ABDK CONSULTING

SMART CONTRACT AUDIT

Chainflip

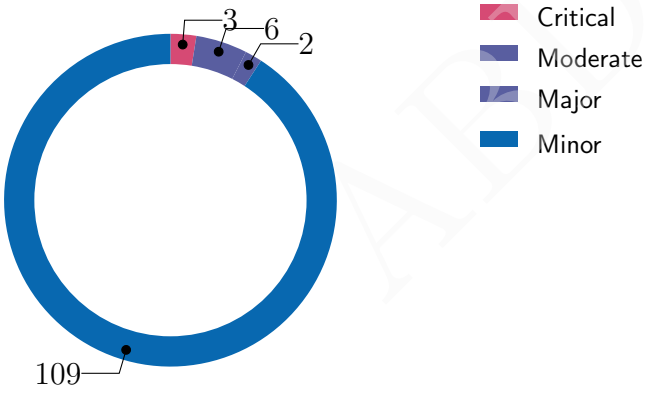
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
29th September 2021

We've been asked to review the 14 files in a github repo. We found 3 critical, 2 major, and a few less important issues.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Opened
CVF-2	Minor	Bad datatype	Opened
CVF-3	Minor	Bad datatype	Opened
CVF-4	Minor	Bad datatype	Opened
CVF-5	Minor	Suboptimal	Opened
CVF-6	Minor	Suboptimal	Opened
CVF-8	Minor	Bad datatype	Opened
CVF-9	Minor	Suboptimal	Opened
CVF-10	Critical	Flaw	Opened
CVF-11	Minor	Suboptimal	Opened
CVF-12	Minor	Suboptimal	Opened
CVF-13	Minor	Suboptimal	Opened
CVF-14	Minor	Procedural	Opened
CVF-15	Minor	Documentation	Opened
CVF-16	Minor	Procedural	Opened
CVF-17	Minor	Documentation	Opened
CVF-18	Minor	Documentation	Opened
CVF-19	Minor	Documentation	Opened
CVF-20	Minor	Unclear behavior	Opened
CVF-21	Minor	Bad datatype	Opened
CVF-22	Minor	Suboptimal	Opened
CVF-23	Minor	Suboptimal	Opened
CVF-24	Minor	Bad naming	Opened
CVF-25	Minor	Suboptimal	Opened
CVF-26	Minor	Documentation	Opened
CVF-27	Critical	Flaw	Opened
CVF-28	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-29	Minor	Suboptimal	Opened
CVF-30	Minor	Suboptimal	Opened
CVF-31	Minor	Suboptimal	Opened
CVF-32	Minor	Suboptimal	Opened
CVF-33	Moderate	Suboptimal	Opened
CVF-34	Minor	Suboptimal	Opened
CVF-35	Major	Flaw	Opened
CVF-36	Minor	Suboptimal	Opened
CVF-37	Moderate	Flaw	Opened
CVF-38	Minor	Procedural	Opened
CVF-39	Minor	Procedural	Opened
CVF-40	Minor	Suboptimal	Opened
CVF-41	Minor	Suboptimal	Opened
CVF-42	Minor	Procedural	Opened
CVF-43	Minor	Procedural	Opened
CVF-44	Minor	Procedural	Opened
CVF-45	Minor	Suboptimal	Opened
CVF-46	Minor	Procedural	Opened
CVF-47	Minor	Flaw	Opened
CVF-48	Moderate	Suboptimal	Opened
CVF-49	Minor	Suboptimal	Opened
CVF-50	Minor	Suboptimal	Opened
CVF-51	Minor	Unclear behavior	Opened
CVF-52	Minor	Suboptimal	Opened
CVF-53	Minor	Procedural	Opened
CVF-54	Minor	Procedural	Opened
CVF-55	Critical	Flaw	Opened
CVF-56	Minor	Flaw	Opened
CVF-57	Major	Flaw	Opened
CVF-58	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-59	Moderate	Flaw	Opened
CVF-60	Minor	Flaw	Opened
CVF-61	Minor	Suboptimal	Opened
CVF-62	Minor	Bad naming	Opened
CVF-63	Minor	Documentation	Opened
CVF-64	Minor	Suboptimal	Opened
CVF-65	Minor	Bad datatype	Opened
CVF-66	Minor	Suboptimal	Opened
CVF-67	Minor	Readability	Opened
CVF-68	Minor	Procedural	Opened
CVF-69	Minor	Suboptimal	Opened
CVF-70	Minor	Bad datatype	Opened
CVF-71	Minor	Bad datatype	Opened
CVF-72	Minor	Bad datatype	Opened
CVF-73	Minor	Unclear behavior	Opened
CVF-74	Minor	Bad datatype	Opened
CVF-75	Minor	Suboptimal	Opened
CVF-76	Minor	Suboptimal	Opened
CVF-77	Minor	Bad datatype	Opened
CVF-78	Minor	Bad datatype	Opened
CVF-79	Minor	Suboptimal	Opened
CVF-80	Moderate	Procedural	Opened
CVF-81	Minor	Unclear behavior	Opened
CVF-82	Minor	Procedural	Opened
CVF-83	Minor	Suboptimal	Opened
CVF-84	Minor	Documentation	Opened
CVF-85	Minor	Suboptimal	Opened
CVF-86	Minor	Bad datatype	Opened
CVF-87	Minor	Suboptimal	Opened
CVF-88	Minor	Suboptimal	Opened

