Privacy Hook Architecture

🎯 Overview

UniversalPrivacyHook is a groundbreaking Uniswap V4 hook that enables completely private token swaps using Zama's Fully Homomorphic Encryption (FHE) technology combined with an EigenLayer AVS for batch processing. Users can swap tokens without revealing their swap amounts to anyone - not even validators or MEV bots.

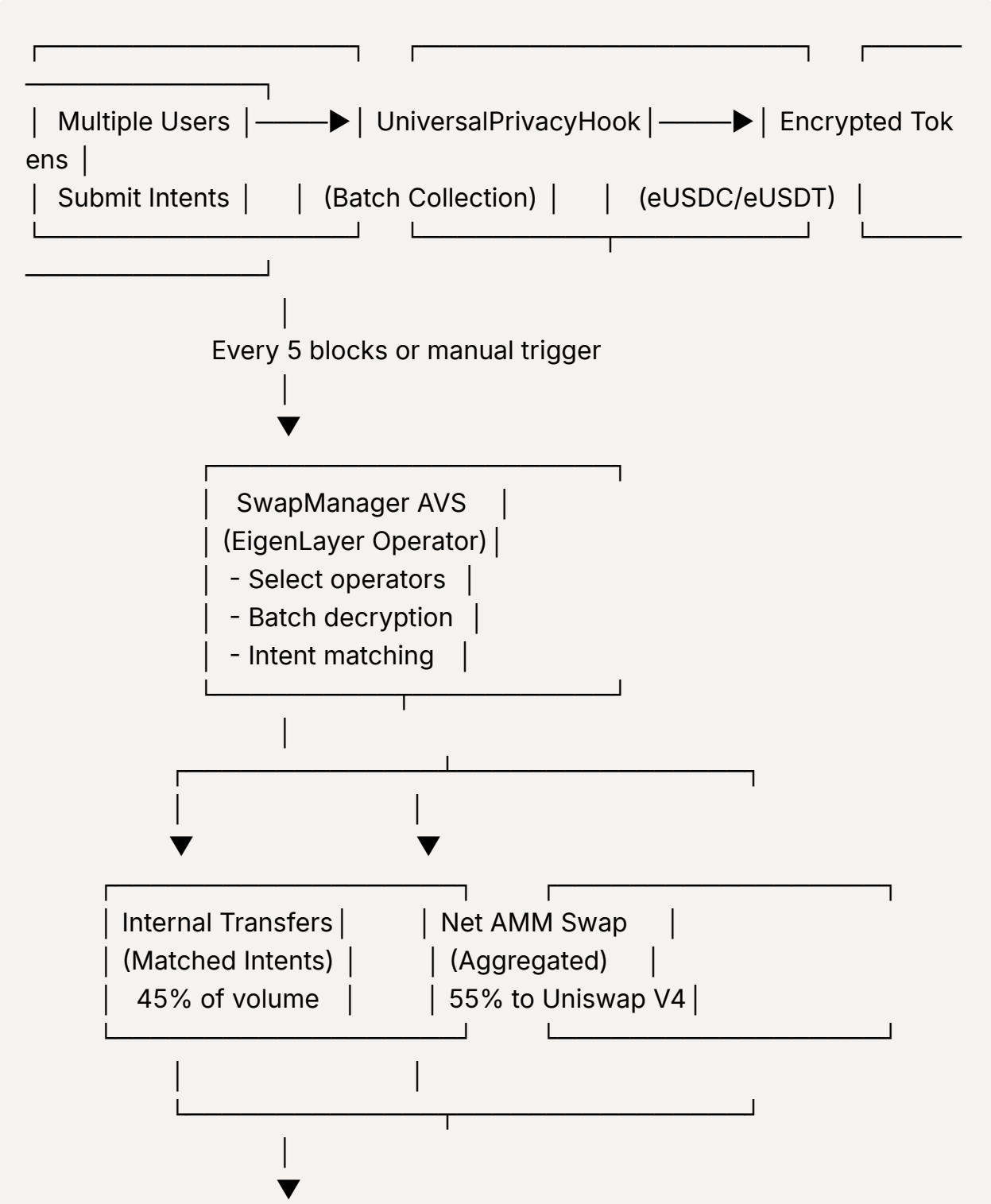
🌟 Key Innovation

This project introduces a novel approach to DeFi privacy with intelligent batching:

- **Complete Privacy Decoupling**: Breaks the link between encrypted intent amounts and on-chain execution - only net AMM amounts are visible on-chain, while matched trades remain fully encrypted end-to-end
- **Encrypted Swap Amounts**: All swap amounts remain encrypted throughout the entire process
- **MEV Protection**: Front-runners cannot see or exploit your trades
- **Batch Settlement with Intent Matching**: AVS operators decrypt and match opposite intents, reducing AMM usage up to 100% (average 45-55% reduction)
- **Save Up to 100% of Impermanent Loss**: Users pay ZERO impermanent loss on matched trades since they never touch the AMM - only the unmatched net amount interacts with the pool
- **Trustless Execution**: Smart contracts process encrypted data with AVS consensus
- **User Sovereignty**: Only users can decrypt their own balances
- **Perfect Execution**: Internal matching provides 1:1 exchange rate with zero slippage and zero impermanent loss

🏗️ Technical Architecture

System Components - Batch Processing Flow



```
┌─────────────────────────────┐
│  settleBatch()              │
│  - Burn/Mint matched        │
│  - Execute net swap         │
│  - Distribute output        │
└─────────────────────────────┘
```

Detailed Sequence Diagram - Complete Batch Flow

```
%%{init: {'theme':'dark', 'themeVariables': { 'primaryColor':'#03dac6', 'primaryTextColor':'#fff', 'primaryBorderColor':'#00a896', 'lineColor':'#03dac6', 'secondaryColor':'#bb86fc', 'tertiaryColor':'#3700b3', 'background':'#121212', 'mainBkg':'#1f1f1f', 'secondBkg':'#2d2d2d', 'tertiaryBkg':'#3d3d3d', 'textColor':'#ffffff', 'labelBackground':'#2d2d2d', 'labelTextColor':'#ffffff', 'actorBkg':'#424242', 'actorBorder':'#03dac6', 'actorTextColor':'#fff', 'signalColor':'#03dac6', 'signalTextColor':'#fff'}}}%%
sequenceDiagram
    participant Users as Multiple Users
    participant Frontend as Frontend<br/>(✅ Working)
    participant Hook as Privacy Hook
    participant ZAMA as ZAMA FHEVM
    participant AVS as AVS Operators<br/>(✅ With Batching)
    participant SwapManager
    participant Pool as Uniswap V4

    rect rgb(30, 60, 80)
        Note over Users,Hook: PHASE 1: Deposit & Token Creation
        Users->>Hook: Deposit USDC/USDT
        Hook->>Hook: Update poolReserves
        Hook->>ZAMA: Create encrypted tokens
        ZAMA->>Hook: Deploy eUSDC/eUSDT contracts
        Hook->>Users: Mint encrypted tokens
    end

    rect rgb(60, 80, 40)
```

```
    Note over Users,Hook: PHASE 2: Intent Collection (5 blocks)
    Users->>ZAMA: Encrypt swap amounts locally
    Users->>Hook: submitIntent(encAmount, tokenIn, tokenOut)
    Hook->>Hook: Transfer eTokens as collateral
    Hook->>Hook: Add to current batch

    alt Batch interval reached (5 blocks)
      Hook->>Hook: Auto-finalize previous batch
      Hook->>SwapManager: Submit batch to AVS
    end
  end

  rect rgb(40, 80, 60)
    Note over AVS,SwapManager: PHASE 3: AVS Processing (Off-chain)
    SwapManager->>AVS: Batch of encrypted intents

    AVS->>ZAMA: Batch decrypt all intents
    Note over AVS: Example batch:<br/>U1: 12k eUSDC→eUSDT<br/>U2: 7.5
k eUSDT→eUSDC<br/>U3: 4k eUSDC→eUSDT<br/>U4: 1.2k eUSDT→eUSDC

    AVS->>AVS: Match opposite intents
    Note over AVS: Internal matching:<br/>U1↔U2: 7.5k<br/>U3↔U4: 1.2k<
br/>Net: 7.3k USDC→USDT

    AVS->>AVS: Calculate net swap needed
  end

  rect rgb(60, 40, 80)
    Note over AVS,Pool: PHASE 4: Settlement
    AVS->>Hook: settleBatch(internalTransfers, netSwap, distributions)

    par Internal Transfers (Encrypted)
      Hook->>ZAMA: burnEncrypted(from, amount)
      Hook->>ZAMA: mintEncrypted(to, amount)
      Note over Hook: 8.7k matched internally<br/>(no AMM needed)
    and Net AMM Swap (Public)
```

```
        Hook->>Pool: unlock() for callback
        Pool->>Hook: unlockCallback()
        Hook->>Pool: swap(7.3k USDC→USDT)
        Pool->>Hook: Return USDT
        Note over Pool: Only 7.3k visible on-chain<br/>(aggregated amount)
    end

    Hook->>Users: Distribute encrypted outputs
    Note over Users: U1: 12k eUSDT<br/>U2: 7.5k eUSDC<br/>U3: 4k eUSDT
<br/>U4: 1.2k eUSDC
  end

  rect rgb(30, 80, 50)
    Note over Users,Pool: IMPROVEMENTS:<br/>✅ Up to 100% reduction in
AMM usage<br/>✅ Complete privacy via batching<br/>✅ Zero impermanen
t loss for matched trades<br/>✅ Working frontend
  end
```

Core Technologies

1. **FHEVM (Fully Homomorphic Encryption Virtual Machine)**
   - Enables computation on encrypted data
   - Maintains privacy throughout transaction lifecycle
   - Integrates seamlessly with EVM
   - Batch decryption via FHE Gateway for AVS operators
2. **EigenLayer AVS (Actively Validated Service)**
   - Decentralized operator network for batch processing
   - Consensus-based settlement verification
   - Intent matching and netting algorithm
   - Reduces on-chain AMM usage by 45-100% (depending on intent matching rate)
   - Matched trades are pure internal transfers - zero impermanent loss

3. **Uniswap V4 Hooks**
    - Custom logic at key pool lifecycle points
    - Enables encrypted token management
    - Handles private swap intents
    - Batch settlement integration
4. **Hybrid Encrypted Tokens**
    - ERC20-compatible tokens with encrypted balances
    - Support both public and private operations
    - Automatic conversion between regular and encrypted tokens
    - Efficient burn/mint for internal matching

✨ Features

For Users

- 🔐 **Complete Privacy**: Swap amounts remain encrypted end-to-end - matched trades NEVER appear on-chain
- 🔒 **Privacy Decoupling**: Your encrypted intent amount is completely decoupled from on-chain execution - only the net aggregated amount hits the chain unencrypted
- 🛡️ **MEV Protection**: Immune to sandwich attacks via batch aggregation
- 💰 **Token Faucet**: Easy testing with mock USDC/USDT
- 📊 **Balance Management**: View and decrypt your encrypted balances
- 🔄 **Intent-Based Swaps**: Submit swap intents that execute asynchronously
- 💎 **Save Up to 100% of Impermanent Loss**: Pay ZERO impermanent loss on matched trades - only unmatched amounts touch the AMM
- 🎯 **Perfect Pricing**: Internal matching provides 1:1 exchange rate with zero slippage and zero IL

- ⚡ **AMM Usage Reduction**: Up to 100% reduction when trades are perfectly matched

For Developers

- 📝 **Extensive FHEVM Integration**: Full implementation of FHE operations
- 🎣 **Custom Hook Implementation**: Complete Uniswap V4 hook with batch settlement
- 🌐 **Frontend SDK Integration**: Uses `@zama-fhe/relayer-sdk` for client-side encryption
- 🧪 **Comprehensive Testing**: Full test suite with hardhat tasks
- 🔗 **AVS Integration**: EigenLayer operator network for decentralized batch processing
- 🧱 **Intent Matching Algorithm**: Off-chain netting reduces AMM dependency

```
cd packages/fhevm-hardhat-template

# Deploy all contracts to Sepolia
npx hardhat deploy --network sepolia

# Or use individual tasks
npx hardhat task:deployHook --network sepolia
npx hardhat task:initializePool --network sepolia
npx hardhat task:test-deposit --amount 100 --network sepolia
npx hardhat task:test-intent --amount 10 --network sepolia
```

📜 Smart Contracts

Core Contracts

UniversalPrivacyHook.sol (`packages/fhevm-hardhat-template/contracts/UniversalPrivacyHook.sol`)

The main hook contract implementing Uniswap V4 hook interface with FHE capabilities.

Key Features:

- `deposit()`: Converts regular tokens to encrypted tokens
- `submitIntent()`: Creates encrypted swap intent
- `executeIntent()`: Processes decrypted swap
- `beforeSwap()`: Validates encrypted swap amounts
- `afterSwap()`: Updates encrypted balances

FHEVM Integration:

```
struct Intent {
    euint128 encAmount;      // Encrypted amount to swap
    Currency tokenIn;        // Input currency
    Currency tokenOut;       // Output currency
    address owner;           // Intent owner
    uint64 deadline;         // Expiration timestamp
    bool processed;          // Execution status
    bool decrypted;          // Decryption status
    uint128 decryptedAmount;// Amount after decryption
    PoolKey poolKey;         // Pool configuration
}
```

HybridFHERC20.sol (`packages/fhevm-hardhat-template/contracts/HybridFHERC20.sol`)

Encrypted ERC20 token implementation supporting both public and private operations.

