Report

v. 0.9

Customer

Uniswap

Smart Contract Audit

Uniswap. v4-core

5th September 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	05.09.24	A. Zveryanskaya	Initial Draft
0.2	05.09.24	A. Zveryanskaya	Minor revision
0.9	05.09.24	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

Uniswap V4, the latest iteration of the Uniswap protocol, is a significant advancement in decentralized exchanges (DEXs) and automated market makers (AMMs).

3 Project scope

We were asked to review:

- Original Code

Files:

/			
	ERC6909.sol	ERC6909Claims.sol	Extsload.sol
	Exttload.sol	NoDelegateCall.sol	PoolManager.sol
	ProtocolFees.sol		
interfaces/callback/			
	IUnlockCallback.sol		
interfaces/external/			
	IERC20Minimal.sol	IERC6909Claims.sol	
interfaces/			
	IExtsload.sol	IExttload.sol	IHooks.sol
	IPoolManager.sol	IProtocolFeeCon-troller.sol	IProtocolFees.sol
libraries/			
	BitMath.sol	CurrencyDelta.sol	CustomRevert.sol
	UnsafeMath.sol	FixedPoint96.sol	FixedPoint128.sol
	FullMath.sol	Hooks.sol	LiquidityMath.sol
	Lock.sol	LPFeeLibrary.sol	NonZeroDeltaCount.sol
	ParseBytes.sol	Pool.sol	Position.sol
	ProtocolFeeLibrary.sol	Reserves.sol	SafeCast.sol
	SqrtPriceMath.sol	StateLibrary.sol	SwapMath.sol
	TickBitmap.sol	TickMath.sol	TransientStateLibrary.sol

types/			
	BalanceDelta.sol	BeforeSwapDelta.sol	Currency.sol
	PoolId.sol	PoolKey.sol	Slot0.sol

All found issues were left as is.
After fixing the indicated issues, the smart contracts should be re-audited.

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Recommendations** contain code style, best practices and other suggestions.

5 Our findings

We found 17 major, and a few less important issues.

6 Major Issues

CVF-1 Reported

- **Category** Flaw
- **Source** TickBitmap.sol

Description Here local variables of a type narrower than 256 bits are accessed as it they were 256 bit wide. Solidity documentation warns that: "if you access variables of a type that spans less than 256 bits (for example uint64, address, or bytes16), you cannot make any assumptions about bits not part of the encoding of the type" (https://docs.soliditylang.org/en/v0.8.26/assembly.html#access-to-external-variables-functions-and-libraries).

Recommendation Properly clean extra bits between using narrow variables as recommended in the documentation: "to be safe, always clear the data properly before you use it in a context where this is important".

```
22        sdiv(tick, tickSpacing),
```

```
24        slt(smod(tick, tickSpacing), 0)
```

```
36 wordPos := sar(8, tick)
   bitPos := and(tick, 0xff)
```

```
53 if smod(tick, tickSpacing) {
```

```
55     mstore(0x20, tick)
       mstore(0x40, tickSpacing)
```

```
59 tick := sdiv(tick, tickSpacing)
```

```
62 mstore(0, sar(8, tick))
```

```
68 sstore(slot, xor(sload(slot), shl(and(tick, 0xff), 1)))
```

CVF-2 Reported

- **Category** Flaw
- **Source** Position.sol

Description Here local variables of a type narrower than 256 bits are accessed as it they were 256 bit wide. Solidity documentation warns that: "if you access variables of a type that spans less than 256 bits (for example uint64, address, or bytes16), you cannot make any assumptions about bits not part of the encoding of the type" (https://docs.soliditylang.org/en/v0.8.26/assembly.html#access-to-external-variables-functions-and-libraries).

Recommendation Properly clean extra bits between using narrow variables as recommended in the documentation: "to be safe, always clear the data properly before you use it in a context where this is important".

```
44 mstore(0x06, tickUpper) // [0x23, 0x26)
   mstore(0x03, tickLower) // [0x20, 0x23)
   mstore(0, owner) // [0x0c, 0x20)
```

CVF-3 Reported

- **Category** Flaw
- **Source** TickMath.sol

Description Here local variables of a type narrower than 256 bits are accessed as it they were 256 bit wide. Solidity documentation warns that: "if you access variables of a type that spans less than 256 bits (for example uint64, address, or bytes16), you cannot make any assumptions about bits not part of the encoding of the type" (https://docs.soliditylang.org/en/v0.8.26/assembly.html#access-to-external-variables-functions-and-libraries).

Recommendation Properly clean extra bits between using narrow variables as recommended in the documentation: "to be safe, always clear the data properly before you use it in a context where this is important".

```
55 let mask := sar(255, tick)
```

```
59 absTick := xor(mask, add(mask, tick))
```

CVF-4 Reported

- **Category** Flaw
- **Source** SqrtPriceMath.sol

Description Here local variables of a type narrower than 256 bits are accessed as it they were 256 bit wide. Solidity documentation warns that: “if you access variables of a type that spans less than 256 bits (for example uint64, address, or bytes16), you cannot make any assumptions about bits not part of the encoding of the type” (https://docs.soliditylang.org/en/v0.8.26/assembly.html#access-to-external-variables-functions-and-libraries).

Recommendation Properly clean extra bits between using narrow variables as recommended in the documentation: “to be safe, always clear the data properly before you use it in a context where this is important”.

```
105     if iszero(gt(sqrtPX96, quotient)) {
```

```
131 if or(iszero(sqrtPX96), iszero(liquidity)) {
```

```
157 if or(iszero(sqrtPX96), iszero(liquidity)) {
```

```
187     if iszero(sqrtPriceAX96) {
```

```
205 let diff := sub(a, b)
```

CVF-5 Reported

- **Category** Flaw
- **Source** SwapMath.sol

Description Here local variables of a type narrower than 256 bits are accessed as it they were 256 bit wide. Solidity documentation warns that: "if you access variables of a type that spans less than 256 bits (for example uint64, address, or bytes16), you cannot make any assumptions about bits not part of the encoding of the type" (https://docs.soliditylang.org/en/v0.8.26/assembly.html#access-to-external-variables-functions-and-libraries).

Recommendation Properly clean extra bits between using narrow variables as recommended in the documentation: "to be safe, always clear the data properly before you use it in a context where this is important".

```
29 let nextOrLimit := xor(lt(sqrtPriceNextX96, sqrtPriceLimitX96),
      ↪ zeroForOne)
30 let symDiff := xor(sqrtPriceNextX96, sqrtPriceLimitX96)
   sqrtPriceTargetX96 := xor(sqrtPriceLimitX96, mul(symDiff,
      ↪ nextOrLimit))
```

CVF-6 Reported

- **Category** Procedural
- **Source** SwapMath.sol

Description Underflow shouldn't be possible here, but the logic that is supposed to guarantee this is in other files. Relying on it introduces hidden relationship between distant code parts.

Recommendation Use safe subtraction.

```
78 feeAmount = uint256(-amountRemaining) - amountIn;
```

CVF-7 Reported

- **Category** Flaw
- **Source** ProtocolFeeLibrary.sol

Description Here local variables of a type narrower than 256 bits are accessed as it they were 256 bit wide. Solidity documentation warns that: "if you access variables of a type that spans less than 256 bits (for example uint64, address, or bytes16), you cannot make any assumptions about bits not part of the encoding of the type" (https://docs.soliditylang.org/en/v0.8.26/assembly.html#access-to-external-variables-functions-and-libraries).

Recommendation Properly clean extra bits between using narrow variables as recommended in the documentation: "to be safe, always clear the data properly before you use it in a context where this is important".