ID	Severity	Category	Status
CVF-89	Minor	Suboptimal	Opened
CVF-90	Minor	Suboptimal	Opened
CVF-91	Moderate	Flaw	Opened
CVF-92	Minor	Procedural	Opened
CVF-93	Minor	Procedural	Opened
CVF-94	Minor	Suboptimal	Opened
CVF-95	Minor	Readability	Opened
CVF-96	Minor	Suboptimal	Opened
CVF-97	Minor	Procedural	Opened
CVF-98	Minor	Procedural	Opened
CVF-99	Minor	Suboptimal	Opened
CVF-100	Minor	Procedural	Opened
CVF-101	Minor	Flaw	Opened
CVF-102	Minor	Documentation	Opened
CVF-103	Minor	Suboptimal	Opened
CVF-104	Minor	Suboptimal	Opened
CVF-105	Minor	Documentation	Opened
CVF-106	Minor	Suboptimal	Opened
CVF-107	Minor	Documentation	Opened
CVF-108	Minor	Documentation	Opened
CVF-109	Minor	Documentation	Opened
CVF-110	Minor	Procedural	Opened
CVF-111	Minor	Bad datatype	Opened
CVF-112	Minor	Procedural	Opened
CVF-113	Minor	Bad naming	Opened
CVF-114	Minor	Documentation	Opened
CVF-115	Minor	Procedural	Opened
CVF-116	Minor	Procedural	Opened
CVF-117	Minor	Procedural	Opened
CVF-118	Minor	Bad naming	Opened

ID	Severity	Category	Status
CVF-119	Minor	Suboptimal	Opened
CVF-120	Minor	Procedural	Opened

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	September 28, 2021	D. Khovratovich	Initial Draft
0.2	September 29, 2021	D. Khovratovich	Minor revision
1.0	September 29, 2021	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting 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. We have reviewed the contracts in the repository chainflip-eth-contracts (commit 2b6d46) and in the repository chainflip-eth-vesting-contracts (commit aebc23):

- TokenVesting.sol
- SchnorrSECP256K1.sol
- KeyManager.sol
- StakeManager.sol
- IERC20Lite.sol
- Vault.sol
- FLIP.sol
- IVault.sol
- DepositToken.sol
- DepositEth.sol
- Shared.sol
- IStakeManager.sol
- IKeyManager.sol
- IShared.sol

2.1 About ABDK

ABDK Consulting, 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.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best 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 and that 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 is 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 that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check 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.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** TokenVesting.sol

Description A more common syntax is "̂0.8.0".

Listing 1:

```
1 solidity ^0.8;
```

3.2 CVF-2

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** TokenVesting.sol

Recommendation The first parameter should be indexed.

Listing 2:

```
23 event TokensReleased(address token, uint256 amount);
```

3.3 CVF-3

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** TokenVesting.sol

Recommendation The parameter should be indexed.

Listing 3:

```
24 event TokenVestingRevoked(address token);
```

3.4 CVF-4

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** TokenVesting.sol

Recommendation The "_token" parameter should have type "IERC20".

Listing 4:

```
23 event TokensReleased(address token, uint256 amount);
   event TokenVestingRevoked(address token);
```

3.5 CVF-5

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenVesting.sol

Recommendation These variables can be made public so that explicit getters won't be needed.

Listing 5:

```
27 address private _beneficiary;

29 address private _revoker;

31 bool private _revocable;

34 uint private _start;
   uint private _cliff;
   uint private _end;

41 IStakeManager private _stakeManager;

43 mapping (address => uint256) private _released;
   mapping (address => bool) private _revoked;
```

3.6 CVF-6

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenVesting.sol

Recommendation These variables should be declared as immutable.

Listing 6:

```
27 address private _beneficiary;

29 address private _revoker;

31 bool private _revocable;

34 uint private _start;
   uint private _cliff;
   uint private _end;

39 bool private _canStake;

41 IStakeManager private _stakeManager;
```

3.7 CVF-8

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** TokenVesting.sol

Recommendation The first parameter should be IERC20.

Listing 7:

```
43 mapping (address => uint256) private _released;
   mapping (address => bool) private _revoked;
```

3.8 CVF-9

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenVesting.sol

Recommendation These checks can be done offline.

Listing 8:

```
69 require(beneficiary != address(0), "TokenVesting: beneficiary is
   ↪  the zero address");
70 require(revoker != address(0), "TokenVesting: revoker is the
   ↪ zero address");
   require(start > 0, "TokenVesting: start is 0");
   // solhint-disable-next-line max-line-length
   require(start < cliff, "TokenVesting: start isn't before cliff")
   ↪ ;
   require(cliff < end, "TokenVesting: cliff isn't before end");
   // solhint-disable-next-line max-line-length
   require(end > block.timestamp, "TokenVesting: final time is
   ↪ before current time");
   require(address(stakeManager) != address(0), "TokenVesting:
   ↪ beneficiary is the zero address");
```

3.9 CVF-10

- **Severity** Critical
- **Status** Opened
- **Category** Flaw
- **Source** TokenVesting.sol

Description In case the token is a ERC-777 token and the beneficiary is a contract, then the beneficiary contract may be called during the token transfer. As the state is undated after the call, a reentrancy attack.is possible, so the same tokens could be released (i.e. withdrawn) several times.
Recommendation Consider updating the state before transferring tokens.

Listing 9:

```
187 token.safeTransfer(_beneficiary, unreleased);

189 _released[address(token)] += unreleased;
```

3.10 CVF-11

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenVesting.sol

Recommendation It would be more efficient to implement the "_vestedAmount" function on top of the "_releasableAmount". Currently, the "_vestedAmount" function in some cases adds the released amount to the result, and the "_releasableAmount" function substructs this amount back. Consider refactoring.

Listing 10:

```
219 function _releasableAmount(IERC20 token) private view returns (
    ↪ uint) {

227 function _vestedAmount(IERC20 token) private view returns (uint)
    ↪ {
```

3.11 CVF-12

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenVesting.sol

Recommendation This should be calculated only if 'block.timestamp >= _cliff'.

