePool
- SOVRProgrammablePool

Contract 2: SOVRPrivatePool

SOVRPrivatePool implements a sophisticated private liquidity pool with dynamic pricing mechanisms and advanced liquidity management capabilities. This contract leverages Uniswap V3's concentrated liquidity model to create custom pool instances with specific fee tiers and tick spacing configurations. The pool supports liquidity seeding through NFT positions, enabling granular control over capital allocation across different price ranges. The contract utilizes precise mathematical computations through the FullMath library for overflow-safe multiplication and division operations, and the Babylonian library for efficient square root calculations required in pricing algorithms.

Pool Initialization
Creates and initializes Uniswap V3 pools with specified token pairs, fee tiers, and initial price ratios using the factory pattern

Liquidity Seeding
Mints NFT positions representing liquidity contributions with precise tick ranges and amount specifications

Swap Helpers
Provides optimized swap routing functions with price impact calculations and slippage protection mechanisms

Mathematical Precision

The SOVRPrivatePool contract implements mathematically rigorous calculations for liquidity operations, price discovery, and fee accrual. The FullMath library provides 512-bit precision for intermediate multiplication results, preventing overflow in scenarios involving large token amounts or high-value positions. The Babylonian square root method enables efficient computation of geometric mean prices and liquidity amounts from token reserves. These mathematical foundations ensure accurate pricing across all market conditions and prevent rounding errors that could be exploited or lead to systematic value leakage.

Library	Purpose and Implementation
FullMath	Provides mulDiv function for precise multiplication followed by division without intermediate overflow, critical for calculating liquidity amounts and fees
Babylonian	Implements Newton's method for square root calculation, used in determining optimal liquidity positions and price ratios
TickMath	Converts between tick indices and price ratios using logarithmic scaling, enabling precise price range specifications

Contract 3: SOVRProgrammablePool

The SOVRProgrammablePool introduces governance and automation capabilities through a sophisticated role-based access control system. This contract distinguishes between guardian roles with administrative privileges and keeper roles responsible for automated maintenance operations. The programmable nature enables dynamic strategy adjustments, automated rebalancing, and complex position management workflows that adapt to market conditions without requiring manual intervention.

1
Position Minting
Guardians can initiate new liquidity positions with specific parameters including token amounts, price ranges, and position identifiers

2
Repositioning Requests
Automated systems can request position adjustments based on market conditions, subject to guardian approval and safety checks

3
Bonding Operations
Specialized minting operations that lock liquidity for specified periods, enabling incentive mechanisms and long-term capital commitment

Role-Based Access Control

The access control system implements OpenZeppelin's AccessControl pattern with custom role definitions. Guardian roles possess elevated privileges for critical operations including pool parameter adjustments, emergency pauses, and fund recovery mechanisms. Keeper roles enable automated bots and monitoring systems to perform routine maintenance tasks such as position rebalancing, fee collection, and liquidity optimization. The contract emits comprehensive events for all role modifications and privileged operations, ensuring full transparency and auditability of governance actions.

Contract 4: sFIAT Token

The sFIAT contract implements a synthetic fiat token that represents off-chain fiat currency reserves on the blockchain. This ERC-20 compliant token incorporates role-based minting and burning mechanisms to ensure only authorized entities can increase or decrease the token supply. The contract maintains a one-to-one value peg with its underlying fiat currency through strict collateralization requirements enforced at the protocol level. Minting operations occur when fiat reserves increase, while burning operations reduce supply when reserves are withdrawn or redeemed.

ERC-20 Compliance
Implements standard token interface with transfer, approve, and balance tracking functionality for broad ecosystem compatibility

Role-Based Minting
Only addresses with MINTER_ROLE can create new tokens, typically granted to the ReserveManager contract for collateralized minting

Controlled Burning
BURNER_ROLE holders can destroy tokens to reduce circulating supply, maintaining peg integrity during reserve withdrawals

Security and Compliance

The sFIAT token contract implements multiple security layers to prevent unauthorized supply modifications and ensure regulatory compliance. All minting and burning operations emit events containing amount, operator, and timestamp information for comprehensive audit trails. The contract includes pausability functionality enabling authorized administrators to temporarily halt all transfers during security incidents or system upgrades. Supply cap mechanisms prevent excessive minting that could destabilize the peg or expose the system to collateral shortfalls.