27
```
let isOneForZeroFeeOk := lt(self, FEE_1_THRESHOLD)
```

38
```
let numerator := mul(self, lpFee)
```

40
```
swapFee := sub(add(self, lpFee), divRoundingUp)
```

CVF-8 Reported

- **Category** Overflow/Underflow
- **Source** Pool.sol

Description Phantom overflow is possible here.

Recommendation Use the "mulDiv" function.

384
```
uint256 delta = (step.amountIn + step.feeAmount) * protocolFee /
    ↪ ProtocolFeeLibrary.PIPS_DENOMINATOR;
```

CVF-9 Reported

- **Category** Flaw
- **Source** Pool.sol

Description Here local variables of a type narrower than 256 bits are accessed as it they were 256 bit wide. Solidity documentation warns that: "if you access variables of a type that spans less than 256 bits (for example uint64, address, or bytes16), you cannot make any assumptions about bits not part of the encoding of the type" (https://docs.soliditylang.org/en/v0.8.26/assembly.html#access-to-external-variables-functions-and-libraries).

Recommendation Properly clean extra bits between using narrow variables as recommended in the documentation: "to be safe, always clear the data properly before you use it in a context where this is important".

```
552 liquidityGrossAfter,
```

```
554 shl(128, liquidityNet)
```

CVF-10 Reported

- **Category** Overflow/Underflow
- **Source** NonZeroDeltaCount.sol

Description Relying on business-level constraints for preventing low-level problems, such as underflow is a bid practice, as it introduces hidden relationships in the code.

Recommendation Add proper underflow check.

```
28 /// Current usage ensures this will not happen because we call
   ↪ decrement with known boundaries (only up to the number of
   ↪ times we call increment).
```

CVF-11 Reported

- **Category** Unclear behavior
- **Source** Reserves.sol

Description This doesn't allow distinguishing value == 0 and value == ZERO_BALANCE.

Recommendation Explicitly forbid passing "ZERO_BALANCE" as "value".

```
19 if (value == 0) value = ZERO_BALANCE;
```

CVF-12 Reported

- **Category** Flaw
- **Source** TransientStateLibrary.sol

Description Here local variables of a type narrower than 256 bits are accessed as it they were 256 bit wide. Solidity documentation warns that: “if you access variables of a type that spans less than 256 bits (for example uint64, address, or bytes16), you cannot make any assumptions about bits not part of the encoding of the type” (https://docs.soliditylang.org/en/v0.8.26/assembly.html#access-to-external-variables-functions-and-libraries).

Recommendation Properly clean extra bits between using narrow variables as recommended in the documentation: “to be safe, always clear the data properly before you use it in a context where this is important”.

30
```
mstore(32, currency)
```

47
```
mstore(0, caller_)
mstore(32, currency)
```

CVF-13 Reported

- **Category** Flaw
- **Source** CurrencyDelta.sol

Description Here local variables of a type narrower than 256 bits are accessed as it they were 256 bit wide. Solidity documentation warns that: “if you access variables of a type that spans less than 256 bits (for example uint64, address, or bytes16), you cannot make any assumptions about bits not part of the encoding of the type” (https://docs.soliditylang.org/en/v0.8.26/assembly.html#access-to-external-variables-functions-and-libraries).

Recommendation Properly clean extra bits between using narrow variables as recommended in the documentation: “to be safe, always clear the data properly before you use it in a context where this is important”.

12
```
mstore(0, caller_)
mstore(32, currency)
```

CVF-14 Reported

- **Category** Flaw
- **Source** LiquidityMath.sol

Description Here local variables of a type narrower than 256 bits are accessed as it they were 256 bit wide. Solidity documentation warns that: "if you access variables of a type that spans less than 256 bits (for example uint64, address, or bytes16), you cannot make any assumptions about bits not part of the encoding of the type" (https://docs.soliditylang.org/en/v0.8.26/assembly.html#access-to-external-variables-functions-and-libraries).

Recommendation Properly clean extra bits between using narrow variables as recommended in the documentation: "to be safe, always clear the data properly before you use it in a context where this is important".

```
12 z := add(x, y)
```

CVF-15 Reported

- **Category** Flaw
- **Source** Extsload.sol

Description This assumes that the copied ABI offset is 0x20 which is not actually guaranteed.

Recommendation Explicitly store the 0x20 value at the memory offset zero.

```
42 // Copy the abi offset of dynamic array and the length of the array
   ↪ to memory.
calldatacopy(0, 0x04, 0x40)
```

CVF-16 Reported

- **Category** Flaw
- **Source** Exttload.sol

Description This assumes that the copied ABI offset is 0x20 which is not actually guaranteed.

Recommendation Explicitly store the 0x20 value at the memory offset zero.

```
23 // Copy the abi offset of dynamic array and the length of the array
   ↪ to memory.
calldatacopy(0, 0x04, 0x40)
```

CVF-17 Reported

- **Category** Unclear behavior
- **Source** ProtocolFees.sol

Description As “data” is allowed to be shorter than 32 bytes, this could read after the “data” contents.

Recommendation Require “data.length“ to be exactly 32 bytes.

```
80 returnData := mload(add(_data, 0x20))
```

7 Moderate Issues

CVF-18 Reported

- **Category** Procedural
- **Source** Currency.sol

Description Any data returned by the failed call is lost here.

Recommendation Included the data, returned by the failed call, into the error.

```
46 mstore(0x00, 0xf4b3b1bc) // `NativeTransferFailed()`.
   revert(0x1c, 0x04)
```

CVF-19 Reported

- **Category** Procedural
- **Source** Currency.sol

Description Any data returned by the failed call is lost here.

Recommendation Included the data, returned by the failed call, into the error.

```
64 mstore(0x00, 0xf27f64e4) // `ERC20TransferFailed()`.
   revert(0x1c, 0x04)
```

CVF-20 Reported

- **Category** Documentation
- **Source** Currency.sol

Description The comment is inaccurate, as this line doesn't restore, but rather set to zero the overwritten part of the free memory pointer. Such technic looks like a dirty hack.

Recommendation Use memory referred by the free memory pointer rather than the first two memory slots. This would make memory access a bit more expensive, but will make restoring the free memory pointer unnecessary.

```
67 mstore(0x34, 0) // Restore the part of the free memory pointer that
      ↪ was overwritten.
```

CVF-21 Reported

- **Category** Unclear behavior
- **Source** Slot0.sol

Description The maximum fee value looks arbitrary.

Recommendation Implement support for the full fees range, i.e. up to 100%.

```
20 * the maximum is 1000 - meaning the maximum protocol fee is 0.1%
```

CVF-22 Reported

- **Category** Suboptimal
- **Source** Slot0.sol

Recommendation The bitwise “and“ is redundant, as Solidity permits junk in unused bits of narrow types.

```
42 _sqrtPriceX96 := and(MASK_160_BITS, _packed)
```

```
54 _protocolFee := and(MASK_24_BITS, shr(PROTOCOL_FEE_OFFSET, _packed))
```

```
60 _lpFee := and(MASK_24_BITS, shr(LP_FEE_OFFSET, _packed))
```

CVF-23 Reported

- **Category** Suboptimal
- **Source** Slot0.sol

Recommendation The “signextend” opcode is redundant, as Solidity permits junk in unused bits of narrow types.

```
48 _tick := signextend(2, shr(TICK_OFFSET, _packed))
```

CVF-24 Reported

- **Category** Suboptimal
- **Source** SafeCast.sol

Recommendation This could be simplified as: function toUint160(uint256 x) internal pure returns (uint160 y) { if ((y = uint160(x)) != x) _revertOverflow(); }

```
21 function toUint160(uint256 x) internal pure returns (uint160) {
       if (x >= 1 << 160) _revertOverflow();
       return uint160(x);
   }
```