Listing 11:

```
228 uint256 currentBalance = token.balanceOf(address(this));
    uint256 totalBalance = currentBalance + _released[address(token)
    ↪ ];
```

3.12 CVF-13

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** TokenVesting.sol

Recommendation The "_end - _start" value could be precomputed.

Listing 12:

```
236 return totalBalance * (block.timestamp - _start) / (_end -
    ↪ _start);
```

3.13 CVF-14

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** SchnorrSECP256K1.sol, KeyManager.sol, StakeManager.sol, IERC20Lite.sol, Vault.sol, FLIP.sol, IVault.sol, DepositToken.sol, DepositEth.sol, IStakeManager.sol, IKeyManager.sol, IShared.sol.

Recommendation Should be "̂0.8.0" according to a common best practice, unless there is something special about this particular version.

Listing 13:

```
1 solidity ^0.8.7;
```

3.14 CVF-15

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** SchnorrSECP256K1.sol

Recommendation This comment looks odd and should be removed.

Listing 14:

```
4 XXX: Do not use in production until this code has been audited.
```

3.15 CVF-16

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** SchnorrSECP256K1.sol

Description This abstract contract doesn't have any state.
Recommendation Consider turning it into a library.

Listing 15:

```
8 contract SchnorrSECP256K1 {
```

3.16 CVF-17

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** SchnorrSECP256K1.sol

Description Here not ‘x‘ but ‘d‘ is the private key.

Listing 16:

```
58 @dev 6. Let x be your secret key. Compute s = (k - d * e) % Q.
   ↪ Add Q to
```

3.17 CVF-18

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** SchnorrSECP256K1.sol

Description The value can't be negative if it is taken mod Q.

Listing 17:

```
58 @dev 6. Let x be your secret key. Compute s = (k - d * e) % Q.
   ↪ Add Q to
        it, if it's negative. This is your signature. (d is your
          ↪ secret
```

3.18 CVF-19

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** SchnorrSECP256K1.sol

Description The order of the function arguments in the documentation comment and in the code is different.
Recommendation Consider reordering the arguments in the comment.

Listing 18:

```
91    @param signingPubKeyX is the x ordinate of the public key.
        ↪ This must be
            less than HALF_Q.
      @param pubKeyYParity is 0 if the y ordinate of the public key
        ↪ is even, 1
            if it's odd.
      @param signature is the actual signature, described as s in
        ↪ the above
            instructions.
      @param msgHash is a 256-bit hash of the message being signed.
      @param nonceTimesGeneratorAddress is the ethereum address of k
        ↪ *g in the
            above instructions

104   uint256 msgHash,
      uint256 signature,
      uint256 signingPubKeyX,
      uint8 pubKeyYParity,
      address nonceTimesGeneratorAddress
```

3.19 CVF-20

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** SchnorrSECP256K1.sol

Description The "ecrecover" function returns zero address on error, so the good practice is to require or assert the returned address to be non-zero.
Recommendation Consider explicitly checking for zero address and either reverting or returning false in such case.

Listing 19:

```
139   address recoveredAddress = ecrecover(
```

3.20 CVF-21

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** SchnorrSECP256K1.sol

Description The values 27 and 28 should be turned into named constants.

Listing 20:

```
145 (pubKeyYParity == 0) ? 27 : 28,
```

3.21 CVF-22

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** KeyManager.sol

Recommendation If nonces are supposed to go in sequence, then a bit map of used nonces would be more efficient that a standard mapping of booleans, as the former will allow packing up to 256 nonces into a single word.

Listing 21:

```
25 mapping(KeyID => mapping(uint => bool)) private _keyToNoncesUsed
   ↪ ;
```

3.22 CVF-23

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** KeyManager.sol

Recommendation It would be cheaper to have three different events for three different change key use cases, as extra events cost nothing.

Listing 22:

```
28 event KeyChange(
```

3.23 CVF-24

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** KeyManager.sol

Recommendation This parameter should have type "KeyID" and should be named like "signedBy".

Listing 23:

```
29 bool signedByAggKey,
```

3.24 CVF-25

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** KeyManager.sol

Recommendation This parameter is redundant as it has been already logged.

Listing 24:

```
30 Key oldKey,
```

3.25 CVF-26

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** KeyManager.sol

Description This function can check each signature only once, which is not explicitly documented.
Recommendation Consider adding a comment and possibly change the function name to indicate the state change.

Listing 25:

```
49  * @notice  Checks the validity of a signature and msgHash, then
        ↪ updates _lastValidateTime

64 function isValidSig(
```

3.26 CVF-27

- **Severity** Critical
- **Status** Opened
- **Category** Flaw
- **Source** KeyManager.sol

Description As this function is public, anyone may call it with a valid signature but arbitrary keyID and sigData.nonce to "spend" this nonce for this keyID. Such calls could be used to frontrun valid calls to other function that require signatures, effectively preventing them from being executed.. Also, such calls could be used to move ahead the "_lastValidateTime" value effectively disabling the dead mean's switch.
Recommendation Consider making this function internal.

Listing 26:

```
68 ) public override returns (bool) {
```

3.27 CVF-28

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** KeyManager.sol

Description This requirement makes the ‘contractMsgHash‘ parameter redundant.

Listing 27:

```
72 require(sigData.msgHash == uint(contractMsgHash), "KeyManager:
    ↪ invalid msgHash");
```

3.28 CVF-29

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** KeyManager.sol

Description The expression "_keyToNoncesUsed[keyID]" is calcualted twice.
Recommendation Consider calculating once and reusing.

Listing 28:

```
83 require(!_keyToNoncesUsed[keyID][sigData.nonce], "KeyManager:
    ↪ nonce already used");

86 _keyToNoncesUsed[keyID][sigData.nonce] = true;
```

3.29 CVF-30

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** KeyManager.sol

Description The function always returns true.
Recommendation Consider returning nothing.