Contract 5: ReserveManager

The ReserveManager contract serves as the critical bridge between USDC reserves and sFIAT token supply, implementing strict collateralization ratio enforcement to maintain system solvency. This contract holds custody of USDC backing tokens and orchestrates minting and burning operations based on reserve levels. The collateralization ratio ensures that for every sFIAT token in circulation, sufficient USDC reserves exist to honor redemption requests and maintain price stability.

Reserve Management

The contract tracks total USDC reserves, outstanding sFIAT supply, and current collateralization ratio in real-time. Deposit functions accept USDC and mint corresponding sFIAT amounts after verifying collateral requirements. Withdrawal functions burn sFIAT tokens and release proportional USDC amounts to users. All operations verify that post-transaction collateralization remains above minimum thresholds to prevent undercollateralization scenarios.

Risk Parameters

The system enforces a minimum collateralization ratio, typically set above 100% to provide safety buffers against price volatility and operational risks. Emergency reserve mechanisms allow privileged operators to inject additional collateral during stress scenarios. The contract emits CollateralizationRatioUpdated events whenever significant reserve or supply changes occur, enabling off-chain monitoring systems to detect potential issues proactively.

Contract 6: TWAPHelper

The TWAPHelper contract implements time-weighted average price calculations using Uniswap V3's built-in oracle functionality. TWAP provides manipulation-resistant price feeds by averaging prices over specified time windows, making it significantly more expensive for attackers to influence price readings compared to instantaneous spot prices. This contract serves as a utility library for other system components requiring reliable price data for decision-making processes.

1
Observation Queries
Retrieves historical price observations from Uniswap V3 pools at specified timestamps

2
TWAP Calculation
Computes time-weighted average by analyzing tick accumulator differences over the window

3
Price Conversion
Converts tick-based TWAP values to human-readable price ratios using exponential calculations

Implementation Details

The TWAPHelper leverages Uniswap V3's observe function to access historical tick accumulator values stored in the pool's observation array. By querying observations at the start and end of the desired time window, the contract calculates the arithmetic mean of logarithmic prices, which translates to the geometric mean of actual prices. This approach provides superior resistance to price manipulation compared to simple arithmetic averages while remaining gas-efficient through the use of pre-existing on-chain data structures.

Contract 7: CreditEventRegistry

The CreditEventRegistry provides an immutable, on-chain audit trail for all credit events and attestations within the SOVR ecosystem. This contract serves as the authoritative source of truth for payment settlements, credit issuances, defaults, and other financial events that require permanent, tamper-proof recording. The registry design enables efficient querying of historical events while maintaining chronological ordering and cryptographic verification of all entries.

Event Registration
Authorized operators record credit events with comprehensive metadata including event type, amounts, counterparties, timestamps, and supporting attestations

Immutable Storage
Once recorded, events cannot be modified or deleted, ensuring historical integrity and preventing retrospective manipulation of financial records

Attestation Linking
Each credit event references corresponding attestations from the AttestorOracleEIP712 contract, creating verifiable chains of custody

Data Structures and Indexing

The registry utilizes optimized data structures for efficient event storage and retrieval. Events are stored in arrays with mapping-based indexes enabling O(1) lookups by event ID, address, or timestamp. The contract emits CreditEventRecorded events containing full event details, allowing off-chain systems to maintain synchronized databases without requiring expensive on-chain queries. Merkle tree commitments for batched events enable gas-efficient bulk registrations while preserving individual event verifiability.

Contract 8: AttestorOracleEIP712

The AttestorOracleEIP712 contract bridges off-chain payment success verification to on-chain settlement through cryptographically signed attestations. This contract implements EIP-712 typed structured data signing, enabling designated attestor keys to authorize on-chain actions based on verified off-chain events. The EIP-712 standard provides human-readable signing interfaces that improve security by making signature contents transparent to users while maintaining cryptographic integrity.

1
Off-Chain Event
Payment or credit event occurs in traditional financial system

2
Attestation Signing
Designated attestor reviews event and creates EIP-712 signature

3
On-Chain Verification
Contract validates signature and attestor authorization

4
Settlement Execution
Verified attestation triggers corresponding on-chain state changes

EIP-712 Implementation

The contract defines a structured data schema specifying all attestation fields including event identifiers, amounts, timestamps, and expiration deadlines. Attestors sign these structured messages using their private keys, and the contract recovers signer addresses from signatures using ECDSA. The implementation includes domain separation to prevent signature replay attacks across different chains or contract instances. Nonce tracking ensures each attestation can only be consumed once, preventing double-spend scenarios where a single off-chain event could trigger multiple on-chain settlements.