CVF-25 Reported

- **Category** Suboptimal
- **Source** SafeCast.sol

Recommendation This could be simplified as: function toUint128(uint256 x) internal pure returns (uint128 y) { if ((y = uint128(x)) != x) _revertOverflow(); }

```
29 function toUint128(uint256 x) internal pure returns (uint128) {
30     if (x >= 1 << 128) _revertOverflow();
       return uint128(x);
   }
```

CVF-26 Reported

- **Category** Suboptimal
- **Source** SafeCast.sol

Recommendation This could be simplified as: function toInt128(int256 x) internal pure returns (int128 y) { if ((y = int128(x)) != x) _revertOverflow(); }

```
37 function toInt128(int256 x) internal pure returns (int128) {
       unchecked {
           if (((1 << 127) + uint256(x)) >> 128 == uint256(0)) return
              ↪ int128(x);
40         _revertOverflow();
       }
   }
```

CVF-27 Reported

- **Category** Suboptimal
- **Source** SafeCast.sol

Recommendation This could be simplified as: function toInt256(uint256 x) internal pure returns (int256 y) { if ((y = int256(x)) < 0) _revertOverflow(); }

```
47 function toInt256(uint256 x) internal pure returns (int256) {
       if (int256(x) >= 0) return int256(x);
       _revertOverflow();
50 }
```

CVF-28 Reported

- **Category** Suboptimal
- **Source** SafeCast.sol

Recommendation This could be simplified as: function toInt128(uint256 x) internal pure returns (int128 y) { y = int128(int256(x)); if (y < 0 || uint128(y) != x) _revertOverflow(); }

```
55 function toInt128(uint256 x) internal pure returns (int128) {
       if (x >= 1 << 127) _revertOverflow();
       return int128(int256(x));
   }
```

CVF-29 Reported

- **Category** Suboptimal
- **Source** BitMath.sol

Recommendation This function could be significantly optimized. See this implementation: https://github.com/Vectorized/solady/blob/main/src/utils/LibBit.sol#L12-L28

```
13 function mostSignificantBit(uint256 x) internal pure returns (uint8
   ↪ r) {
```

CVF-30 Reported

- **Category** Suboptimal
- **Source** BitMath.sol

Recommendation This function could be significantly optimized. See this implementation: https://github.com/Vectorized/solady/blob/main/src/utils/LibBit.sol#L47-L69

```
55 function leastSignificantBit(uint256 x) internal pure returns (uint8
   ↪  r) {
```

CVF-31 Reported

- **Category** Suboptimal
- **Source** FullMath.sol

Description While this function is very efficient in a general case, for specific cases better approaches do exist. For example, when “a”, “b”, or “denominator“ is a known power of 2, multiplication or division could be replace with shift. When “denominator” is known at the compile time, its modular inverse and shift should be precomputer. Also, if “denominator” fits into 128 bits, simple math tricks could be used: https://medium.com/coinmonks/math-in-solidity-part-3-percents-and-proportions-4db014e080b1#4821

Recommendation Implement efficient versions of this function for specific cases.

```
14 function mulDiv(uint256 a, uint256 b, uint256 denominator) internal
   ↪ pure returns (uint256 result) {
```

CVF-32 Reported

- **Category** Suboptimal
- **Source** TickBitmap.sol

Recommendation This function could be simplified by adding (tickSpacing « 24) to the tick value before the division, and subtracting 2^24 from the division result. This would make the division effectively unsigned, so no need to subtract one from the result in case of a negative tick.

```
16 function compress(int24 tick, int24 tickSpacing) internal pure
   ↪ returns (int24 compressed) {
```

CVF-33 Reported

- **Category** Unclear behavior
- **Source** TickBitmap.sol

Description This function permits negative tick spacings, which is weird.

Recommendation Make the “tickSpacing” argument unsigned or explicitly forbid negative “tickSpacing” values.

```
16 function compress(int24 tick, int24 tickSpacing) internal pure
     ↪ returns (int24 compressed) {
```

CVF-34 Reported

- **Category** Unclear behavior
- **Source** TickBitmap.sol

Description This works correctly only when “tick” is a factor of “tickSpacing“.

Recommendation Explicitly forbid “tick” values that are not factors of “tickSpacing” or do proper compression here.

```
59 tick := sdiv(tick, tickSpacing)
```

CVF-35 Reported

- **Category** Flaw
- **Source** CustomRevert.sol

Description Here local variables of a type narrower than 256 bits are accessed as it they were 256 bit wide. Solidity documentation warns that: "if you access variables of a type that spans less than 256 bits (for example uint64, address, or bytes16), you cannot make any assumptions about bits not part of the encoding of the type" (https://docs.soliditylang.org/en/v0.8.26/assembly.html#access-to-external-variables-functions-and-libraries).

Recommendation Properly clean extra bits between using narrow variables as recommended in the documentation: "to be safe, always clear the data properly before you use it in a context where this is important".

```
40 mstore(0x04, value)
```

```
49 mstore(0x04, value1)
50 mstore(0x24, value2)
```

```
59 mstore(0x04, value1)
60 mstore(0x24, value2)
```

CVF-36 Reported

- **Category** Procedural
- **Source** Position.sol

Description While the "mulDiv" function is very efficient in a generic case, for specific cases better approaches do exist.

Recommendation Implement and use here the "mulShr" function similar to "mulDiv" but doing right shift instead of division.

```
78 FullMath.mulDiv(feeGrowthInside0X128 - self.feeGrowthInside0LastX128
   ↪ , liquidity, FixedPoint128.Q128);
```

```
80 FullMath.mulDiv(feeGrowthInside1X128 - self.feeGrowthInside1LastX128
   ↪ , liquidity, FixedPoint128.Q128);
```

CVF-37 Reported

- **Category** Unclear behavior
- **Source** TickMath.sol

Description This assumes MIN_TICK = -MAX_TICK.

Recommendation Do not rely on that assumption and use both, the “MIN_TICK” and the “MAX_TICK” constants.

```
63 if gt(absTick, MAX_TICK) {
```

CVF-38 Reported

- **Category** Suboptimal
- **Source** TickMath.sol

Description It could make sense to perform a few more iterations estimating the logarithm, in order to make the “getSqrtPriceAtTick” call more rare.

Recommendation Perform experiments to empirically find the optimal number of iterations.

```
269 tick = tickLow == tickHi ? tickLow : getSqrtPriceAtTick(tickHi) <=
    ↪ sqrtPriceX96 ? tickHi : tickLow;
```

CVF-39 Reported

- **Category** Unclear behavior
- **Source** SwapMath.sol

Description There is no range check for this argument.

Recommendation Add a check to ensure feePips <= MAX_FEE_PIPS.

```
51 uint24 feePips
```

CVF-40 Reported

- **Category** Unclear behavior
- **Source** ProtocolFeeLibrary.sol

Description It seems that here “self” is not a tightly packed pair of fees, but rather just a single fee. In other placed, the “uint16” is used for a single fee, while “uint24” is used for a packed pair of fees.

Recommendation Use the “uint16” type for “self” here.

```
38 let numerator := mul(self, lpFee)
```

CVF-41 Reported

- **Category** Unclear behavior
- **Source** Pool.sol

Description There are no range checks for the arguments.

Recommendation Add proper range checks.

```
100 function initialize(State storage self, uint160 sqrtPriceX96, uint24
    ↪  protocolFee, uint24 lpFee)
```

```
112 function setProtocolFee(State storage self, uint24 protocolFee)
    ↪ internal {
```

```
118 function setLPFee(State storage self, uint24 lpFee) internal {
```

CVF-42 Reported

- **Category** Documentation
- **Source** Pool.sol

Description Despite the comment, this code doesn’t actually add fees somewhere, but rather calculates the amounts that ought to be added.