Listing 29:

```
88 return true;
```

3.30 CVF-31

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** KeyManager.sol

Description Using state-changing modifiers makes the code less readable.
Recommendation Consider putting the state changing functionality into the function body.

Listing 30:

```
102 ) external override nzKey(newKey) refundGas validSig(
```

3.31 CVF-32

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** KeyManager.sol

Description In case the "sigData" argument is known to always be the first argument of a function, it is possible to calculate the message hash in a generic way. Just get the whole "msg.data" as bytes array, replace the signature and hash bytes with zeros, and the calculate the hash of it.
Recommendation Consider calculating the message hash inside the "validSig" modifier or insdie the "isValidSig" function.

Listing 31:

```
104 keccak256(abi.encodeWithSelector(
        this.setAggKeyWithAggKey.selector,
        SigData(0, 0, sigData.nonce, address(0)),
        newKey
    )),

128 keccak256(abi.encodeWithSelector(
        this.setAggKeyWithGovKey.selector,
130     SigData(0, 0, sigData.nonce, address(0)),
        newKey
    )),

152 keccak256(abi.encodeWithSelector(
        this.setGovKeyWithGovKey.selector,
        SigData(0, 0, sigData.nonce, address(0)),
        newKey
    )),
```

3.32 CVF-33

- **Severity** Moderate
- **Status** Opened
- **Category** Suboptimal
- **Source** KeyManager.sol

Description It is checked that that no signature check has occurred within the time gap, even if a different key type was modified.
Recommendation Consider changing the update time only when this function is called or only when the agg key is modified.

Listing 32:

```
126 ) external override nzKey(newKey) validTime validSig(
```

3.33 CVF-34

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** KeyManager.sol

Description Out of the 'SigData' fields only the nonce is hashed.
Recommendation Consider explicitly hashing just the nonce,

Listing 33:

```
130 SigData(0, 0, sigData.nonce, address(0)),
```

3.34 CVF-35

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** KeyManager.sol

Recommendation These events should be different otherwise it is impossible to figure out which key was changed if they are the same.

Listing 34:

```
135 emit KeyChange(false, _keyIDToKey[KeyID.Agg], newKey);
159 emit KeyChange(false, _keyIDToKey[KeyID.Gov], newKey);
```

3.35 CVF-36

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** KeyManager.sol

Recommendation The brackets are redundant.

Listing 35:

```
176 return (_keyIDToKey[KeyID.Agg]);
184 return (_keyIDToKey[KeyID.Gov]);
```

3.36 CVF-37

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** KeyManager.sol

Description The contract address and chain ID are not signed, which makes replay attacks possible.
Recommendation Consider following ERC-712 (https://eips.ethereum.org/EIPS/eip-712) for the the signed message format.

Listing 36:

```
231 require(isValidSig(sigData, contractMsgHash, keyID));
```

3.37 CVF-38

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** StakeManager.sol

Recommendation These variables should be declared as immutable.

Listing 37:

```
27 IKeyManager private _keyManager;

29 FLIP private _FLIP;
```

3.38 CVF-39

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** StakeManager.sol

Description Hardcoding mainnet addresses in the code makes testing difficult.

Listing 38:

```
48 IERC1820Registry constant private _ERC1820_REGISTRY =
   ↪ IERC1820Registry(0
   ↪ x1820a4B7618BdE71Dce8cdc73aAB6C95905faD24);
```

3.39 CVF-40

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** StakeManager.sol

Recommendation Using a storage variable just to pass the address of the contract to another contract is clearly redundant.

Listing 39:

```
49 address[] private _defaultOperators;

80     _defaultOperators.push(address(this));

83     _FLIP = new FLIP("ChainFlip", "FLIP", _defaultOperators,
       ↪ address(this), flipTotalSupply);
```

3.40 CVF-41

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** StakeManager.sol

Recommendation This contract should be turned into a constructor argument, as it is not used outside the constructor.

Listing 40:

```
48 IERC1820Registry constant private _ERC1820_REGISTRY =
   ↪ IERC1820Registry(0
   ↪ x1820a4B7618BdE71Dce8cdc73aAB6C95905faD24);
```

3.41 CVF-42

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** StakeManager.sol

Recommendation The 'returnAddr' parameter should be indexed.

Listing 41:

```
64 event Staked(bytes32 indexed nodeID, uint amount, address
   ↪ returnAddr);
```

3.42 CVF-43

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** StakeManager.sol

Recommendation These events should be moved to the "IStakeManager" interface.

Listing 42:

```
64 event Staked(bytes32 indexed nodeID, uint amount, address
     ↪ returnAddr);
   event ClaimRegistered(

72 event ClaimExecuted(bytes32 indexed nodeID, uint amount);
   event FlipSupplyUpdated(uint oldSupply, uint newSupply, uint
     ↪ stateChainBlockNumber);
   event MinStakeChanged(uint oldMinStake, uint newMinStake);
```

3.43 CVF-44

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** StakeManager.sol

Recommendation This parameter should be indexed.

Listing 43:

```
68 address staker,
```

3.44 CVF-45

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** StakeManager.sol

Recommendation Emitting old values is redundant as they have been logged on previous events.

Listing 44:

```
73 event FlipSupplyUpdated(uint oldSupply, uint newSupply, uint
     ↪ stateChainBlockNumber);
   event MinStakeChanged(uint oldMinStake, uint newMinStake);
```

3.45 CVF-46

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** StakeManager.sol

Recommendation The result should be passed to the constructor instead of the other two.

