Report

v. 1.0

Customer

Chronicle

Smart Contract Audit

Chronicle Protocol

10th August 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	10.08.23	A. Zveryanskaya	Initial Draft
0.2	10.08.23	A. Zveryanskaya	Minor revision
1.0	10.08.23	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.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

docs/			
	Schnorr.md		
/			
	IScribe.sol	IScribeOptimistic.sol	Scribe.sol
	ScribeOptimistic.sol		
libs/			
	LibBytes.sol	LibSchnorr.sol	LibSchnorrData.sol
	LibSecp256k1.sol		

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.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 11 major, and a few less important issues.

	Info	Fixed
Major	6	5
Moderate	13	2
Minor	29	6

Fixed 13 out of 61 issues

6 Major Issues

CVF-1. FIXED

- **Category** Unclear behavior
- **Source** Schnorr.md

Description The concatenation operation is not well defined for field elements and booleans.

Recommendation Consider specifying the lengths and conversion rules into bitstrings for the arguments.

Client Comment *Updated definition of concatenation operator as being Solidity's 'abi.encodePacked()' function.*

```
54 1. Compute _challenge_ `e = H(ₓP ||ₚ P || m ||ₑ R) mod Q`
```

CVF-2. INFO

- **Category** Flaw
- **Source** Schnorr.md

Recommendation The generator G must be part of the input.

Client Comment *Generator G is part of the input as R_e = addressOf([k]G). Also referencing BIP-340 here, see https://en.bitcoin.it/wiki/BIP_0340#Default_Signing.*

```
54 1. Compute _challenge_ `e = H(ₓP ||ₚ P || m ||ₑ R) mod Q`
```

CVF-3. INFO

- **Category** Flaw
- **Source** Schnorr.md

Description For some Q the result is not sufficiently uniform mod Q.

Recommendation Consider using a 512-bit hash.

Client Comment *We defined Q to be the order of secp256k1, for which the result is sufficiently uniform as Q is ~100-(3*10^{-37})% of H()'s image. Also referencing BIP-340 here, see https://en.bitcoin.it/wiki/BIP_0340#Default_Signing. We try to be as similar to BIP-340 as possible to ease backend implementation and audibility. We'll therefore accept the risk, as bitcoin does.*

```
54 1. Compute _challenge_ `e = H(ₓP ||ₚ P || m ||ₑ R) mod Q`
```

CVF-4. FIXED

- **Category** Flaw
- **Source** Schnorr.md

Description The MuSig paper indeed notices that this kind of aggregation is vulnerable to rogue key attacks. However, this is not only kind of attacks that is possible. Imagine that sum of private keys of set A is equal to the sum of private keys of set B - for example due to colluding signers. Note that this kind of collusion does not even require the signers to know the keys of each other, only the sum. Now, as the sums of private keys are equal so are aggregated public keys, Therefore, one can not figure out who signed the message – set A or set B. Moreover, the union of these sets yields a public key with 0 private key, which opens many kinds of other attacks.

Recommendation Consider requiring each signer to prove that their private key is randomly generated, or make sure that there is only one aggregation key ever possible.

Client Comment *Updated Schnorr specification to acknowledge the issue. Furthermore, added a script to verify no linear relationships between any subsets of given public keys exist. This script will be invoked before lifting a new feed to ensure such relationships between feeds' public keys will not exist in onchain configurations. NOTE: If you know of any way to prove a private key was created cryptographically sound, would love to implement such a recommendation.*

```
77 Let the signers' public keys be:
       signers = [₁pubKey, ₂pubKey, ..., ₙpubKey]

80 Let the aggregated public key be:
       aggPubKey = sum(signers)
                 = ₁pubKey     + ₂pubKey     + ... + ₙpubKey
                 = [₁privKey]G + [₂privKey]G + ... + [ₙprivKey]G
                 = [₁privKey   + ₂privKey    + ... + ₙprivKey]G
```

CVF-5. FIXED

- **Category** Suboptimal
- **Source** LibBytes.sol

Recommendation It would be more efficient to do this in an assembly block using "BYTE" opcode: assembly { result := byte(sub(0×20, index), word) }