Recommendation Rephrase the comment.

```
194 // Fees earned from LPing are added to the user's currency delta.
feeDelta = toBalanceDelta(feesOwed0.toInt128(), feesOwed1.toInt128()
    ↪ );
```

CVF-43 Reported

- **Category** Suboptimal
- **Source** Pool.sol

Description This logic seems redundant, as the initial fee growth values for a tick doesn't matter.

Recommendation Remove this logic and leave zero initial values.

```
535 // by convention, we assume that all growth before a tick was
       ↪ initialized happened _below_ the tick
    if (tick <= self.slot0.tick()) {
        Reported.feeGrowthOutside0X128 = self.feeGrowthGlobal0X128;
        Reported.feeGrowthOutside1X128 = self.feeGrowthGlobal1X128;
    }
```

CVF-44 Reported

- **Category** Unclear behavior
- **Source** ParseBytes.sol

Description There is no length check for the "result" array, so data after the array could be read here.

Recommendation Add proper length check.

```
12 selector := mload(add(result, 0x20))
```

```
19 lpFee := mload(add(result, 0x60))
```

```
26 hookReturn := mload(add(result, 0x40))
```

CVF-45 Reported

- **Category** Suboptimal
- **Source** Hooks.sol

Description This way of error reporting doesn't allow distinguishing the following two situations: i) a call failed with no revert reason, and ii) a call failed with "FailedHookCall" error.

Recommendation Consider either always bubbling up the unchanged revert reason returned by a call, or (preferred) to always wrap the revert reason, returned by a call, into a named error.

132
```
if iszero(returndatasize()) {
    // if the call failed without a revert reason, revert with `
       ↪ FailedHookCall()`
    mstore(0, 0x36bc48c5)
    revert(0x1c, 0x04)
}
// bubble up revert
returndatacopy(0, 0, returndatasize())
revert(0, returndatasize())
```

CVF-46 Reported

- **Category** Suboptimal
- **Source** StateLibrary.sol

Description This zeros high bits of the free memory counter regardless of what was the original value of these bits.

Recommendation Properly restore the original bits.

262
```
mstore(0x26, 0) // rewrite 0x26 to 0
```

CVF-47 Reported

- **Category** Bad naming
- **Source** IProtocolFees.sol

Description The semantics of the arguments and returned values is unclear.

Recommendation Give descriptive names to the unnamed arguments and returned values and/or explain then in documentation comments.

23
```
function protocolFeesAccrued(Currency) external view returns (
    ↪ uint256);
```

26
```
function setProtocolFee(PoolKey memory key, uint24) external;
```

32
```
function collectProtocolFees(address, Currency, uint256) external
    ↪ returns (uint256);
```

CVF-48 Reported

- **Category** Procedural
- **Source** ERC6909.sol

Description Here a low-level underflow check is used to enforce a business-level constraint, which is a bad practice, as it makes code more error-prone and harder to read.

Recommendation Implement explicit balance and allowance checks.

36
```
balanceOf[msg.sender][id] -= amount;
```

48
```
    if (allowed != type(uint256).max) allowance[sender][msg.sender][
        ↪ id] = allowed - amount;
```

51
```
balanceOf[sender][id] -= amount;
```

96
```
balanceOf[sender][id] -= amount;
```

CVF-49 Reported

- **Category** Procedural
- **Source** ERC6909Claims.sol

Description Here a low-level underflow check is used to enforce a business-level constraint, which is a bad practice, as it makes code more error-prone and harder to read.

Recommendation Implement an explicit allowance check.

```
18 allowance[from][sender][id] = senderAllowance - amount;
```

CVF-50 Reported

- **Category** Procedural
- **Source** ProtocolFees.sol

Description Here a low-level underflow check is used to enforce a business-level constraint, which is a bad practice, as it makes code more error-prone and harder to read.

Recommendation Implement an explicit balance check.

```
53 protocolFeesAccrued[currency] -= amountCollected;
```

8 Minor Issues

CVF-51 Reported

- **Category** Procedural
- **Source** Currency.sol

Description This version requirement looks arbitrary.

Recommendation Specify as “^0.8.0” unless there is something special regarding this particular version. Also relevant for: PoolKey.sol, BalanceDelta.sol, Slot0.sol, PoolId.sol, BeforeSwapDelta.sol, SafeCast.sol, BitMath.sol, FullMath.sol, FixedPoint128.sol,TickBitmap.sol, Position.sol,TickMath.sol, UnsafeMath.sol, FixedPoint96.sol, SqrtPriceMath.sol, SwapMath.sol, ProtocolFeeLibrary.sol, LPFeeLibrary.sol, Pool.sol, ParseBytes.sol, Hooks.sol, Lock.sol, NonZeroDeltaCount.sol, Reserves.sol, StateLibrary.sol, TransientStateLibrary.sol, CurrencyDelta.sol, LiquidityMath.sol, IERC20Minimal.sol, IUnlockCallback.sol, IExttload.sol, IHooks.sol, IProtocolFeeController.sol, IProtocolFees.sol, IPoolManager.sol, Exttload.sol, NoDelegateCall.sol, ProtocolFees.sol, PoolManager.sol.

```
2 pragma solidity ^0.8.20;
```

CVF-52 Reported

- **Category** Procedural
- **Source** Currency.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-53 Reported

- **Category** Procedural
- **Source** Currency.sol

Recommendation This library should be moved into a separate file named “CurrencyLibrary.sol”.

```
29 library CurrencyLibrary {
```

CVF-54 Reported

- **Category** Suboptimal
- **Source** Currency.sol

Recommendation This errors could be made more useful by adding certain parameters into them.

```
31 error NativeTransferFailed();
```

```
34 error ERC20TransferFailed();
```

CVF-55 Reported

- **Category** Procedural
- **Source** PoolKey.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.19;
```

CVF-56 Reported

- **Category** Documentation
- **Source** PoolKey.sol

Description Here “the first bit” sounds ambiguous.

Recommendation Rephrase as “the highest bit”.

```
13 /// @notice The pool swap fee, capped at 1_000_000. If the first bit
   ↪  is 1, the pool has a dynamic fee and must be exactly equal to
   ↪  0x800000
```

CVF-57 Reported

- **Category** Procedural
- **Source** BalanceDelta.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-58 Reported

- **Category** Suboptimal
- **Source** BalanceDelta.sol

Recommendation This could be simplified as: balanceDelta := or(shl(128, _amount0), shr(128(shl(128, _amount1))))

```
16 balanceDelta := or(shl(128, _amount0), and(sub(shl(128, 1), 1),
   ↪ _amount1))
```

CVF-59 Reported

- **Category** Procedural
- **Source** BalanceDelta.sol

Recommendation The functionality of these calls is quite simple and could be also incorporated into the assembly blocks for efficiency.

```
31 return toBalanceDelta(res0.toInt128(), res1.toInt128());
```

```
45 return toBalanceDelta(res0.toInt128(), res1.toInt128());
```

CVF-60 Reported

- **Category** Procedural
- **Source** Slot0.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-61 Reported

- **Category** Suboptimal
- **Source** Slot0.sol

Description So there are actually two fields for protocol fees, rather than one.

Recommendation Reflect this in the “layout” section above.

```
19 * Protocol fee, expressed in hundredths of a bip, upper 12 bits are
      ↪ for 1->0, and the lower 12 are for 0->1
```

CVF-62 Reported

- **Category** Documentation
- **Source** Slot0.sol

Description The number format for thsi value is unclear.

Recommendation Explain in the documentation comment.