Innovations:

- Dual balance system (public + encrypted)
- FHE arithmetic operations
- User-controlled decryption permissions

- Seamless conversion between modes

FHEVM Operations:

```
// Encrypted balance storage
mapping(address ⇒ euint128) public encBalances;

// FHE operations
function _transferEncrypted(address from, address to, euint128 amount) inter
nal {
    euint128 fromBalance = encBalances[from];
    ebool canTransfer = FHE.gte(fromBalance, amount);

    encBalances[from] = FHE.select(
        canTransfer,
        FHE.sub(fromBalance, amount),
        fromBalance
    );

    encBalances[to] = FHE.select(
        canTransfer,
        FHE.add(encBalances[to], amount),
        encBalances[to]
    );
}
```

Deployed Addresses (Sepolia)

Contract	Address	Explorer
UniversalPrivacyHook	0x32841c9E0245C4B1a9cc29137d7E1F078e6f0080	View
SwapManager (AVS)	0xFbce8804FfC5413d60093702664ABfd71Ce0E592	View
PoolManager	0xE03A1074c86CFeDd5C142C4F04F1a1536e203543	View
MockUSDC	0x59dd1A3Bd1256503cdc023bfC9f10e107d64C3C1	View
MockUSDT	0xB1D9519e953B8513a4754f9B33d37eDba90c001D	View
EncryptedUSDC	0x161270dC19388dAd88E80Ee4Fec9e380946Af78d	View

Contract	Address	Explorer
EncryptedUSDT	`0x642C2074a7067675d559dbB899258a8366c07b9B`	View

🔧 Technical Deep Dive

Batch Processing & Intent Matching

The core innovation of UniversalPrivacyHook is its **batch settlement architecture**:

1. **Intent Collection Phase (5 blocks)**
   - Users submit encrypted swap intents to the hook
   - Hook collects intents into batches every 5 blocks
   - Each intent includes encrypted amount and token direction
   - Hook holds encrypted tokens as collateral
2. **AVS Processing Phase**
   - Batch is finalized and sent to SwapManager AVS
   - Selected EigenLayer operators receive FHE decryption permissions
   - Operators decrypt all intents in the batch
   - **Intent Matching Algorithm** runs off-chain:

```
Example batch:
User A: 12,000 USDC → USDT
User B:  7,500 USDT → USDC
User C:  4,000 USDC → USDT
User D:  1,200 USDT → USDC

Matching:
A ↔ B: 7,500 (internal transfer)
C ↔ D: 1,200 (internal transfer)

Result:
```

```
- 8,700 matched internally (54% of volume)
- 7,300 USDC needs AMM swap to USDT
```

3. **Settlement Phase**
   - AVS submits settlement to hook with operator signatures
   - **Internal transfers**: Burn/mint encrypted tokens (matched intents)
   - **Net AMM swap**: Execute aggregated swap on Uniswap V4
   - **Output distribution**: Allocate swap output proportionally to users
   - All output amounts are encrypted and minted to users

```
uint256 requestId = FHE.requestDecryption(
    cts,                        // Ciphertext array
    this.finalizeIntent.selector  // Callback function selector
);
```

AVS Architecture

The **SwapManager** contract (`packages/hello-world-avs/contracts/src/SwapManager.sol`) is an EigenLayer AVS that:

1. **Operator Management**
   - Operators register with the AVS to process batches
   - Committee selection is deterministic based on batch ID
   - Minimum attestation threshold for consensus (currently 1 for testing)
2. **Batch Processing**
   - Receives finalized batches from UniversalPrivacyHook
   - Grants FHE decryption permissions to selected operators
   - Operators decrypt intents off-chain and compute optimal matching
   - Multiple operators sign the settlement for consensus
3. **Settlement Submission**

```
function submitBatchSettlement(
    bytes32 batchId,
    InternalTransfer[] calldata internalTransfers,  // Matched intents
    uint128 netAmountIn,                            // Net to AMM
    address tokenIn,
    address tokenOut,
    address outputToken,
    UserShare[] calldata userShares,                // Output distribution
    bytes calldata inputProof,
    bytes[] calldata operatorSignatures             // Consensus proof
) external onlyOperator
```