Client Comment *(Note that recommendation is erroneous, should be 'sub(0×1F, index)')*

```
24 result = (word >> (index << 3)) & 0xFF;
```

CVF-6. INFO

- **Category** Suboptimal
- **Source** LibSchnorrData.sol

Recommendation This function could be simplified as: return schnorrData.signersBlob[index];

Client Comment *The recommended solution reverts for IndexOutOfBounds, which is explicitly defined to not be allowed for 'getSignerIndex()' (see function doc).*

```
59 function getSignerIndex(
```

CVF-7. FIXED

- **Category** Suboptimal
- **Source** IScribe.sol

Description Indexing numeric event parameters usually doesn't make much sense, as indexed only speed up exact value searches, but not range searches.

Recommendation Consider not indexing numeric parameters "val" and "age".

Client Comment *Implemented as recommended.*

```
63 address indexed caller, uint128 indexed val, uint32 indexed age
```

CVF-8. FIXED

- **Category** Procedural
- **Source** IScribe.sol

Description A common way to report errors in Solidity is to revert with a named error.

Recommendation Consider doing so, instead of returning a flag and a message.

Client Comment *Updated and renamed the function. The function is now 'isAcceptableSignatureNow' returning a bool. The name was updated to make it clearer that a once-acceptable signature may not be acceptable in the future (e.g. bar updated). Note that the function is implemented as a predicate instead, i.e. it returns bool instead of reverting.*

```
144 /// @return err Null if `message`'s integrity proven via `
    ↪ schnorrData`,
    ///             abi-encoded custom error otherwise.
```

CVF-9. INFO

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

Recommendation Should probably be “<” rather than “!=”.

Client Comment *Weakening the check to ‘<‘ does not introduce new security guarantees, but opens the door to potential block gas limit issues. Note that ‘opChallenge‘ always needs to be able to invalidate an invalid ‘opPoke‘, i.e. ‘_verifySignature()‘ is never allowed to run out of gas. As EVM compatibility is a spectrum and Scribe will be launched on potentially many different EVM chains, we rather not introduce assumptions to the chains respective block gas limits.*

```
177 if (numberSigners != bar) {
```

CVF-10. INFO

- **Category** Documentation
- **Source** Scribe.sol

Description This commented out line is confusing.

Recommendation Consider either uncommenting it, or removing, or explaining why it is commented out.

Client Comment *Assert statements indicate invariants of the system. Defining them inside the code as much as possible is useful during development. Furthermore, formal testing methods heavily depend on assert statements. However, following best practices we don’t execute them onchain.*

```
213     // assert(aggPubKey.x != signerPubKey.x); // Indicates rogue-key
            ↪  attack
```

```
304 // assert(uint(answer) == uint(_pokeData.val));
```

```
315 // assert(index != 0 ? !_pubKeys[index].isZeroPoint() : true);
```

CVF-11. INFO

- **Category** Documentation
- **Source** ScribeOptimistic.sol

Description This commented out line is confusing.

Recommendation Consider either uncommenting it, or removing, or explaining why it is commented out.

Client Comment *Assert statements indicate invariants of the system. Defining them inside the code as much as possible is useful during development. Furthermore, formal testing methods heavily depend on assert statements. However, following best practices we don't execute them onchain.*

```
390 // assert(uint(answer) == uint(pokeData.val));
```

7 Moderate Issues

CVF-12. FIXED

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

Description The protocol is not guaranteed to fail if ‘pubkey‘ is not a valid EC point.

Recommendation Consider adding this check.

Client Comment *Implemented as recommended. While we still assume this check to be unnecessary, we enabled it as additional defense mechanism.*

```
20 LibSecp256k1.Point memory pubKey,
```

CVF-13. INFO

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

Description This is suspicious.

Recommendation Consider initiating the pubkey set with some valid keys.

Client Comment *Having ‘_pubKeys[0]‘ as zero point is clearly documented as invariant in ‘docs/Invariants.md‘. The reason for this invariant is that a ‘calldataload()‘ of non-existing calldata returns zero (see ‘tests/EVMTest.sol::testFuzz_calldataload_ReadingNonExisting-CalldataReturnsZero()‘). This ensures erroneous calls to ‘verifySignature()‘ always fail as early as possible.*

```
70 // Let _pubKeys[0] be the zero point.
   _pubKeys.push(LibSecp256k1.ZERO_POINT());
```

CVF-14. INFO

- **Category** Flaw
- **Source** Schnorr.md

Description This doesn't allow signing the same message twice with different nonces, which is quite common use case.

Recommendation Consider generating "k" as H(x || m || nonce), in which case "k" won't be called "nonce" anymore.

Client Comment *This is not a use case in our protocol as a timestamp is always part of the message. Furthermore, using a deterministic nonce helps differential fuzz testing different signer implementations and generally during test environments where the private keys are known.*

```
27 Note that `k` can be deterministically constructed via `H(x || m) mod
   ↪ Q`.
```

CVF-15. FIXED

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

Description This call returns a zero address on incorrect arguments.

Recommendation Consider explicitly returning false in such a case. Currently, false result is guaranteed only due to a check in the beginning of the function that ensures that "commitment" is not zero, however, the reason for that check is different.

Client Comment *Added additional comment.*

```
91 address recovered =
       ecrecover(bytes32(msgHash), uint8(v), bytes32(r), bytes32(s));
```

CVF-16. INFO

- **Category** Suboptimal
- **Source** Scribe.sol

Description Setting the poke age to the current time rather then to “pokeData.age” means that a poke with newer age could be rejected if its age is older than current time.

Recommendation Consider setting the age to “pokeData.age”.

Client Comment *The age of a val is defined as the timestamp Scribe accepts it as it’s new value. This is in sync with MakerDAO’s old Median contract, see https://github.com/maker-dao/median/blob/master/src/median.sol#L126.*

```
128 _pokeData.age = uint32(block.timestamp);
```

CVF-17. INFO

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

Description Overflow is possible here.

Recommendation Consider using safe conversion.

Client Comment *Overflow is only possible in ~100 years, for which we assume to have launched a new version.*

```
128 _pokeData.age = uint32(block.timestamp);
```

CVF-18. INFO

- **Category** Procedural
- **Source** Scribe.sol

Description A signed message doesn’t include chain ID nor contract address, which makes it possible to use it not where it was supposed to be used.

Recommendation Consider following EIP-712 standard for signed messages.

Client Comment *This is intended behavior. Price messages are designed to be able to send to different Scribe instances (with same wat) on different chains. This enables feeds to not have to implement chain-specific functionality. This is in sync with MakerDAO’s old Median contract, see https://github.com/makerdao/median/blob/master/src/median.sol#L94.*

```
142 keccak256(abi.encodePacked(wat, pokeData.val, pokeData.age))
```

CVF-19. INFO

- **Category** Procedural
- **Source** Scribe.sol

Description A signed message doesn't include chain ID nor contract address, which makes it possible to use it not where it was supposed to be used.

Recommendation Consider following EIP-712 standard for signed messages.

Client Comment *See above.*

```
143 )
```

CVF-20. INFO

- **Category** Flaw
- **Source** Scribe.sol

Description There is no check to ensure that a value is actually available.

Recommendation Consider adding such a check and reverting in case it is not.

Client Comment *This is intended behavior to stay compatible with Chainlinks behavior.*

```
303 answer = int(uint(_pokeData.val));
```

CVF-21. INFO

- **Category** Suboptimal
- **Source** Scribe.sol

Description The "ecrecover" function returns a zero address on incorrect input.

Recommendation Consider explicitly checking whether the returned address is zero and reverting in such a case.

Client Comment *Assuming the keccack hash function is not broken, the difficulty of producing a 160-bit zero hash with an input of 64 bytes is well within our security assumptions. Also, see https://github.com/chroniclеprotocol/scribe/blob/main/docs/Schnorr.md#other-ecurity-considerations for more info.*

```
413 address recovered = ecrecover(
```

CVF-22. INFO

- **Category** Suboptimal
- **Source** Scribe.sol

Description We didn't reviewed the toll functionality, but this override looks very suspicions.

Recommendation Consider either removing it or clearly explaining.

Client Comment *Via overriding this function we define the access control for Toll's admin functions. Toll's admin functions can therefore only be accessed if the caller is auth'ed as defined via the inherited Auth module.*

```
553 function toll_auth() internal override(Toll) auth {}
```

CVF-23. INFO

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

Description Overflow is possible here.

Recommendation Consider using safe conversion.

Client Comment *Overflow is only possible in ~100 years, for which we assume to have launched a new version.*

```
69  if (pokeData.age > uint32(block.timestamp)) {
```

```
87  _pokeData.age = uint32(block.timestamp);
```

```
124     opPokeData.age + opChallengePeriod <= uint32(block.timestamp);
```

```
140 if (pokeData.age > uint32(block.timestamp)) {
        revert FutureMessage(pokeData.age, uint32(block.timestamp));
```

```
184 _opPokeData.age = uint32(block.timestamp);
```

```
202     opPokeData.age + opChallengePeriod > uint32(block.timestamp);
```

```
403     opPokeData.age + opChallengePeriod <= uint32(block.timestamp);
```

```
470     opPokeData.age + opChallengePeriod <= uint32(block.timestamp);
```

```
506 _pokeData.age = uint32(block.timestamp);
```

CVF-24. INFO

- **Category** Suboptimal
- **Source** ScribeOptimistic.sol

Description Setting the poke age to the current time rather then to “pokeData.age” means that a poke with newer age could be rejected if its age is older than current time.

Recommendation Consider setting the age to “pokeData.age”.

Client Comment *The age of a val is defined as the timestamp ScribeOptimistic accepts it as it’s new value. This is in sync with MakerDAO’s old Median contract, see https://github.com/-makerdao/median/blob/master/src/median.sol#L126.*

```
87 _pokeData.age = uint32(block.timestamp);
```

CVF-25. INFO

- **Category** Suboptimal
- **Source** ScribeOptimistic.sol

Description The “ecrecover” function returns a zero address on incorrect input.

Recommendation Consider explicitly checking whether the returned address is zero and reverting in such a case.

Client Comment *‘lift‘ is the only function able to add new addresses to the ‘_feeds‘ mapping (see docs/Invariants.md). Assuming there exists no known public key with Ethereum address of zero, a return value of zero from the ecrecover function reverts with ‘SignerNotFeed(address(0))‘.*

```
145 address signer = ecrecover(
```

CVF-26. INFO

- **Category** Procedural
- **Source** ScribeOptimistic.sol

Description A signed message doesn't include chain ID nor contract address, which makes it possible to use it not where it was supposed to be used.

Recommendation Consider following EIP-712 standard for signed messages.

Client Comment *This is intended behavior. Price messages are designed to be able to send to different Scribe instances (with same wat) on different chains. This enables feeds to not have to implement chain-specific functionality. This is in sync with MakerDAO's old Median contract, see https://github.com/makerdao/median/blob/master/src/median.sol#L94.*

```
282 abi.encodePacked(
        "\x19Ethereum Signed Message:\n32",
        keccak256(
            abi.encodePacked(
                wat,
                pokeData.val,
                pokeData.age,
                schnorrData.signature,
290             schnorrData.commitment,
                schnorrData.signersBlob
            )
        )
    )
```

8 Minor Issues

CVF-27. INFO

- **Category** Suboptimal
- **Source** Schnorr.md

Description It is unclear whether this verification is performed in a trustless way.

Client Comment *This verification is performed in Scribe's 'lift()' function via interpreting an ECDSA signature as a zero-knowledge proof proving the knowledge of the corresponding private key.*

```
88 In order to prevent such attacks, it **MUST** be verified that
      ↪ participating
   public keys own the corresponding private key.
```

CVF-28. INFO

- **Category** Procedural
- **Source** LibBytes.sol

Recommendation Consider specifying as "^0.8.0" unless there is something special about this particular version. Also relevant for: LibSecp256k1.sol, LibSchnorrData.sol, LibSchnorr.sol, IScribe.sol, IScribeOptimistic.sol, Scribe.sol, ScribeOptimistic.sol.