```
24 * Used for the lp fee, either static at initialize or dynamic via
      ↪ hook
   * uint24 lpFee;
```

CVF-63 Reported

- **Category** Readability
- **Source** Slot0.sol

Recommendation This value could be rendered as “type(uint160).max”.

```
32 uint160 internal constant MASK_160_BITS = 0
   ↪ x00FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF;
```

CVF-64 Reported

- **Category** Readability
- **Source** Slot0.sol

Recommendation This value could be rendered as “type(uint24).max”.

```
33 uint24 internal constant MASK_24_BITS = 0xFFFFFF;
```

CVF-65 Reported

- **Category** Suboptimal
- **Source** Slot0.sol

Description A protocol fee value is actually a pair of numbers, so using the “uint24” type for it is confusing.

Recommendation Use a user-defined type wrapping “uint24”.

```
52 function protocolFee(Slot0 _packed) internal pure returns (uint24
   ↪ _protocolFee) {
```

```
77 function setProtocolFee(Slot0 _packed, uint24 _protocolFee) internal
   ↪  pure returns (Slot0 _result) {
```

CVF-66 Reported

- **Category** Suboptimal
- **Source** Slot0.sol

Description The value “not(MAX_160_BITS” is constant.

Recommendation Don’t calculate it in run time.

```
67 _result := or(and(not(MASK_160_BITS), _packed), and(MASK_160_BITS,
   ↪ _sqrtPriceX96))
```

CVF-67 Reported

- **Category** Suboptimal
- **Source** Slot0.sol

Description The value “not(shl(TICK_OFFSET, MASK_24_BITS))” is constant.

Recommendation Don’t calculate it in run time.

```
73 _result := or(and(not(shl(TICK_OFFSET, MASK_24_BITS)), _packed), shl
   ↪ (TICK_OFFSET, and(MASK_24_BITS, _tick)))
```

CVF-68 Reported

- **Category** Suboptimal
- **Source** Slot0.sol

Description The value “not(shl(PROTOCOL_FEE_OFFSET, MASK_24_BITS))” is constant.

Recommendation Don’t calculate it in run time.

```
81 and(not(shl(PROTOCOL_FEE_OFFSET, MASK_24_BITS)), _packed),
```

CVF-69 Reported

- **Category** Suboptimal
- **Source** Slot0.sol

Description The value “not(shl(LP_FEE_OFFSET, MASK_24_BITS))” is constant.

Recommendation Don’t calculate it in run time.

```
90 or(and(not(shl(LP_FEE_OFFSET, MASK_24_BITS)), _packed), shl(
   ↪ LP_FEE_OFFSET, and(MASK_24_BITS, _lpFee)))
```

CVF-70 Reported

- **Category** Procedural
- **Source** PoolId.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-71 Reported

- **Category** Suboptimal
- **Source** PoolId.sol

Description The value “mul(32, 5)” is constant.

Recommendation Don’t calculate it in run time.

```
13 poolId := keccak256(poolKey, mul(32, 5))
```

CVF-72 Reported

- **Category** Procedural
- **Source** BeforeSwapDelta.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-73 Reported

- **Category** Suboptimal
- **Source** BeforeSwapDelta.sol

Description As this type actually encapsulates two values, using the “int256” type for internal representation doesn’t make much sense.

Recommendation Use “bytes32” instead.

```
7 // Upper 128 bits is the delta in specified tokens. Lower 128 bits
     ↪ is delta in unspecified tokens (to match the afterSwap hook)
  type BeforeSwapDelta is int256;
```

CVF-74 Reported

- **Category** Suboptimal
- **Source** BeforeSwapDelta.sol

Description The value “sub(shl(128, 1), 1)” is constant.

Recommendation Don’t calculate it in run time.

```
16 beforeSwapDelta := or(shl(128, deltaSpecified), and(sub(shl(128, 1),
     ↪  1), deltaUnspecified))
```

CVF-75 Reported

- **Category** Procedural
- **Source** SafeCast.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-76 Reported

- **Category** Procedural
- **Source** BitMath.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-77 Reported

- **Category** Procedural
- **Source** FullMath.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-78 Reported

- **Category** Suboptimal
- **Source** FullMath.sol

Description The value “mulmod(a, b, denominator)” was already calculated inside the “mulDiv” function.

Recommendation Reuse the already calculated value.

```
112 if (mulmod(a, b, denominator) != 0) {
```

CVF-79 Reported

- **Category** Procedural
- **Source** FixedPoint128.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-80 Reported

- **Category** Procedural
- **Source** FixedPoint128.sol

Description This library consists only of a constant.

Recommendation Move the constant to the top level and remove the library.

```
6 library FixedPoint128 {
```

CVF-81 Reported

- **Category** Procedural
- **Source** TickBitmap.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-82 Reported

- **Category** Suboptimal
- **Source** TickBitmap.sol

Description The subexpression “1 « bitPos” is calculated twice.

Recommendation Calculate once and reuse or calculate like this: unchecked{ mask = (1 « uint256(bitPos) + 1) - 1; }

```
92 uint256 mask = (1 << bitPos) - 1 + (1 << bitPos);
```

CVF-83 Reported

- **Category** Procedural
- **Source** TickBitmap.sol

Recommendation Multiplication by “tickSpacing” should be done in one place outside the ternary operator.

```
99  ? (compressed - int24(uint24(bitPos - BitMath.mostSignificantBit(
        ↪ masked)))) * tickSpacing
100 : (compressed - int24(uint24(bitPos))) * tickSpacing;
```

```
112 ? (compressed + int24(uint24(BitMath.leastSignificantBit(masked) -
        ↪ bitPos))) * tickSpacing
    : (compressed + int24(uint24(type(uint8).max - bitPos))) *
        ↪ tickSpacing;
```

CVF-84 Reported

- **Category** Procedural
- **Source** TickBitmap.sol

Recommendation Subtraction from “compressed” should be done in one place outside the ternary operator.

```
99  ? (compressed - int24(uint24(bitPos - BitMath.mostSignificantBit(
        ↪ masked)))) * tickSpacing
100 : (compressed - int24(uint24(bitPos))) * tickSpacing;
```

CVF-85 Reported

- **Category** Procedural
- **Source** TickBitmap.sol

Recommendation Addition to “compressed” should be done in one place outside the ternary operator.

```
112 ? (compressed + int24(uint24(BitMath.leastSignificantBit(masked) -
        ↪ bitPos))) * tickSpacing
    : (compressed + int24(uint24(type(uint8).max - bitPos))) *
        ↪ tickSpacing;
```

CVF-86 Reported

- **Category** Procedural
- **Source** CustomRevert.sol

Description Specifying a compiler version range without upper bound is a bad practice, as it is impossible to guarantee compatibility with future major releases.

Recommendation Specify as "^0.8.0". Also relevant for: IERC6909Claims.sol, IExtsload.sol, IExttload.sol, Extsload.sol, Exttload.sol, ERC6909Claims.sol,ERC6909.sol.

```
2 pragma solidity >=0.8.4;
```

CVF-87 Reported

- **Category** Procedural
- **Source** CustomRevert.sol

Description This version requirement is inconsistent with other files in this code base.

```
2 pragma solidity >=0.8.4;
```

CVF-88 Reported

- **Category** Procedural
- **Source** Position.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-89 Reported

- **Category** Suboptimal
- **Source** Position.sol

Recommendation This error could be made more useful by adding certain parameters into it.

```
16 error CannotUpdateEmptyPosition();
```

CVF-90 Reported

- **Category** Procedural
- **Source** TickMath.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-91 Reported

- **Category** Suboptimal
- **Source** TickMath.sol

Recommendation These errors could be made more useful by adding some parameters into them.