Listing 45:

```
81 uint genesisValidatorFlip = numGenesisValidators * genesisStake;
```

3.46 CVF-47

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** StakeManager.sol

Description The returned value is ignored.
Recommendation Consider checking it for consistency.

Listing 46:

```
84 _FLIP.transfer(msg.sender, flipTotalSupply -
   ↪ genesisValidatorFlip);
```

3.47 CVF-48

- **Severity** Moderate
- **Status** Opened
- **Category** Suboptimal
- **Source** StakeManager.sol

Description Sending ERC-777 tokens from a contract address may cause the token origin contract to be invoked. Invoking untrusted contracts before updating the state is discouraged.
Recommendation Consider doing all the state updates before invoking any untrusted contracts.

Listing 47:

```
113 _FLIP.operatorSend(msg.sender, address(this), amount, "", "stake
    ↪ ");

116 _totalStake += amount;
```

3.48 CVF-49

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** StakeManager.sol

Recommendation This check is redundant, ans the FLIP code is fully trusted. Tricks like this are needed when dealing with untrusted tokens that could behave incorrectly.

Listing 48:

```
114 require(_FLIP.balanceOf(address(this)) == balBefore + amount, "
    ↪ StakeMan: token transfer failed");
```

3.49 CVF-50

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** StakeManager.sol

Recommendation This condition is redundant since 'operatorSend' should revert in the case of unsuccessful transfer

Listing 49:

```
114 require(_FLIP.balanceOf(address(this)) == balBefore + amount, "
    ↪ StakeMan: token transfer failed");
```

3.50 CVF-51

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** StakeManager.sol

Description It is suspicious that the function does not store any information about the staker and the amount in the contract.

Listing 50:

```
117 emit Staked(nodeID, amount, returnAddr);
```

3.51 CVF-52

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** StakeManager.sol

Description In case the "sigData" argument is known to always be the first argument of a function, it is possible to calculate the message hash in a generic way. Just get the whole "msg.data" as bytes array, replace the signature and hash bytes with zeros, and the calculate the hash of it.
Recommendation Consider calculating the message hash inside the "validSig" modifier or insdie the "isValidSig" function.

Listing 51:

```
140 keccak256(
        abi.encodeWithSelector(
            this.registerClaim.selector,
            SigData(0, 0, sigData.nonce, address(0)),
            nodeID,
            amount,
            staker,
            expiryTime
        )
    ),

206 keccak256(
        abi.encodeWithSelector(
            this.updateFlipSupply.selector,
            SigData(0, 0, sigData.nonce, address(0)),
210         newTotalSupply,
            stateChainBlockNumber
        )
    ),

241 keccak256(
        abi.encodeWithSelector(
            this.setMinStake.selector,
            SigData(0, 0, sigData.nonce, address(0)),
            newMinStake
        )
    ),
    KeyID.Gov
```

3.52 CVF-53

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** StakeManager.sol

Description It is odd that every node can file only one claim within its period.
Recommendation Consider allowing multiple claims.

Listing 52:

```
154 block.timestamp > uint(_pendingClaims[nodeID].expiryTime),
```

3.53 CVF-54

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** StakeManager.sol

Description No block height condition is present in the code.

Listing 53:

```
170 *           claim before 48h have passed after registering it, or
    ↪ after the specified
*           expiry block height
```

3.54 CVF-55

- **Severity** Critical
- **Status** Opened
- **Category** Flaw
- **Source** StakeManager.sol

Description When ERC-777 tokens are transferred to a contract, the recipient contract could be called. As the claim is deleted after the transfer, this makes a reentrancy attack possible and allows executing (i.e. withdrawing) the same claim several times.

Listing 54:

```
186 require(_FLIP.transfer(claim.staker, claim.amount));

189 delete _pendingClaims[nodeID];
```

3.55 CVF-56

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** StakeManager.sol

Description The function may be out of money to refund gas.
Recommendation Consider making it payable to enable updates without an extra ether transfer.

Listing 55:

```
204 ) external override nzUint(newTotalSupply) noFish refundGas
      ↪ validSig(
```

3.56 CVF-57

- **Severity** Major
- **Status** Opened
- **Category** Flaw
- **Source** StakeManager.sol

Description These calls change the token balance of the contract, but doesn't update the "_totalStake" value. This could cause the "noFish" modifier to revert.

Listing 56:

```
221 flip.burn(oldSupply - newTotalSupply, "");

223 flip.mint(address(this), newTotalSupply - oldSupply, "", "");
```

3.57 CVF-58

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** StakeManager.sol

Description This event is emitted even if nothing was changed.
Recommendation Consider emitting only if the new minimum stake value differs from the current value.

Listing 57:

```
250 emit MinStakeChanged(_minStake, newMinStake);
```

3.58 CVF-59

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** StakeManager.sol

Description As the contract address is not mixed into the message hash, it is possible to execute a signed transaction against wrong contract.
Recommendation Consider following ERC-712 (https://eips.ethereum.org/EIPS/eip-712) standard for the signed message format.

Listing 58:

```
334 require(_keyManager.isValidSig(sigData, contractMsgHash, keyID))
    ↪ ;
```

3.59 CVF-60

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** StakeManager.sol

Description Even without unsolicited token transfers (that could be prevented in the "tokensReceived" function by requiring _operator == this), the "updateFlipSupply" function could cause the "_totalSTake" value to become out of sync with the actual FLIP balance of the contract.

Listing 59:

```
342 // >= because someone could send some tokens to this contract
    ↪ and disable it if it was ==
```

3.60 CVF-61

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** IERC20Lite.sol

Recommendation Interfaces themselves don't produce any bytecode, so reducing them doesn't save gas.