Client Comment *The project uses custom errors which were introduced in v0.8.4. Furthermore, we don't want to endorse versions that we didn't test sufficiently. The officially supported versions are defined via the CI workflow "solc-version-tests". The currently supported versions are v0.8.16 - v0.8.20.*

```
1 pragma solidity ^0.8.16;
```

CVF-29. INFO

- **Category** Bad datatype
- **Source** LibBytes.sol

Recommendation The type "bytes32" would be more appropriate for "word", as "word" is not treated as a number.

Client Comment *Acknowledged.*

```
14 function getByteAtIndex(uint word, uint index)
```

CVF-30. INFO

- **Category** Suboptimal
- **Source** LibSecp256k1.sol

Recommendation The “and” operation is redundant, as Solidity tolerates junk in unused bytes of values shorter than 32 bytes.

Client Comment *Acknowledged. However, to my knowledge, this is an implementation detail of solc and not part of their compatibility promise. In order to increase compatibility with future versions, and avoid footguns, the recommendation is not implemented.*

```
71 addr := and(keccak256(self, 0x40), ADDRESS_MASK)
```

CVF-31. FIXED

- **Category** Suboptimal
- **Source** LibSecp256k1.sol

Recommendation This could be optimized as: return self.x | self.y == 0;

Client Comment *Implemented as recommended*

```
87 return self.x == 0 && self.y == 0;
```

CVF-32. FIXED

- **Category** Suboptimal
- **Source** LibSecp256k1.sol

Recommendation Bitwise “and” would be more efficient.

Client Comment *Implemented as recommended*

```
96 return self.y % 2;
```

CVF-33. INFO

- **Category** Suboptimal
- **Source** LibSchnorrData.sol

Recommendation It would be more efficient to zero the lowest 5 bits via bitwise operations.

Client Comment *Acknowledged. Added tests to document optimization, but decided to not implement them for the current version.*

```
65 let wordIndex := mul(div(index, WORD_SIZE), WORD_SIZE)
```

CVF-34. INFO

- **Category** Suboptimal
- **Source** LibSchnorrData.sol

Recommendation It would be more efficient to calculate the inverted lowest 5 bits via bitwise operations.

Client Comment *Acknowledged. Added tests to document optimization, but decided to not implement them for the current version.*

```
81 byteIndex = 31 - (index % WORD_SIZE);
```

CVF-35. FIXED

- **Category** Suboptimal
- **Source** LibSchnorr.sol

Recommendation Consider still enabling this check to enforce that the signature is a valid field element.

Client Comment *Implemented as recommended. While we still assume this check to be unnecessary, we enabled it as additional defense mechanism.*

```
40 // Note that this check MUST be enabled for general purpose Schnorr
   // signature verifications!
   //
   // if (uint(signature) >= LibSecp256k1.Q()) {
   //     return false;
   // }
```

CVF-36. INFO

- **Category** Procedural
- **Source** IScribe.sol

Description We didn't review this file.

Client Comment *Acknowledged*

```
3 import {IChronicle} from "chronicle-std/IChronicle.sol";
```

CVF-37. FIXED

- **Category** Procedural
- **Source** IScribe.sol

Recommendation The "index" parameters should be indexed.

Client Comment *Implemented as recommended.*

```
70 event FeedLifted(address indexed caller, address indexed feed, uint
   ↪ index);
```

```
76 event FeedDropped(address indexed caller, address indexed feed, uint
   ↪  index);
```

CVF-38. INFO

- **Category** Documentation
- **Source** IScribe.sol

Description These comments are confusing. Do they mean that these returned values are actually constant?

Recommendation Consider elaborating more.

Client Comment *Acknowledged. The return values are indeed constant. Scribe has no notion of Chainlink's terms such as roundId, etc. However, the returns values are chosen to pass common Chainlink integrator's data validation checks.*

```
108 /// @return roundId 1.
```

```
110 /// @return startedAt 0.
```

```
112 /// @return answeredInRound 1.
```

CVF-39. INFO

- **Category** Suboptimal
- **Source** IScribe.sol

Recommendation The “isFeed” returned value is redundant, as it could be derived as “feed != address(0)”.

Client Comment *Acknowledged*

```
181 returns (bool isFeed, address feed);
```

CVF-40. INFO

- **Category** Suboptimal
- **Source** IScribe.sol

Recommendation It would be more efficient to return a single array of structs with two fields, rather than two parallel arrays.

Client Comment *Acknowledged. However, we would like to not introduce another type into IScribe.*

```
189 returns (address[] memory feeds, uint[] memory feedIndexes);
```

CVF-41. INFO

- **Category** Suboptimal
- **Source** IScribe.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays.

Client Comment *Acknowledged. However, we would like to not introduce another type into IScribe.*

```
210 LibSecp256k1.Point[] memory pubKeys,
    ECDSAData[] memory ecdsaDatas
```

CVF-42. INFO

- **Category** Suboptimal
- **Source** IScribeOptimistic.sol

Recommendation The old value parameters are redundant as they could be derived from the previous events.

Client Comment *Acknowledged. Keeping as is to reduce monitoring complexity.*

```
65 uint16 oldOpChallengePeriod,
```

```
75 uint oldMaxChallengeReward,
```

CVF-43. INFO

- **Category** Procedural
- **Source** Scribe.sol

Description We didn't review these files.

Client Comment *Acknowledged*

```
3 import {IChronicle} from "chronicle-std/IChronicle.sol";
  import {Auth} from "chronicle-std/auth/Auth.sol";
  import {Toll} from "chronicle-std/toll/Toll.sol";
```

CVF-44. INFO

- **Category** Procedural
- **Source** Scribe.sol

Recommendation Constants are usually named IN_UPPER_CASE.

Client Comment *The 'IScribe' interface defines the 'maxFeeds()(uint)' function. We could implement the function by returning the constant. However, it seems ok to break convention in this case to reduce boilerplate code.*

```
26 uint public constant maxFeeds = type(uint8).max - 1;
```

```
29 uint8 public constant decimals = 18;
```

CVF-45. INFO

- **Category** Suboptimal
- **Source** Scribe.sol

Description The value calculated here is constant and doesn't need to be computed at deployment time.

Recommendation Consider computing at build time.

Client Comment *Acknowledged*

```
76 mstore(0x00, _pubKeys.slot)
   pubKeysSlot := keccak256(0x00, 0x20)
```

CVF-46. INFO

- **Category** Suboptimal
- **Source** Scribe.sol

Recommendation This check should be performed earlier.

Client Comment *Where though? The 'signerPubKey' and 'signer' variables need to be updated before the check is executed (see error inside the check). If there are no security issues, we'll leave it as is.*

```
203 if (signerPubKey.isZeroPoint()) {
```

CVF-47. INFO

- **Category** Suboptimal
- **Source** Scribe.sol

Recommendation This check should be performed earlier.

Client Comment *Acknowledged*

```
258 require(val != 0);
```

CVF-48. INFO

- **Category** Suboptimal
- **Source** Scribe.sol

Description This should be performed only if “val” is not zero.

Client Comment *Why? Scribe’s val can be set to zero at any given point. Being able to receive the age of that val seems reasonable.*

```
272 uint age = _pokeData.age;
```

CVF-49. INFO

- **Category** Suboptimal
- **Source** Scribe.sol

Recommendation It would be more efficient to return a single array or structs with two fields, rather than two parallel arrays.

Client Comment *Acknowledged. However, we would like to not introduce another type into IScribe.*

```
334 function feeds() external view returns (address[] memory, uint[]
    ↪ memory) {
```

CVF-50. INFO

- **Category** Suboptimal
- **Source** Scribe.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays.

Client Comment *Acknowledged. However, we would like to not introduce another type into IScribe.*

```
388 LibSecp256k1.Point[] memory pubKeys,
    ECDSAData[] memory ecdsaDatas
```

CVF-51. INFO

- **Category** Suboptimal
- **Source** Scribe.sol

Recommendation This check should be done earlier.

Client Comment *Disagree. This is a postcondition executed directly after the lift operation. Note that the event emission is part of the lift operation.*

```
429 require(index <= maxFeeds);
```

CVF-52. INFO

- **Category** Suboptimal
- **Source** Scribe.sol

Recommendation Variables are redundant here, consider doing like: revert(add(err, 0×20), mload(err))

Client Comment *Acknowledged. Keeping as is as variables increase readability.*

```
482 let size := mload(err)
    let offset := add(err, 0x20)
    revert(offset, size)
```

CVF-53. INFO

- **Category** Suboptimal
- **Source** Scribe.sol

Recommendation Left shift would be more efficient.

Client Comment *Acknowledged*

```
501 let realIndex := mul(index, 2)
```

CVF-54. INFO

- **Category** Suboptimal
- **Source** Scribe.sol

Description Variables are redundant here.

Recommendation Consider doing like: mstore(pubKey, sload(slot)) mstore(add(pubKey, 0×20), sload(add(slot, 1)))

Client Comment *Acknowledged. Keeping as is as variables increase readability.*

```
507 let x := sload(slot)
    let y := sload(add(slot, 1))
```

```
511 mstore(pubKey, x)
    mstore(add(pubKey, 0x20), y)
```

CVF-55. INFO

- **Category** Procedural
- **Source** ScribeOptimistic.sol

Description We didn't review this file.

Client Comment *Acknowledged*

```
3 import {IChronicle} from "chronicle-std/IChronicle.sol";
```

CVF-56. FIXED

- **Category** Bad datatype
- **Source** ScribeOptimistic.sol

Recommendation The default challenge period should be a named constant.

Client Comment *Implemented as recommended.*

```
50 _setOpChallengePeriod(1 hours);
```

CVF-57. INFO

- **Category** Procedural
- **Source** ScribeOptimistic.sol

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

Client Comment *Implementing receive functionality like this is quite common and expected to be understood by readers.*

```
53 receive() external payable {}
```

CVF-58. INFO

- **Category** Suboptimal
- **Source** ScribeOptimistic.sol

Description This overridden function looks identical to the function is overrides.

Recommendation Consider not overriding.

Client Comment *Taking a closer look reveals that the functions are not identical. While the age of a val in Scribe is defined via '_pokeData.age', the age in ScribeOptimistic is defined via '_currentPokeData().age'. Creating a function like '_getCurrentPokeDataAge()' in Scribe that is overridden in ScribeOptimistic would be possible. However, doing so would leak logic from ScribeOptimistic into the upstream Scribe contract, which we want to circumvent.*

```
57 function _poke(PokeData calldata pokeData, SchnorrData calldata
   ↪ schnorrData)
    internal
    override(Scribe)
```

CVF-59. FIXED

- **Category** Documentation
- **Source** ScribeOptimistic.sol

Recommendation Consider documenting why overflow is not possible here, as the range check is far too away in the code.

Client Comment *Added additional comment explaining why downcast is safe.*

```
162 opFeedIndex = uint8(signerIndex);
```

CVF-60. INFO

- **Category** Suboptimal
- **Source** ScribeOptimistic.sol

Description This check consumes gas on every invocation.

Recommendation Consider refactoring to not executing this code during development at all.

Client Comment *Acknowledged. Note that auth'ed functions do not get executed frequently so gas costs are of no big concern. We rather keep the clean formal definition of '_afterAuthedAction()' being executed after every auth'ed configuration change.*

458
```
// Do nothing during deployment.
if (address(this).code.length == 0) return;
```

CVF-61. INFO

- **Category** Suboptimal
- **Source** ScribeOptimistic.sol

Description The storage value "_pokeData.age" is assigned twice.

Recommendation Consider refactoring to assign at most once.

Client Comment *Acknowledged. Keeping as-is because savings are marginal.*

485
```
    _pokeData = opPokeData;
```

506
```
_pokeData.age = uint32(block.timestamp);
```

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