```
9 error InvalidTick();
```

```
11 error InvalidSqrtPrice();
```

CVF-92 Reported

- **Category** Suboptimal
- **Source** TickMath.sol

Recommendation This expression should use the “MAX_SQRT_PRICE” and “MIN_SQRT_PRICE” constants instead of hardcoded values.

```
29 1461446703485210103287273052203988822378723970342 - 4295128739 - 1;
```

CVF-93 Reported

- **Category** Suboptimal
- **Source** TickMath.sol

Recommendation This could be simplified as: MAX_TICK - MAX_TICK % tickSpacing

```
34 return (MAX_TICK / tickSpacing) * tickSpacing;
```

CVF-94 Reported

- **Category** Suboptimal
- **Source** TickMath.sol

Recommendation This could be simplified as: MIN_TICK - MIN_TICK % tickSpacing

```
41 return (MIN_TICK / tickSpacing) * tickSpacing;
```

CVF-95 Reported

- **Category** Procedural
- **Source** TickMath.sol

Description The value “shl(128, 1)” is calculated twice.

Recommendation Calculate once and reuse.

```
75 price := xor(shl(128, 1), mul(xor(shl(128, 1), 0
   ↪ xfffcb933bd6fad37aa2d162d1a594001), and(absTick, 0x1)))
```

CVF-96 Reported

- **Category** Suboptimal
- **Source** TickMath.sol

Description The value “xor(shl(128, 1), 0xfffcb933bd6fad37aa2d162d1a594001)” is actually a constant.

Recommendation Hardcode it instead of calculating.

```
75 price := xor(shl(128, 1), mul(xor(shl(128, 1), 0
   ↪ xfffcb933bd6fad37aa2d162d1a594001), and(absTick, 0x1)))
```

CVF-97 Reported

- **Category** Suboptimal
- **Source** TickMath.sol

Description These error bounds seem to be found analytically and could be too pessimistic.

Recommendation Iterate throw all valid ticks to empirically find the exact error bounds.

```
266 int24 tickLow = int24((log_sqrt10001 -
    ↪ 3402992956809132418596140100660247210) >> 128);
    int24 tickHi = int24((log_sqrt10001 +
    ↪ 291339464771989622907027621153398088495) >> 128);
```

CVF-98 Reported

- **Category** Procedural
- **Source** UnsafeMath.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-99 Reported

- **Category** Procedural
- **Source** FixedPoint96.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-100 Reported

- **Category** Procedural
- **Source** FixedPoint96.sol

Description This library consists only of constants.

Recommendation Move the constants to the top level and remove the library.

```
7 library FixedPoint96 {
```

CVF-101 Reported

- **Category** Procedural
- **Source** SqrtPriceMath.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-102 Reported

- **Category** Suboptimal
- **Source** SqrtPriceMath.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
15 error InvalidPriceOrLiquidity();
   error InvalidPrice();
   error NotEnoughLiquidity();
   error PriceOverflow();
```

CVF-103 Reported

- **Category** Procedural
- **Source** SqrtPriceMath.sol

Description This line is the same in both branches.

Recommendation Do this calculation in one place before the conditional statement.

```
42 uint256 product = amount * sqrtPX96;
```

```
55 uint256 product = amount * sqrtPX96;
```

CVF-104 Reported

- **Category** Suboptimal
- **Source** SqrtPriceMath.sol

Description The “mulDiv” function is very efficient in a generic case, while for particular cases better approaches could exist.

Recommendation Implement and use here a “shlDiv” function similar to “mulDiv” but performing left shift instead of multiplication.

```
92 : FullMath.mulDiv(amount, FixedPoint96.Q96, liquidity)
```

CVF-105 Reported

- **Category** Procedural
- **Source** SwapMath.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-106 Reported

- **Category** Suboptimal
- **Source** SwapMath.sol

Description The value “MAX_FEE_PIPS - _feePips” is calculated twice.

Recommendation Calculate once and reuse.

```
60    FullMath.mulDiv(uint256(-amountRemaining), MAX_FEE_PIPS -
         ↪ _feePips, MAX_FEE_PIPS);
```

```
69        : FullMath.mulDivRoundingUp(amountIn, _feePips, MAX_FEE_PIPS
             ↪  - _feePips);
```

```
100 feeAmount = FullMath.mulDivRoundingUp(amountIn, _feePips,
       ↪ MAX_FEE_PIPS - _feePips);
```

CVF-107 Reported

- **Category** Suboptimal
- **Source** SwapMath.sol

Description The value “-amountRemaining” is calculated twice.

Recommendation Calculate once and reuse.

```
60 FullMath.mulDiv(uint256(-amountRemaining), MAX_FEE_PIPS - _feePips,
       ↪ MAX_FEE_PIPS);
```

```
78 feeAmount = uint256(-amountRemaining) - amountIn;
```

CVF-108 Reported

- **Category** Procedural
- **Source** ProtocolFeeLibrary.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-109 Reported

- **Category** Suboptimal
- **Source** ProtocolFeeLibrary.sol

Description This value looks quite arbitrary.

Recommendation Support full fees range up to 100%.

```
5 // Max protocol fee is 0.1% (1000 pips)
  uint16 public constant MAX_PROTOCOL_FEE = 1000;
```

CVF-110 Reported

- **Category** Suboptimal
- **Source** ProtocolFeeLibrary.sol

Description These values actually depend on the value of the “MAX_PROTOCL_FEE” constant.

Recommendation Derive these values from the “MAX_PROTOCL_FEE” constant.

```
 9 uint24 internal constant FEE_0_THRESHOLD = 1001;
10 uint24 internal constant FEE_1_THRESHOLD = 1001 << 12;
```

CVF-111 Reported

- **Category** Procedural
- **Source** LPFeeLibrary.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-112 Reported

- **Category** Suboptimal
- **Source** LPFeeLibrary.sol

Recommendation This error could be made more useful by adding certain parameters into it.

```
12 error FeeTooLarge();
```

CVF-113 Reported

- **Category** Procedural
- **Source** Pool.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-114 Reported

- **Category** Suboptimal
- **Source** Pool.sol

Description Setting each field individually is suboptimal.

Recommendation Implement a function to pack all the components in a “slot0” value at once.

```
108 self.slot0 = Slot0.wrap(bytes32(0)).setSqrtPriceX96(sqrtPriceX96).
    ↪ setTick(tick).setProtocolFee(protocolFee)
      .setLpFee(lpFee);
```

CVF-115 Reported

- **Category** Unclear behavior
- **Source** Pool.sol

Recommendation These functions should emit some events.

```
112 function setProtocolFee(State storage self, uint24 protocolFee)
       ↪ internal {
```

```
118 function setLPFee(State storage self, uint24 lpFee) internal {
```

CVF-116 Reported

- **Category** Suboptimal
- **Source** Pool.sol

Description This check does more harm than good. It solves minor problem that apprear rarely at cost of making the most comnon use case more expensive.