Listing 60:

```
6 @notice   The interface for functions ERC20Lite implements. This
    ↪  is intended to
          be used with DepositEth so that there is as little
             ↪ code that goes into
          it as possible to reduce gas costs since it'll be
             ↪ deployed frequently
```

3.61 CVF-62

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** IERC20Lite.sol

Description Mixing the "uint256" type with its alias "uint" makes code harder to read
Recommendation Consider using consistent type naming.

Listing 61:

```
12 function transfer(address, uint256) external returns (bool);
   function balanceOf(address) external returns(uint);
```

3.62 CVF-63

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** IERC20Lite.sol

Recommendation Consider adding documentation comments to the functions to make the code easier to read..

Listing 62:

```
12 function transfer(address, uint256) external returns (bool);
   function balanceOf(address) external returns(uint);
```

3.63 CVF-64

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Vault.sol

Recommendation This variable should be declared as immutable.

Listing 63:

```
25 IKeyManager private _keyManager;
```

3.64 CVF-65

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Vault.sol

Recommendation This parameter should be indexed

Listing 64:

```
29 address payable recipient,
```

3.65 CVF-66

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Vault.sol

Description In case the "sigData" argument is known to always be the first argument of a function, it is possible to calculate the message hash in a generic way. Just get the whole "msg.data" as bytes array, replace the signature and hash bytes with zeros, and the calculate the hash of it.
Recommendation Consider calculating the message hash inside the "validSig" modifier or insdie the "isValidSig" function.

Listing 65:

```
68  keccak256(
        abi.encodeWithSelector(
70          this.allBatch.selector,
            SigData(0, 0, sigData.nonce, address(0)),
            fetchSwapIDs,
            fetchTokenAddrs,
            tranTokenAddrs,
            tranRecipients,
            tranAmounts
        )
    ),

126 keccak256(
        abi.encodeWithSelector(
            this.transfer.selector,
            SigData(0, 0, sigData.nonce, address(0)),
130         tokenAddr,
            recipient,
            amount
        )
    ),
    (...)
340 keccak256(
        abi.encodeWithSelector(
            this.fetchDepositTokenBatch.selector,
            SigData(0, 0, sigData.nonce, address(0)),
            swapIDs,
            tokenAddrs
        )
    ),
```

3.66 CVF-67

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** Vault.sol

Description The variable "i" is not initialized.
Recommendation Consider explicitly initializing with zero value for readability.

Listing 66:

```
91  for (uint i; i < fetchSwapIDs.length; i++) {

193 for (uint i; i < tokenAddrs.length; i++) {
```

3.67 CVF-68

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Vault.sol

Recommendation It is more common to use zero token address as a marker of plain ether.

Listing 67:

```
92      if (fetchTokenAddrs[i] == _ETH_ADDR) {

222 if (tokenAddr == _ETH_ADDR) {
```

3.68 CVF-69

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Vault.sol

Description The expression "fetchTokenAddres[i]" may be calculated twice.
Recommendation Consider calculating once and reusing.

Listing 68:

```
92  if (fetchTokenAddrs[i] == _ETH_ADDR) {

95      new DepositToken{salt: fetchSwapIDs[i]}(IERC20Lite(
          ↪ fetchTokenAddrs[i]));
```

3.69 CVF-70

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Vault.sol

Recommendation The type of this argument should be "IERC20".

Listing 69:

```
121 address tokenAddr,
```

3.70 CVF-71

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Vault.sol

Recommendation The type of this argument should be "IERC20 calldata []".

Listing 70:

```
154 address[] calldata tokenAddrs,
```

3.71 CVF-72

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Vault.sol

Recommendation The type of this argument should be "IERC20 calldata []".

Listing 71:

```
189 address[] calldata tokenAddrs,
```

3.72 CVF-73

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** Vault.sol

Recommendation Just use "recipient.send" or "recipient.call" instead of "recipient.transfer". The "send" function returns a boolean success flag and doesn't revert on error.

Listing 72:

```
199 * @notice  Annoyingly, doing `try addr.transfer` in `_transfer`
    ↪ fails because
200 *          Solidity doesn't see the `address` type as an
    ↪ external contract
*           and so doing try/catch on it won't work. Need to make
    ↪  it an external
*           call, and doing `this.something` counts as an
    ↪ external call, but that
*           means we need a fcn that just sends eth
```

3.73 CVF-74

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Vault.sol

Recommendation The type of this argument should be "IERC20".

Listing 73:

```
218 address tokenAddr,
```

3.74 CVF-75

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Vault.sol

Recommendation Just use "recipient.call" function. It returns a success flag and returned data and doesn't revert on errors.

Listing 74:

```
223 try this.sendEth{value: amount}(recipient) {
} catch (bytes memory lowLevelData) {
```

3.75 CVF-76

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Vault.sol

Description The "safeTransfer" function already wraps "require" around a plain "transfer" call, and also is able to handle non-compliant tokens that don't return any value.

Listing 75:

```
228 // It would be nice to wrap require around this line, but
    // some older tokens don't return a bool
```

3.76 CVF-77

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Vault.sol

Recommendation The type of this argument should be "IERC20".

Listing 76:

```
308 address tokenAddr
```

3.77 CVF-78

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** Vault.sol

Recommendation The type of this argument should be "IERC20 calldata []".

Listing 77:

```
337 address[] calldata tokenAddrs
```

3.78 CVF-79

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Vault.sol

Recommendation Replacing the modifier with its content would make the code much more readable.

Listing 78:

```
389 require(_keyManager.isValidSig(sigData, contractMsgHash, keyID))
    ↪ ;
```

3.79 CVF-80

- **Severity** Moderate
- **Status** Opened
- **Category** Procedural
- **Source** Vault.sol

Description As the contract address is not mixed into the message hash, it is possible to execute a signed transaction against wrong contract.
Recommendation Consider following ERC-712 (https://eips.ethereum.org/EIPS/eip-712) standard for the signed message format.

Listing 79:

```
389 require(_keyManager.isValidSig(sigData, contractMsgHash, keyID))
    ↪ ;
```

3.80 CVF-81

- **Severity** Minor
- **Status** Opened
- **Category** Unclear behavior
- **Source** FLIP.sol

Description On Ethereum, "ERC777" tokens are used much less often that "ERC20" tokens. Is it really necessary to support a more complicated and less popular API in a simple token contract?

Listing 80:

```
15 FLIP is ERC777, Ownable, Shared {
```

3.81 CVF-82

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** FLIP.sol

Description Here minting is restricted by modifiers whereas the constructor's minting is not.
Recommendation Consider using a consistent approach.

Listing 81:

```
32 ) external nzAddr(receiver) nzUint(amount) onlyOwner {
```

3.82 CVF-83

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** IVault.sol

Recommendation It is not necessary to inherit an interface in order to use type declared in it. Also, Solidity allows declaring enums and structs outside interfaces and contracts.

Listing 82:

```
13 IVault is IShared {
```

3.83 CVF-84

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** IVault.sol

Recommendation Consider adding documentation comments to the functions to make the code easier to read.

Listing 83:

```
15 function allBatch(

31 function transfer(

38 function transferBatch(

52 function fetchDepositEth(

57 function fetchDepositEthBatch(

62 function fetchDepositToken(

68 function fetchDepositTokenBatch(

81 function getKeyManager() external returns (IKeyManager);
```

3.84 CVF-85

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** IVault.sol

Recommendation A single array of structs with two fields would be more efficient than two parallel arrays, and also would make length check unnecessary.

Listing 84:

```
17 bytes32[] calldata fetchSwapIDs,
   address[] calldata fetchTokenAddrs,
```

3.85 CVF-86

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** IVault.sol

Recommendation The type of these arguments should be "IERC20 memory []".

Listing 85:

```
18 address[] calldata fetchTokenAddrs,
   address[] calldata tranTokenAddrs,
```

3.86 CVF-87

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** IVault.sol

Recommendation A single array of structs with three fields would be more efficient than three parallel arrays, and also would make length check unnecessary.

Listing 86:

```
19 address[] calldata tranTokenAddrs,
20 address payable[] calldata tranRecipients,
   uint[] calldata tranAmounts
```

3.87 CVF-88

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** IVault.sol

Recommendation A single array of structs with three fields would be more efficient than three parallel arrays, and also would make length check unnecessary.

Listing 87:

```
40 address[] calldata tokenAddrs,
   address payable[] calldata recipients,
   uint[] calldata amounts
```

3.88 CVF-89

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** IVault.sol

Recommendation A single array of structs with two fields would be more efficient than two parallel arrays, and also would make length check unnecessary.

Listing 88:

```
70 bytes32[] calldata swapIDs,
   address[] calldata tokenAddrs
```

3.89 CVF-90

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** DepositToken.sol

Recommendation This constructor should probably emit some event to make it easier to track when deposits are processed.

Listing 89:

```
15 constructor(IERC20Lite token) {
```

3.90 CVF-91

- **Severity** Moderate
- **Status** Opened
- **Category** Flaw
- **Source** DepositToken.sol

Description The returned value is ignored.

Listing 90:

```
19 token.transfer(msg.sender, token.balanceOf(address(this)));
```

3.91 CVF-92

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** DepositEth.sol

Recommendation This constructor should probably emit some event to make it easier to track when deposits are processed.

Listing 91:

```
12 constructor() {
```

3.92 CVF-93

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Shared.sol

Description This abstract contract doesn't have any state.
Recommendation Consider turning it into a library.

Listing 92:

```
13 contract Shared is IShared {
```

3.93 CVF-94

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Shared.sol

Description The contract uses both uint and uint256 types interchangeably.
Recommendation Consider using only one type.

Listing 93:

```
18 uint constant internal _E_18 = 10**18;
```

3.94 CVF-95

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** Shared.sol

Recommendation This value could be rendered as "1e18".

Listing 94:

```
18 uint constant internal _E_18 = 10**18;
```

3.95 CVF-96

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Shared.sol

Recommendation The first parameter should be indexed.

Listing 95:

```
21 event RefundFailed(address to, uint256 amount, uint256
   ↪ currentBalance);
```

3.96 CVF-97

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Shared.sol

Recommendation "isn't nonzero" means "is zero". Should be "isn't zero" or "is nonzero".

Listing 96:

```
24 /// @dev     Checks that a uint isn't nonzero/empty
30 /// @dev     Checks that an address isn't nonzero/empty
36 /// @dev     Checks that a bytes32 isn't nonzero/empty
```

3.97 CVF-98

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Shared.sol

Description Contrary to the comment, only one field is checked.

Listing 97:

```
42 /// @dev     Checks that all of a Key's values are populated
```

3.98 CVF-99

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Shared.sol

Recommendation It would be cheaper to check the ether balance before trying to refund, rather than to catch a failed refund.

Listing 98:

```
55 try this.refundEth(msg.sender, refundWei) {
```

3.99 CVF-100

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** Shared.sol

Recommendation It is inconsistent that in one case one parameter is logged and in the other three ones are.

Listing 99:

```
56 emit Refunded(refundWei);

59 emit RefundFailed(msg.sender, refundWei, address(this).balance);
```

3.100 CVF-101

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Shared.sol

Recommendation The gas is paid by "tx.origin" rather than "msg.sender', so consider refunding to "tx.origin".

Listing 100:

```
55 try this.refundEth(msg.sender, refundWei) {
```

3.101 CVF-102

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** Shared.sol

Description This comment is confusing. The "transfer" function uses gas limit of 2300 rather than zero, and actually doesn't necessary fail when sending to a smart contract.

Listing 101:

```
66 // Send 0 gas so that if we go to a contract it fails
```

3.102 CVF-103

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Shared.sol

Description Usage of "transfer" is discouraged as it implicitly relies of gas cost of particular operations.
Recommendation Consider using "send" instead.

Listing 102:

```
67 payable(to).transfer(amount);
```

3.103 CVF-104

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** IStakeManager.sol

Description This interface looks like a subset of the "IStakeManager" interface defined in the other repository, with only difference, that the "nodeID" type is "uint" here and "bytes32" there.
Recommendation Consider merging these two interfaces into one and using the same type for node IDs everywhere.

Listing 103:

```
4 IStakeManager {
```

3.104 CVF-105

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** IStakeManager.sol

Description The order of function arguments is different in the documentation comment and in the function declaration.
Recommendation Consider reordering arguments in the documentation comment.

Listing 104:

```
 9   * @param amount     The amount of stake to be locked up
10   * @param nodeID     The nodeID of the staker
     * @param claimAddr      The address which, if specified, forces
       ↪ the State Chain

14 function stake(uint nodeID, uint amount, address claimAddr)
     ↪ external;
```

3.105 CVF-106

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** IStakeManager.sol

Recommendation It is not necessary to inherit an interface in order to use type declared in it. Also, Solidity allows declaring enums and structs outside interfaces and contracts.

Listing 105:

```
12 IStakeManager is IShared {
```

3.106 CVF-107

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** IStakeManager.sol

Description The order of function arguments is different in the documentation comment and in the function declaration.
Recommendation Consider reordering arguments in the documentation comment.

Listing 106:

```
26 * @param amount     The amount of stake to be locked up
   * @param nodeID     The nodeID of the staker
   * @param returnAddr     The address which the staker requires to
     ↪ be used

32     bytes32 nodeID,
       uint amount,
       address returnAddr
```

3.107 CVF-108

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** IStakeManager.sol

Recommendation The "it" should be removed.

Listing 107:

```
71 * @notice  Compares a given new FLIP supply it against the old
     ↪ supply,
```

3.108 CVF-109

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** IStakeManager.sol

Description It is unclear who gets the minted tokens and whose tokens are burned.
Recommendation Consider explaining.

Listing 108:

```
72 *          then mints and burns as appropriate
```

3.109 CVF-110

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** IStakeManager.sol

Recommendation This function should emit some event, and such event should be defined in this interface.

Listing 109:

```
91 function setMinStake(
```

3.110 CVF-111

- **Severity** Minor
- **Status** Opened
- **Category** Bad datatype
- **Source** IStakeManager.sol

Recommendation The return type for this function should be "FLIP" of some interface extracted from the "FLIP" smart contract.

Listing 110:

```
112 function getFLIPAddress() external view returns (address);
```

3.111 CVF-112

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** IKeyManager.sol

Recommendation It is not necessary to inherit an interface in order to use type declared in it. Also, Solidity allows declaring enums and structs outside interfaces and contracts.

Listing 111:

```
12 IKeyManager is IShared {
```

3.112 CVF-113

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** IKeyManager.sol

Description The function name looks like a simple checker without any side effects, while actually the function modifies the blockchain state.
Recommendation Consider renaming.

Listing 112:

```
20 function isValidSig(
```

3.113 CVF-114

- **Severity** Minor
- **Status** Opened
- **Category** Documentation
- **Source** IKeyManager.sol

Recommendation Consider adding documentation comments to the functions to make the code easier to read.

Listing 113:

```
20 function isValidSig(

26 function setAggKeyWithAggKey(

31 function setAggKeyWithGovKey(

36 function setGovKeyWithGovKey(

48 function getAggregateKey() external view returns (Key memory);

50 function getGovernanceKey() external view returns (Key memory);

52 function getLastValidateTime() external view returns (uint);

54 function isNonceUsedByKey(KeyID keyID, uint nonce) external view
   ↪ returns (bool);
```

3.114 CVF-115

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** IShared.sol

Description Solidity allows declaring enums and structs outside interfaces and contracts, so this interface is redundant.
Recommendation Just declare shared types on top level in a separate file and import this file where necessary.

Listing 114:

```
5 @title    Shared interface
  @notice   Holds structs needed by other interfaces
```

3.115 CVF-116

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** IShared.sol

Recommendation A good practice is to name enum constants IN_ALL_CAPITALS.

Listing 115:

```
18 Agg,
Gov
```

3.116 CVF-117

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** IShared.sol

Recommendation It would be more correct to say that Key is a compressed form of a public key.

Listing 116:

```
23 * @dev  SchnorrSECP256K1 requires that each key has a public key
   ↪  part (x coordinate),
*       a parity for the y coordinate (0 if the y ordinate of
   ↪ the public key is even, 1
*       if it's odd)
*/
```

3.117 CVF-118

- **Severity** Minor
- **Status** Opened
- **Category** Bad naming
- **Source** IShared.sol

Recommendation Consider defining named constants for the 'y' parity values.

Listing 117:

```
24 *       a parity for the y coordinate (0 if the y ordinate of
   ↪ the public key is even, 1
*       if it's odd)
```

3.118 CVF-119

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** IShared.sol

Recommendation This format is not more efficient than just the two coordinates since uint8 is stored as a 256-bit value in Solidity.

Listing 118:

```
27 struct Key {
       uint pubKeyX;
       uint8 pubKeyYParity;
30 }
```

3.119 CVF-120

- **Severity** Minor
- **Status** Opened
- **Category** Procedural
- **Source** IShared.sol

Recommendation A common practice is to use the "bytes32" type for hashes, rather than "uint".

Listing 119:

```
37 uint msgHash;
```