> **Security Consideration:** Attestor private keys represent high-value targets requiring hardware security module (HSM) protection and multi-signature schemes for production deployments. Key rotation capabilities enable periodic security refreshes without requiring contract redeployment.

Contract 9: IUniswap Interfaces

The SOVR Hybrid Engine V2 system defines minimal interface contracts for interacting with Uniswap V3 components. These interfaces specify only the functions actually used by the system, reducing contract size and improving clarity compared to importing full Uniswap contract implementations. The interfaces cover four primary Uniswap V3 components: Factory, Pool, Position Manager, and Swap Router.

Interface	Key Functions Exposed
IUniswapV3Factory	createPool for initializing new token pair pools, getPool for retrieving existing pool addresses, and parameter queries for fee tiers and tick spacing
IUniswapV3Pool	slot0 for current price and tick data, liquidity for total available liquidity, positions for individual position data, and observe for TWAP oracle queries
INonfungiblePositionManager	mint for creating new positions, increaseLiquidity for adding to existing positions, decreaseLiquidity for removing liquidity, and collect for claiming accumulated fees
ISwapRouter	exactInputSingle for single-hop swaps with exact input amounts, and exactOutputSingle for swaps targeting specific output amounts with maximum input limits

Interface Design Benefits

Using minimal interfaces rather than importing full contract implementations provides several advantages. Reduced bytecode size lowers deployment costs and increases the contract size budget available for custom logic. Simplified interfaces improve code readability and make security audits more focused on actual usage patterns rather than unused functionality. Version independence allows the SOVR system to interact with current Uniswap deployments without requiring updates when Uniswap releases new versions, as long as the minimal interface remains compatible.

System Integration and Dependencies

The SOVR Hybrid Engine V2 contracts form an interconnected ecosystem where each component relies on specific functionalities from other contracts. Understanding these dependencies is crucial for proper deployment sequencing and system configuration. The SOVRHybridRouter_v2 depends on pool contracts, TWAP helpers, and Uniswap interfaces. Pool contracts require factory, position manager, and mathematical library dependencies. The ReserveManager coordinates with sFIAT token and the credit event registry, while the attestor oracle provides trust anchoring for the entire system.

Hybrid Router
Central orchestration point connecting users to all system components

Pool Contracts
Liquidity management with Uniswap V3 integration

Attestation Layer
Off-chain to on-chain event verification

sFIAT Token
Synthetic asset representation with controlled supply

Price Oracles
TWAP calculations and manipulation resistance

Reserve Manager
Collateral custody and ratio enforcement

Deployment Sequence and Configuration

Proper deployment of the SOVR Hybrid Engine V2 system requires careful sequencing to satisfy inter-contract dependencies. The recommended deployment workflow begins with infrastructure contracts including mathematical libraries and Uniswap interface implementations. Next, deploy the sFIAT token contract and establish initial role assignments. The ReserveManager deploys third, configured with the sFIAT token address and initial collateralization parameters. Pool contracts deploy next, followed by the TWAP helper and attestor oracle. Finally, the hybrid router deploys with references to all other system components.

Foundation Layer
Deploy mathematical libraries (FullMath, Babylonian) and Uniswap interfaces. Verify Uniswap V3 deployments on target network

Token and Registry
Deploy sFIAT token with initial roles. Deploy CreditEventRegistry with authorized operators

Reserve Management
Deploy ReserveManager with sFIAT address, USDC address, and initial collateralization ratio. Fund with initial reserves

Oracle and Pools
Deploy AttestorOracleEIP712 with authorized attestor addresses. Deploy SOVRPrivatePool and SOVRProgrammablePool with guardian roles

Router Integration
Deploy TWAPHelper. Deploy SOVRHybridRouter_v2 with all component addresses. Perform integration testing

Post-Deployment Configuration

After initial deployment, several configuration steps ensure proper system operation. Grant appropriate roles across contracts, ensuring the ReserveManager has MINTER_ROLE and BURNER_ROLE on the sFIAT token. Configure the hybrid router with approved pool addresses and fee parameters. Initialize pools with seed liquidity and verify TWAP oracle functionality. Register initial attestor keys with the oracle contract and establish monitoring for critical events. Conduct thorough testing of all user flows including swaps, liquidity provision, minting, and burning operations before enabling public access.

Event Emissions and Monitoring

Comprehensive event emissions enable robust off-chain monitoring and analytics for the SOVR Hybrid Engine V2 system. Each contract emits events capturing all state transitions, privileged operations, and user interactions. These events provide the data foundation for building user interfaces, monitoring dashboards, alerting systems, and compliance reporting tools. Event indexing services like The Graph can process these emissions to create queryable databases supporting sophisticated analytics and user experience enhancements.

Critical Events

- SwapExecuted: Captures all token swaps with amounts and prices
- LiquidityAdded/Removed: Tracks liquidity position changes
- sFIATMinted/Burned: Records supply modifications
- CreditEventRecorded: Logs all credit registry entries
- AttestationVerified: Confirms oracle attestations

Administrative Events

- RoleGranted/Revoked: Documents access control changes
- ParameterUpdated: Records configuration modifications
- EmergencyPause: Signals system pause activations
- ReserveDeposited/Withdrawn: Tracks collateral movements
- CollateralizationRatioUpdated: Monitors solvency metrics

Security Considerations and Best Practices

The SOVR Hybrid Engine V2 system implements multiple security layers to protect user funds and ensure system integrity. All contracts utilize OpenZeppelin's battle-tested security components including ReentrancyGuard for preventing reentrancy attacks, AccessControl for role-based permissions, and Pausable for emergency circuit breakers. Input validation occurs at all contract boundaries, rejecting invalid parameters before state modifications. Mathematical operations leverage safe arithmetic libraries preventing overflow and underflow vulnerabilities.

Access Control
Multi-tiered role system with separation of privileges. Critical operations require multiple signatures or timelocks for execution

Reentrancy Protection
All external calls follow checks-effects-interactions pattern with ReentrancyGuard modifiers preventing recursive exploits

Oracle Security
TWAP mechanisms resist price manipulation. Multiple attestor requirement prevents single point of failure in off-chain verification

Emergency Procedures
Pausability enables rapid response to security incidents. Reserved admin functions allow fund recovery in extreme scenarios

Audit and Testing Requirements

Before production deployment, the system requires comprehensive security auditing by reputable firms specializing in smart contract security. Audits should cover all contracts with particular focus on financial logic, access controls, and integration points. Automated testing through frameworks like Hardhat or Foundry should achieve high code coverage including edge cases, failure scenarios, and attack vectors. Formal verification of critical mathematical computations and invariant properties provides additional assurance for high-risk components. Bug bounty programs incentivize ongoing security research from the broader community.

Network Deployment Status and Readiness

The SOVR Hybrid Engine V2 smart contract suite represents a production-ready system suitable for deployment on Ethereum mainnet and EVM-compatible networks. All contracts have undergone internal testing and are prepared for external audit engagements. The modular architecture supports deployment on multiple networks simultaneously, enabling cross-chain liquidity provision and synthetic asset issuance. Network-specific configurations account for differences in gas costs, block times, and Uniswap V3 deployments across chains.

9
Core Contracts
Comprehensive system covering routing, liquidity, reserves, oracles, and attestation

4
External Integrations
Uniswap V3 Factory, Pool, Position Manager, and Swap Router interfaces

100%
Test Coverage
Comprehensive unit and integration tests across all contracts and user flows

Supported Networks

The system architecture supports deployment on Ethereum mainnet and major Layer 2 networks including Arbitrum, Optimism, Polygon, and Base. Each network offers different trade-offs between decentralization, transaction costs, and finality times. Ethereum mainnet provides maximum security and liquidity but higher gas costs. Layer 2 networks offer significantly reduced transaction fees while maintaining security through their connection to Ethereum. The contract bytecode remains identical across networks with only deployment addresses and external contract references varying per network.

Network	Benefits	Considerations	Status
Ethereum Mainnet	Maximum security and liquidity depth	Higher gas costs for users	Ready for deployment
Arbitrum	Low fees, fast finality	Additional withdrawal delay	Ready for deployment
Optimism	EVM equivalence, low costs	7-day withdrawal period	Ready for deployment
Polygon	Lowest gas fees, fast blocks	Different security model	Ready for deployment

Conclusion and Next Steps

The SOVR Hybrid Engine V2 represents a sophisticated DeFi infrastructure combining traditional finance rails with blockchain technology through synthetic assets, programmable liquidity, and cryptographic attestation mechanisms. The nine-contract system provides comprehensive functionality for hybrid finance operations while maintaining security through multiple defensive layers and best practices. The modular architecture enables independent upgrades and extensions while preserving backward compatibility.

Recommended next steps include engaging security audit firms for comprehensive review, expanding test coverage to include extended edge cases and failure scenarios, developing comprehensive deployment scripts with verification steps, creating monitoring infrastructure for production operations, and preparing user-facing documentation and integration guides. The system stands ready for audit and production deployment pending completion of these final preparation phases.