Recommendation Remove this check.

```
315 if (!exactInput && (swapFee == LPFeeLibrary.MAX_LP_FEE)) {
        InvalidFeeForExactOut.selector.revertWith();
    }
```

CVF-117 Reported

- **Category** Procedural
- **Source** Pool.sol

Recommendation These checks should be done earlier.

```
325 if (params.sqrtPriceLimitX96 <= TickMath.MIN_SQRT_PRICE) {
        PriceLimitOutOfBounds.selector.revertWith(params.
           ↪ sqrtPriceLimitX96);
    }
```

```
332 if (params.sqrtPriceLimitX96 >= TickMath.MAX_SQRT_PRICE) {
        PriceLimitOutOfBounds.selector.revertWith(params.
           ↪ sqrtPriceLimitX96);
    }
```

CVF-118 Reported

- **Category** Suboptimal
- **Source** Pool.sol

Recommendation This condition could be simplified as: state.amountSpecifiedRemaining != 0 && state.sqrtPriceX96 != params.sqrtPriceLimitX96

340

CVF-119 Reported

- **Category** Readability
- **Source** Pool.sol

Recommendation Removing the logical “not” from this condition and interchanging corresponding branches would make code more readable.

```
366 if (!exactInput) {
```

CVF-120 Reported

- **Category** Readability
- **Source** Pool.sol

Recommendation This line could be simplified using the “-=” operator.

```
370 state.amountCalculated = state.amountCalculated - (step.amountIn +
    ↪ step.feeAmount).toInt256();
```

CVF-121 Reported

- **Category** Readability
- **Source** Pool.sol

Recommendation This line could be simplified using the “+=” operator.

```
376 state.amountCalculated = state.amountCalculated + step.amountOut.
    ↪ toInt256();
```

CVF-122 Reported

- **Category** Suboptimal
- **Source** Pool.sol

Description Setting two fields separately is suboptimal.

Recommendation Implement a function to set both fields at once.

```
431 self.slot0 = slot0Start.setTick(state.tick).setSqrtPriceX96(state.
       ↪ sqrtPriceX96);
```

CVF-123 Reported

- **Category** Procedural
- **Source** ParseBytes.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2
```

CVF-124 Reported

- **Category** Procedural
- **Source** Hooks.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.24;
```

CVF-125 Reported

- **Category** Suboptimal
- **Source** Hooks.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
70 error InvalidHookResponse();
```

```
73 error FailedHookCall();
```

```
76 error HookDeltaExceedsSwapAmount();
```

CVF-126 Reported

- **Category** Suboptimal
- **Source** Hooks.sol

Recommendation It would me more efficient to convert “permissions” into a bit mask and then compare with the corresponding bits of “self” at once.

```
84 permissions.beforeInitialize != self.hasPermission(
      ↪ BEFORE_INITIALIZE_FLAG)
       || permissions.afterInitialize != self.hasPermission(
          ↪ AFTER_INITIALIZE_FLAG)
       || permissions.beforeAddLiquidity != self.hasPermission(
          ↪ BEFORE_ADD_LIQUIDITY_FLAG)
       || permissions.afterAddLiquidity != self.hasPermission(
          ↪ AFTER_ADD_LIQUIDITY_FLAG)
       || permissions.beforeRemoveLiquidity != self.hasPermission(
          ↪ BEFORE_REMOVE_LIQUIDITY_FLAG)
       || permissions.afterRemoveLiquidity != self.hasPermission(
          ↪ AFTER_REMOVE_LIQUIDITY_FLAG)
90     || permissions.beforeSwap != self.hasPermission(BEFORE_SWAP_FLAG
          ↪ )
       || permissions.afterSwap != self.hasPermission(AFTER_SWAP_FLAG)
       || permissions.beforeDonate != self.hasPermission(
          ↪ BEFORE_DONATE_FLAG)
       || permissions.afterDonate != self.hasPermission(
          ↪ AFTER_DONATE_FLAG)
       || permissions.beforeSwapReturnDelta != self.hasPermission(
          ↪ BEFORE_SWAP_RETURNS_DELTA_FLAG)
       || permissions.afterSwapReturnDelta != self.hasPermission(
          ↪ AFTER_SWAP_RETURNS_DELTA_FLAG)
       || permissions.afterAddLiquidityReturnDelta != self.
          ↪ hasPermission(AFTER_ADD_LIQUIDITY_RETURNS_DELTA_FLAG)
       || permissions.afterRemoveLiquidityReturnDelta
           != self.hasPermission(
              ↪ AFTER_REMOVE_LIQUIDITY_RETURNS_DELTA_FLAG)
```

CVF-127 Reported

- **Category** Procedural
- **Source** Lock.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-128 Reported

- **Category** Procedural
- **Source** Lock.sol

Description This import isn't used.

Recommendation Remove it.

```
4 import {IHooks} from "../interfaces/IHooks.sol";
```

CVF-129 Reported

- **Category** Procedural
- **Source** NonZeroDeltaCount.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-130 Reported

- **Category** Procedural
- **Source** NonZeroDeltaCount.sol

Description This import isn't used.

Recommendation Remove it.

```
4 import {IHooks} from "../interfaces/IHooks.sol";
```

CVF-131 Reported

- **Category** Procedural
- **Source** Reserves.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-132 Reported

- **Category** Procedural
- **Source** StateLibrary.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.21;
```

CVF-133 Reported

- **Category** Procedural
- **Source** StateLibrary.sol

Recommendation These bitwise “and” operations are redundant, as Solidity tolerates junk in unused bits of narrow types.

```
58  sqrtPriceX96 := and(data, 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
      ↪ )
62  protocolFee := and(shr(184, data), 0xFFFFFF)
64  lpFee := and(shr(208, data), 0xFFFFFF)
96  liquidityGross := and(firstWord, 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF)
121 liquidityGross := and(value, 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF)
```

CVF-134 Reported

- **Category** Procedural
- **Source** StateLibrary.sol

Recommendation The “signextend” operation is redundant as Solidity tolerates junk in unused bits of narrow types.

```
60 tick := signextend(2, shr(160, data))
```

CVF-135 Reported

- **Category** Procedural
- **Source** TransientStateLibrary.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.21;
```

CVF-136 Reported

- **Category** Procedural
- **Source** TransientStateLibrary.sol

Description Solidity compiler is smart enough to precompute constant hash expressions.

Recommendation Use hash expressions instead of hardcoded hashes.

```
10 /// bytes32(uint256(keccak256("ReservesOf")) - 1)
bytes32 public constant RESERVES_OF_SLOT =
0x1e0745a7db1623981f0b2a5d4232364c00787266eb75ad546f190e6cebe9bd95;
```

```
13 // The slot holding the number of nonzero deltas. bytes32(uint256(
   ↪ keccak256("NonzeroDeltaCount")) - 1)
bytes32 public constant NONZERO_DELTA_COUNT_SLOT =
0x7d4b3164c6e45b97e7d87b7125a44c5828d005af88f9d751cfd78729c5d99a0b;
```

```
17 // The slot holding the unlocked state, transiently. bytes32(uint256
   ↪ (keccak256("Unlocked")) - 1)
bytes32 public constant IS_UNLOCKED_SLOT =
0xc090fc4683624cfc3884e9d8de5eca132f2d0ec062aff75d43c0465d5ceeab23;
```

CVF-137 Reported

- **Category** Procedural
- **Source** CurrencyDelta.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-138 Reported

- **Category** Procedural
- **Source** LiquidityMath.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-139 Reported

- **Category** Procedural
- **Source** LiquidityMath.sol

Description The only function implemented in this library is actually not liquidity specific. It just adds an int128 value to a uint128 value and returns the result as uint128.

Recommendation Move the function into a generic utility library and remove this library.

```
5 library LiquidityMath {
```

CVF-140 Reported

- **Category** Procedural
- **Source** LiquidityMath.sol

Recommendation This function could be implemented in pure Solidity like this: uint256 result = uint256(int256(uint256(x)) + int256(y)); if ((z = uint128(result)) != result) revert SafeCastOverflow();

```
10 function addDelta(uint128 x, int128 y) internal pure returns (
     ↪ uint128 z) {
```

CVF-141 Reported

- **Category** Documentation
- **Source** IERC20Minimal.sol

Description Here “balance of a token” sounds ambiguous.

Recommendation Rephrase as “token balance of an account”.

```
7 /// @notice Returns the balance of a token
```

CVF-142 Reported

- **Category** Procedural
- **Source** IERC6909Claims.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity >=0.8.0;
```

CVF-143 Reported

- **Category** Unclear behavior
- **Source** IERC6909Claims.sol

Recommendation This interface misses event definitions.

```
4 interface IERC6909Claims {
```

CVF-144 Reported

- **Category** Documentation
- **Source** IERC6909Claims.sol

Description The returned value is not documented.

Recommendation Give a descriptive name to the returned value and/or describe in the documentation comment.

```
28 function transfer(address receiver, uint256 id, uint256 amount)
      ↪ external returns (bool);
```

```
35 function transferFrom(address sender, address receiver, uint256 id,
      ↪ uint256 amount) external returns (bool);
```

```
41 function approve(address spender, uint256 id, uint256 amount)
      ↪ external returns (bool);
```

```
46 function setOperator(address spender, bool approved) external
      ↪ returns (bool);
```

CVF-145 Reported

- **Category** Procedural
- **Source**

Description Specifying a compiler version range without upper bound is a bad practice, as it is impossible to guarantee compatibility with future major releases.

Recommendation Specify as "^0.6.0 || ^0.7.0 || ^0.8.0".

```
2 pragma solidity >=0.6.0;
```

CVF-146 Reported

- **Category** Procedural
- **Source** IExtsload.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity >=0.6.0;
```

CVF-147 Reported

- **Category** Documentation
- **Source** IExtsload.sol

Recommendation It would be more reasonable to return an array of “bytes32“ values.

```
13 /// @return value The value of the sload-ed slots concatenated as
      ↪ dynamic bytes
```

CVF-148 Reported

- **Category** Procedural
- **Source** IExttload.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity >=0.8.24;
```

CVF-149 Reported

- **Category** Procedural
- **Source** IHooks.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.24;
```

CVF-150 Reported

- **Category** Procedural
- **Source** IProtocolFeeController.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-151 Reported

- **Category** Documentation
- **Source** IProtocolFeeController.sol

Description The number format of the returned value is unclear.

Recommendation Explain in the documentation comment.

```
10 function protocolFeeForPool(PoolKey memory poolKey) external view
   ↪ returns (uint24);
```

CVF-152 Reported

- **Category** Procedural
- **Source** IProtocolFees.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.19;
```

CVF-153 Reported

- **Category** Suboptimal
- **Source** IProtocolFees.sol

Recommendation This error could be made more useful by adding certain parameters into it.

```
13 error InvalidProtocolFee();
```

CVF-154 Reported

- **Category** Procedural
- **Source** IProtocolFees.sol

Recommendation The argument should be indexed.

```
18 event ProtocolFeeControllerUpdated(address protocolFeeController);
```

CVF-155 Reported

- **Category** Procedural
- **Source** IPoolManager.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.24;
```

CVF-156 Reported

- **Category** Suboptimal
- **Source** IPoolManager.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

```
18 error CurrencyNotSettled();
```

```
21 error PoolNotInitialized();
```

```
30 error TickSpacingTooLarge();
```

```
33 error TickSpacingTooSmall();
```

```
36 error CurrenciesOutOfOrderOrEqual();
```

CVF-157 Reported

- **Category** Procedural
- **Source** IPoolManager.sol

Recommendation The “id” parameter should be indexed.

```
56 PoolId id, Currency indexed currency0, Currency indexed currency1,
     ↪ uint24 fee, int24 tickSpacing, IHooks hooks
```

CVF-158 Reported

- **Category** Procedural
- **Source** IPoolManager.sol

Recommendation This parameter should be indexed.

```
80 address sender,
```

CVF-159 Reported

- **Category** Procedural
- **Source** IPoolManager.sol

Recommendation These functions should be replaced by constants.

```
89 /// @notice Returns the constant representing the maximum
   ↪ tickSpacing for an initialized pool key
90 function MAX_TICK_SPACING() external view returns (int24);
```

```
92 /// @notice Returns the constant representing the minimum
   ↪ tickSpacing for an initialized pool key
   function MIN_TICK_SPACING() external view returns (int24);
```

CVF-160 Reported

- **Category** Bad naming
- **Source** IPoolManager.sol

Description The semantics of the returned value is unclear.

Recommendation Give a descriptive name to the returned value and/or explain the semantics in the documentation comment.

```
103 returns (int24 tick);
```

CVF-161 Reported

- **Category** Documentation
- **Source** IPoolManager.sol

Description This phrase is confusing.

Recommendation Explain a typical use case for the function.

```
105 /// @notice All operations go through this function
```

CVF-162 Reported

- **Category** Documentation
- **Source** IPoolManager.sol

Description The returned values are actually unnamed.

Recommendation Give names to the returned values.

```
125 /// @return callerDelta The balance delta of the caller of
    ↪ modifyLiquidity. This is the total of both principal and fee
    ↪ deltas.
/// @return feeDelta The balance delta of the fees generated in the
    ↪ liquidity range. Returned for informational purposes.
```

CVF-163 Reported

- **Category** Documentation
- **Source** IPoolManager.sol

Description The returned value is actually unnamed.

Recommendation Give a name to the returned value.

```
141 /// @return swapDelta The balance delta of the address swapping
```

CVF-164 Reported

- **Category** Bad naming
- **Source** IPoolManager.sol

Description The semantics of the returned value is unclear.

Recommendation Give a descriptive name to the returned value and/or explain in the documentation comment.

```
152 returns (BalanceDelta);
```

CVF-165 Reported

- **Category** Procedural
- **Source**

Description Specifying a compiler version range without upper bound is a bad practice, as it is impossible to guarantee compatibility with future major releases.

Recommendation Specify as “^0.6.0 || ^0.7.0 || ^0.8.0”.

```
2 pragma solidity >=0.6.0;
```

CVF-166 Reported

- **Category** Procedural
- **Source** Extsload.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity >=0.6.0;
```

CVF-167 Reported

- **Category** Procedural
- **Source** ERC6909.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity >=0.8.0;
```

CVF-168 Reported

- **Category** Bad naming
- **Source** ERC6909.sol

Description The semantics of keys and values for these mappings is unclear.

Recommendation Give descriptive names to the keys and values and/or explain in documentation comments.

```
25 mapping(address => mapping(address => bool)) public isOperator;
```

```
27 mapping(address => mapping(uint256 => uint256)) public balanceOf;
```

```
29 mapping(address => mapping(address => mapping(uint256 => uint256)))
   ↪ public allowance;
```

CVF-169 Reported

- **Category** Procedural
- **Source** Exttload.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity >=0.8.24;
```

CVF-170 Reported

- **Category** Procedural
- **Source** NoDelegateCall.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.20;
```

CVF-171 Reported

- **Category** Procedural
- **Source** ERC6909Claims.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity >=0.8.0;
```

CVF-172 Reported

- **Category** Procedural
- **Source** ProtocolFees.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.19;
```

CVF-173 Reported

- **Category** Unclear behavior
- **Source** ProtocolFees.sol

Description This event is emitted even if nothing actually changed.

```
33 emit ProtocolFeeControllerUpdated(address(controller));
```

CVF-174 Reported

- **Category** Procedural
- **Source** ProtocolFees.sol

Description The “data” value is converted to “uint24” twice.

Recommendation Convert once and reuse.

```
84 (success, protocolFee) = (returnData == uint24(returnData)) &&
      ↪ uint24(returnData).isValidProtocolFee()
```

CVF-175 Reported

- **Category** Procedural
- **Source** PoolManager.sol

Description This version requirement is inconsistent with other files in this code base.

Recommendation Use the same compiler version requirement across the code base.

```
2 pragma solidity ^0.8.24;
```

CVF-176 Reported

- **Category** Procedural
- **Source** PoolManager.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
98 constructor(uint256 controllerGasLimit) ProtocolFees(
      ↪ controllerGasLimit) {}
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting