ABDK CONSULTING

SMART CONTRACT AUDIT

ChainFlip

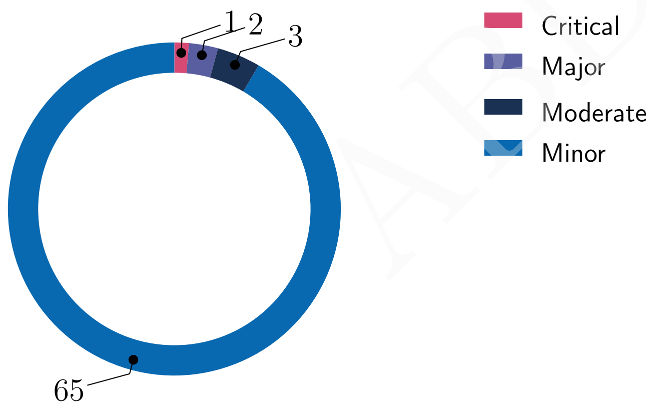
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov, Dmitry Khovratovich and Vladimir Smelov
31st August 2022

We've been asked to review updates to 19 files in a GitHub repo. We found 1 critical, 2 major, and a few less important issues. All identified critical and major issues have been fixed or otherwise addressed in collaboration with the client.



Findings

ID	Severity	Category	Status
CVF-1	Moderate	Flaw	Info
CVF-2	Minor	Unclear behavior	Fixed
CVF-3	Moderate	Suboptimal	Info
CVF-4	Minor	Documentation	Fixed
CVF-5	Minor	Procedural	Info
CVF-6	Minor	Suboptimal	Info
CVF-7	Minor	Suboptimal	Info
CVF-8	Minor	Documentation	Info
CVF-9	Minor	Bad naming	Fixed
CVF-10	Major	Suboptimal	Fixed
CVF-11	Minor	Suboptimal	Info
CVF-12	Minor	Procedural	Info
CVF-13	Minor	Unclear behavior	Fixed
CVF-14	Minor	Bad naming	Info
CVF-15	Minor	Documentation	Info
CVF-16	Minor	Suboptimal	Fixed
CVF-17	Minor	Unclear behavior	Info
CVF-18	Minor	Suboptimal	Info
CVF-19	Minor	Suboptimal	Info
CVF-20	Minor	Suboptimal	Fixed
CVF-21	Minor	Suboptimal	Info
CVF-22	Minor	Suboptimal	Info
CVF-23	Minor	Suboptimal	Fixed
CVF-24	Minor	Bad datatype	Info
CVF-25	Minor	Unclear behavior	Fixed
CVF-26	Minor	Bad naming	Fixed
CVF-27	Minor	Bad naming	Info

ID	Severity	Category	Status
CVF-28	Minor	Suboptimal	Info
CVF-29	Moderate	Procedural	Fixed
CVF-30	Minor	Suboptimal	Info
CVF-31	Minor	Suboptimal	Fixed
CVF-32	Minor	Suboptimal	Info
CVF-33	Major	Flaw	Fixed
CVF-34	Minor	Unclear behavior	Info
CVF-35	Minor	Procedural	Fixed
CVF-36	Minor	Suboptimal	Fixed
CVF-37	Minor	Unclear behavior	Fixed
CVF-38	Minor	Unclear behavior	Info
CVF-39	Minor	Suboptimal	Fixed
CVF-40	Minor	Documentation	Fixed
CVF-41	Minor	Unclear behavior	Fixed
CVF-42	Minor	Suboptimal	Info
CVF-43	Minor	Suboptimal	Info
CVF-44	Critical	Flaw	Info
CVF-45	Minor	Procedural	Info
CVF-46	Minor	Unclear behavior	Info
CVF-47	Minor	Suboptimal	Fixed
CVF-48	Minor	Suboptimal	Info
CVF-49	Minor	Procedural	Info
CVF-50	Minor	Suboptimal	Fixed
CVF-51	Minor	Suboptimal	Fixed
CVF-52	Minor	Procedural	Info
CVF-53	Minor	Procedural	Info
CVF-54	Minor	Documentation	Fixed
CVF-55	Minor	Suboptimal	Fixed
CVF-56	Minor	Bad naming	Fixed
CVF-57	Minor	Bad naming	Fixed

ID	Severity	Category	Status
CVF-58	Minor	Bad naming	Fixed
CVF-59	Minor	Procedural	Info
CVF-60	Minor	Suboptimal	Fixed
CVF-61	Minor	Procedural	Info
CVF-62	Minor	Suboptimal	Fixed
CVF-63	Minor	Procedural	Fixed
CVF-64	Minor	Suboptimal	Fixed
CVF-65	Minor	Procedural	Fixed
CVF-66	Minor	Suboptimal	Info
CVF-67	Minor	Suboptimal	Info
CVF-68	Minor	Suboptimal	Fixed
CVF-69	Minor	Unclear behavior	Fixed
CVF-70	Minor	Bad naming	Info
CVF-71	Minor	Suboptimal	Fixed

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	July 12, 2022	D. Khovratovich	Initial Draft
0.2	July 13, 2022	D. Khovratovich	Minor revision
1.0	August 1, 2022	D. Khovratovich	Release
1.1	August 30, 2022	D. Khovratovich	CVF-29,33 Severity downgraded
1.2	August 30, 2022	D. Khovratovich	CVF-38 Removed
2.0	August 31, 2022	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 following contracts in the chainflip-eth-contracts repository:

- abstract/SchnorrSECP256K1.sol
- abstract/Shared.sol
- interfaces/IAggKeyNonceConsumer.sol
- interfaces/IERC20Lite.sol
- interfaces/IFLIP.sol
- interfaces/IGovernanceCommunityGuarded.sol
- interfaces/IKeyManager.sol
- interfaces/IShared.sol
- interfaces/IStakeManager.sol
- interfaces/IVault.sol
- AggKeyNonceConsumer.sol
- DepositEth.sol
- DepositToken.sol
- FLIP.sol
- GovernanceCommunityGuarded.sol
- KeyManager.sol
- StakeManager.sol
- TokenVesting.sol
- Vault.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** Moderate
- **Category** Flaw
- **Status** Info
- **Source** KeyManager.sol

Description This function is callable by anyone.
Recommendation Consider restricting access to it.
Client Comment It is fine as is – we will make sure to call this after deploying the contract.

Listing 1:

```
+function setCanConsumeKeyNonce(address[] calldata addrs)
    ↪ external {
```

3.2 CVF-2

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Fixed
- **Source** KeyManager.sol

Description These functions should emit some events.

Listing 2:

```
+function setCanConsumeKeyNonce(address[] calldata addrs)
    ↪ external {

+function updateCanConsumeKeyNonce(
```

3.3 CVF-3

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

Description In case the "addrs"/"newAddres" would be required to be sorted, this check could be replaced by a check that the previous element is less than the current element, which would be more efficient than the current approach.
Client Comment It is fine as is.

Listing 3:

```
+require(!_canConsumeKeyNonce[addrs[i]], "KeyManager: address
    ↪ already whitelisted");

+require(!_canConsumeKeyNonce[newAddrs[i]], "KeyManager: address
    ↪  already whitelisted");
```

3.4 CVF-4

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

Description This comment is misleading as one could thing that this function could delist as many addresses as the caller wants, while actually the only option is to delist all the currently listed addresses.
Recommendation Consider emphasizing this fact.
Client Comment Comment updated.

Listing 4:

```
+* @notice  Replaces the specific addresses that can call
  ↪ consumeKeyNonce. To be used if
+*          contracts are updated. Can delist addresses and can
  ↪ add an arbitrary number of new addresses.
```

3.5 CVF-5

- **Severity** Minor
- **Category** Procedural
- **Status** Info
- **Source** KeyManager.sol

Description Here a custom format of a signed message is used.
Recommendation Consider using a standard format as described in EIP-712: https://eips.ethereum.org/EIPS/eip-712
Client Comment It is fine as is.

Listing 5:

```
+keccak256(
+    abi.encodeWithSelector(
+        this.updateCanConsumeKeyNonce.selector,
+        SigData(sigData.keyManAddr, sigData.chainID, 0, 0,
   ↪ sigData.nonce, address(0)),
+        currentAddrs,
+        newAddrs
+    )
+)

+keccak256(
+    abi.encodeWithSelector(
+        this.setAggKeyWithAggKey.selector,
+        SigData(sigData.keyManAddr, sigData.chainID, 0, 0,
   ↪ sigData.nonce, address(0)),
+        newAggKey
+    )
+)

+keccak256(
+    abi.encodeWithSelector(
+        this.setGovKeyWithAggKey.selector,
+        SigData(sigData.keyManAddr, sigData.chainID, 0, 0,
   ↪ sigData.nonce, address(0)),
+        newGovKey
+    )
+)

+keccak256(
+    abi.encodeWithSelector(
+        this.setCommKeyWithAggKey.selector,
+        SigData(sigData.keyManAddr, sigData.chainID, 0, 0,
   ↪ sigData.nonce, address(0)),
+        newCommKey
+    )
+)
```

3.6 CVF-6

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

Description This check makes the “contractMsgHash” argument redundant.
Recommendation Consider removing this argument.
Client Comment We will consider that.

Listing 6:

```
+require(sigData.msgHash == uint256(contractMsgHash), "
    ↪ KeyManager: invalid msgHash");
```

3.7 CVF-7

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

Description These events are emitted even if nothing actually changed.
Client Comment It is fine as it is.

Listing 7:

```
+emit AggKeySetByAggKey(_aggKey, newAggKey);

+emit AggKeySetByGovKey(_aggKey, newAggKey);

+emit GovKeySetByAggKey(_govKey, newGovKey);

+emit GovKeySetByGovKey(_govKey, newGovKey);

+emit CommKeySetByAggKey(_commKey, newCommKey);

+emit CommKeySetByCommKey(_commKey, newCommKey);
```

3.8 CVF-8

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

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.
Client Comment There is a comment on top of the function so it is fine as is.

Listing 8:

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

3.9 CVF-9

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

Description The names of these modifiers look like names of getter functions.
Recommendation More conventional names would be "onlyGovernor" and "onlyCommunityKey".

Listing 9:

```
+modifier isGovernor() {

+modifier isCommunityKey() {
```

3.10 CVF-10

- **Severity** Major
- **Category** Suboptimal
- **Status** Fixed
- **Source** KeyManager.sol

Description This looks like waste of gas.
Recommendation Consider just creating an internal version of the "consumeKeyNonce" function that doesn't perform the whitelist check and use it here. The original "consumeKeyNonce" function could do the whitelist check and then delegate to this internal version.

Listing 10:

```
+// Need to make this an external call so that the msg.sender is
   ↪ the
+// address of this contract, otherwise calling a function with
   ↪ this
+// modifier from any address would fail the whitelist check
```

3.11 CVF-11

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** SchnorrSECP256K1.sol

Description String error messages are suboptimal.
Recommendation Consider using named errors.
Client Comment It is fine as it is.

Listing 11:

```
172 +require(signingPubKeyX < HALF_Q, "Public-key x >= HALF_Q");

174 +require(signature < Q, "Sig must be reduced modulo Q");

184 +    "No zero inputs allowed"

210 +require(recoveredAddress != address(0), "Schnorr: recoveredAddress is 0");

216 +require(signingPubKeyX < HALF_Q, "Public-key x >= HALF_Q");
```

3.12 CVF-12

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

Description This function is not used in this contract.
Recommendation Consider moving it to where it is used.
Client Comment It is fine as it is.

Listing 12:

```
215 +function verifySigningKeyX(uint256 signingPubKeyX) public pure {
```

3.13 CVF-13

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

Description This function doesn't need to be public.
Recommendation Consider declaring as internal.

Listing 13:

```
215 +function verifySigningKeyX(uint256 signingPubKeyX) public pure {
```

3.14 CVF-14

- **Severity** Minor
- **Category** Bad naming
- **Status** Info
- **Source** Vault.sol

Recommendation Events are usually named via nouns, such as "TransferFailure".
Client Comment It is fine as it is.

Listing 14:

```
36 +event TransferFailed(address payable indexed recipient, uint256
    ↪ amount, bytes lowLevelData);
```

3.15 CVF-15

- **Severity** Minor
- **Category** Documentation
- **Status** Info
- **Source** Vault.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.
Client Comment It is fine as it is.

Listing 15:

```
47 +constructor(IKeyManager keyManager) AggKeyNonceConsumer(
    ↪ keyManager) {}

716  receive() external payable {}
```

3.16 CVF-16

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

Recommendation It would be more efficient to pass a single array of structs with four fields rather than four parallel arrays. This would also make the length checks unnecessary.

Listing 16:

```
96   bytes32[] calldata fetchSwapIDs,

99  +IERC20[] calldata fetchTokens,
100 +IERC20[] calldata tranTokens,
     address payable[] calldata tranRecipients,
```

3.17 CVF-17

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

Description Here a custom format of a signed message is used.
Recommendation Consider using a standard format as described in EIP-712: https://eips.ethereum.org/EIPS/eip-712
Client Comment It is fine as it is.

Listing 17:

```
121 +keccak256(
    +     abi.encodeWithSelector(
    +         this.allBatch.selector,
    +         SigData(sigData.keyManAddr, sigData.chainID, 0, 0,
        ↪ sigData.nonce, address(0)),
    +         fetchSwapIDs,
    +         fetchTokens,
    +         tranTokens,
    +         tranRecipients,
    +         tranAmounts
130 +     )
      )

212 +keccak256(
    +     abi.encodeWithSelector(
    +         this.transfer.selector,
    +         SigData(sigData.keyManAddr, sigData.chainID, 0, 0,
        ↪ sigData.nonce, address(0)),
    +         token,
    +         recipient,
    +         amount
    +     )
220   )

266 +keccak256(
    +     abi.encodeWithSelector(
    +         this.transferBatch.selector,
    +         SigData(sigData.keyManAddr, sigData.chainID, 0, 0,
        ↪ sigData.nonce, address(0)),
270 +         tokens,
    +         recipients,
    +         amounts
    +     )
      )
```

(..., 395, 435, 489, 536)

3.18 CVF-18

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

Description Explicit conversion from "IERC20" to "IERC20Lite" looks weird.
Recommendation Consider refactoring to avoid such conversions.
Client Comment It is fine as it is.

Listing 18:

```
+          new DepositToken{salt: fetchSwapIDs[i]}(IERC20Lite(address(fetchTokens[i])));

+new DepositToken{salt: swapID}(IERC20Lite(address(token)));

+     new DepositToken{salt: swapIDs[i]}(IERC20Lite(address(tokens[i])));
```

3.19 CVF-19

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

Description Does this really save gas?
Recommendation Consider incrementing the loop counter inside "for" loop construct for readability.
Client Comment Yes, it saves gas since it skips the overflow math check.

Listing 19:

```
+unchecked {
+     ++i;
 }

+unchecked {
+     ++i;
+}

+unchecked {
+     ++i;
+}

+unchecked {
+     ++i;
+}
```

3.20 CVF-20

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

Recommendation It would be more efficient to use a single array of structs with three fields, rather than three parallel arrays. This would also make the length checks unnecessary.

Listing 20:

```
+IERC20[] calldata tokens,
 address payable[] calldata recipients,

+uint256[] calldata amounts
```

3.21 CVF-21

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

Recommendation The conversion would be necessary if the type of "_ETH_ADDR" would be "IERC20".
Client Comment It is fine as it is.

Listing 21:

```
+if (address(token) == _ETH_ADDR) {
```

3.22 CVF-22

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

Description This performs two external calls in order to send ether and capture possible error. This looks like waste of gas.
Recommendation Consider just doing: recipient.send{value: amount}(""); and processing the returned values.
Client Comment Downside of using send is that we don't get the lowLevelData back so it will be difficult to figure out why the transaction failed. We will consider whether the gas decrease is worth not getting the lowleveldata.

Listing 22:

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

3.23 CVF-23

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

Recommendation It would be more efficient to pass a single array of structs with two fields rather than two parallel arrays. This would also make the length check unnecessary.

Listing 23:

```
 bytes32[] calldata swapIDs,

+IERC20[] calldata tokens
```

3.24 CVF-24

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

Recommendation The type of this argument should be "IERC20".
Client Comment It is fine as it is.

Listing 24:

```
+address ingressToken,
```

3.25 CVF-25

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

Description These functions should emit some events.

Listing 25:

```
+function enableSwaps() external override isGovernor
    ↪ swapsDisabled {

+function disableSwaps() external override isGovernor
    ↪ swapsEnabled {
```

3.26 CVF-26

- **Severity** Minor
- **Category** Bad naming
- **Status** Fixed
- **Source** Vault.sol

Description The name is too generic and doesn't give a clue regarding what exactly the time is valid for.
Recommendation Consider renaming.

Listing 26:

```
682 +modifier validTime() {
```

3.27 CVF-27

- **Severity** Minor
- **Category** Bad naming
- **Status** Info
- **Source** TokenVesting.sol

Recommendation Events are usually named via nouns, such as "Vesting", "VestingRevocation".
Client Comment It is fine as it is.

Listing 27:

```
21 +event TokensReleased(IERC20 indexed token, uint256 amount);
   +event TokenVestingRevoked(IERC20 indexed token);
```

3.28 CVF-28

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

Description This flag is redundant.
Recommendation Just set "revoker" to a zero address to effectively disable revocation.
Client Comment It is fine as it is.

Listing 28:

```
31 +bool public immutable revocable;
```

3.29 CVF-29

- **Severity** Moderate
- **Category** Procedural
- **Status** Fixed
- **Source** TokenVesting.sol

Description This parameter is never used.
Recommendation Consider removing it.

Listing 29:

```
+uint256 public immutable start;
```

3.30 CVF-30

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

Description This flag is redundant.
Recommendation Just use a zero "stakeManager" address as as indicator of disabled staking.
Client Comment True. But we leave this variable for readability - we don't care too much about gas here.

Listing 30:

```
+bool public immutable canStake;
```

3.31 CVF-31

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

Description This flag is redundant. A zero "revoker_" value could be used as an indicator that revocation is disabled.

Listing 31:

```
+bool revocable_,
```

3.32 CVF-32

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

Description This flag is redundant.
Recommendation Just use a zero "stakeManager_" address as as indicator of disabled staking.
Client Comment True. But we leave this variable for readability – we don't care too much about gas here.

Listing 32:

```
+bool canStake_,
```

3.33 CVF-33

- **Severity** Major
- **Category** Flaw
- **Status** Fixed
- **Source** TokenVesting.sol

Description If FLIP is not yet set for the stack manager, one may create a valid stake without actually spencing any tokens,.
Recommendation Consider explicitly requiring that FLIP is set and also checking the value returned from the "approve" call.
Client Comment Fixed by checking that FLIP is different than address 0 in the stake function.

Listing 33:

```
+stakeManager.getFLIP().approve(address(stakeManager), amount);
+stakeManager.stake(nodeID, amount, address(this));
```

3.34 CVF-34

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** TokenVesting.sol

Description This should be called only when "refund" is non-zero.
Client Comment It is fine as it is.

Listing 34:

```
+token.safeTransfer(revoker, refund);
```

3.35 CVF-35

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

Recommendation This event should include the refund amount as a parameter.

Listing 35:

```
+emit TokenVestingRevoked(token);
```

3.36 CVF-36

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

Description This check is redundant, as it is superseded by the next check.
Client Comment Fixed by revoker being address(0) when it's not revocable - the msg.sender==revoker check should take care of that. Then removed this line.

Listing 36:

```
+require(revocable, "Vesting: cannot revoke");
```

3.37 CVF-37

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Fixed
- **Source** TokenVesting.sol

Description These values should be calculated only if block.timestamp < cliff;
Client Comment FIxed by first returning 0 if block.timestamp<cliff. If not, then making the calculation.

Listing 37:

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

3.38 CVF-38

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** TokenVesting.sol

Description This formula ignores the start time. Usually, the cliff amount is calculated as: totalAmount * (cliff - start) / (end - start).
Client Comment This is because we want the cliff to be 20% of the total(initial) balance and then unlock linearly for the rest of the vesting time (cliff to end period). Doing like the suggested formula means that we cannot control the amount released on the cliff unless we hardcode the start variable relative to the cliff and end (which doesn't seem like a good alternative).

Listing 38:

```
195 +uint256 cliffAmount = totalBalance / CLIFF_DENOMINATOR;
```

3.39 CVF-39

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

Description There is no "updateFLIP" function.
Recommendation Should probably be "setFlip" instead.

Listing 39:

```
47 +/// @dev    The FLIP token. Initial value to be set using
     ↪ updateFLIP
```

3.40 CVF-40

- **Severity** Minor
- **Category** Documentation
- **Status** Fixed
- **Source** StakeManager.sol

Description This comment is confusing. The word "initial" could make one thing that the "_FILP" value could be updated, which is not the case.
Recommendation Consider rephrasing.

Listing 40:

```
47 +/// @dev    The FLIP token. Initial value to be set using
     ↪ updateFLIP
```

3.41 CVF-41

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

Description This function should emit some event.

Listing 41:

```
144 +function setFlip(FLIP flip) external onlyDeployer nzAddr(
      ↪ address(flip)) {
```

3.42 CVF-42

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

Recommendation As zero "_FLIP" value has a special meaning, consider explicitly requiring "flip" to be non-zero.
Client Comment There is a nzAddrs modifier to check that flip is non-zero. Also, In the stake function I have added a check to make sure that _FLIP is non-zero, as written in issue #34.

Listing 42:

```
145 +require(address(_FLIP) == address(0), "Staking: Flip address
      ↪ already set");
```

3.43 CVF-43

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

Description String error messages are suboptimal.
Recommendation Consider using named errors instead.
Client Comment It is fine as it is.

Listing 43:

```
145 +require(address(_FLIP) == address(0), "Staking: Flip address
      ↪ already set");

174 +require(amount >= _minStake, "Staking: stake too small");

244 +require(expiryTime > startTime, "Staking: expiry time too soon
      ↪ ");

270 +     "Staking: early, late, or execd"

451 +require(msg.sender == deployer, "Staking: not deployer");
```

3.44 CVF-44

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

Description By calling this function before “setFlip” was called, one may create a valid stake without actually spending any tokens.
Recommendation Consider explicitly requiring “_FLIP” to be non-zero and checking the value returned from “transferFrom”.
Client Comment It is fine as it is.

Listing 44:

```
176 +_FLIP.transferFrom(msg.sender, address(this), amount);
```

3.45 CVF-45

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

Description Here a custom format of a signed message is used.
Recommendation Consider using a standard format as described in EIP-712: https://eips.ethereum.org/EIPS/eip-712
Client Comment We are OK with that.

Listing 45:

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

3.46 CVF-46

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

Description This condition prohibits 0-second validity periods but permits 1-second validity period. Does this make sense? Should there be a nontrivial minimum validity time?
Client Comment The claim registery is performed by the statechain so there can be checks there. No need to add more checks here - it is fine as is.

Listing 46:

```
+require(expiryTime > startTime, "Staking: expiry time too soon
  ↪ ");
```

3.47 CVF-47

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

Description Conversions to "uint256" are redundant as "block.timestamp" is already "uint256".

Listing 47:

```
+uint256(block.timestamp) >= claim.startTime && uint256(block.
    ↪ timestamp) <= claim.expiryTime,
```

3.48 CVF-48

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

Description This event is emitted even if nothing actually changed.
Client Comment It is fine as it is.

Listing 48:

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

3.49 CVF-49

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

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.
Client Comment There is a comment on top of the function so it is fine.

Listing 49:

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

3.50 CVF-50

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

Description The conversions are redundant as "_FLIP" variable of type "FLIP" already implements the "IFLIP" interface.

Listing 50:

```
+return IFLIP(address(_FLIP));
```

3.51 CVF-51

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** FLIP.sol

Description The comment says about time, while the variable name is about block number, which is not the same.
Recommendation Consider making them consistent.

Listing 51:

```
+/// @dev The last time that the State Chain updated the
    ↪ totalSupply
+uint256 private _lastSupplyUpdateBlockNum = 0;
```

3.52 CVF-52

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

Description Here an implicit underflow check performed by compiler is used to enforce an important business constraint.
Recommendation Consider explicitly requiring "flipTotalSupply" to not be lower than "genesisValidatorFlip".
Client Comment It is fine as it is – we rely on the compiler.

Listing 52:

```
+_mint(msg.sender, flipTotalSupply - genesisValidatorFlip);
```

3.53 CVF-53

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

Description Here a custom format of a signed message is used.
Recommendation Consider using a standard format as described in EIP-712: https://eips.ethereum.org/EIPS/eip-712
Client Comment We are OK with that.

Listing 53:

```
+keccak256(
+     abi.encodeWithSelector(
+          this.updateFlipSupply.selector,
+          SigData(sigData.keyManAddr, sigData.chainID, 0, 0,
   ↪ sigData.nonce, address(0)),
+          newTotalSupply,
+          stateChainBlockNumber,
+          staker
+     )
+)
```

3.54 CVF-54

- **Severity** Minor
- **Category** Documentation
- **Status** Fixed
- **Source** GovernanceCommunityGuarded.sol

Description Solidity does support virtual modifiers, so it is unclear why the "isCommunityKey" modifier cannot be made virtual.
Recommendation Consider clarifying.
Client Comment Wrong explanation on our side. The compiler enforces modifiers to have some logic (at least _;) which means it is not enforced that the child must override that. To avoid any future problem inheriting the contract and not being aware of that, we rather implement it this way so it is mandatory to override the _getGovernor and the _getCommunityKey. I have fixed the comment in the contract.

Listing 54:

```
+*             to the children. This is a workaround since the
   ↪ isCommunityKey modifier can't be
+*             made virtual. This contract needs to be marked as
   ↪ abstract.
```

3.55 CVF-55

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** GovernanceCommunityGuarded.sol

Description These functions should emit some events.

Listing 55:

```
+function enableCommunityGuard() external override
    ↪ isCommunityKey isCommunityGuardDisabled {

+function disableCommunityGuard() external override
    ↪ isCommunityKey isCommunityGuardEnabled {

+function suspend() external override isGovernor isNotSuspended
    ↪ {

+function resume() external override isGovernor isSuspended {
```

3.56 CVF-56

- **Severity** Minor
- **Category** Bad naming
- **Status** Fixed
- **Source** GovernanceCommunityGuarded.sol

Description The name is confusing, as this function actually returned a community guard state, rather than a community guard itself.
Recommendation Consider renaming to "getCommunityGuardState" or "isCommunityGuardDisabled".

Listing 56:

```
+function getCommunityGuard() external view override returns (
    ↪ bool) {
```

3.57 CVF-57

- **Severity** Minor
- **Category** Bad naming
- **Status** Fixed
- **Source** GovernanceCommunityGuarded.sol

Description The interpretation of a returned value is counter-intuitive, as "false" means that a community guard is enabled and “true” means it is disabled.
Recommendation Consider emphasizing this in the name and/or documentation comment, or inverting the returned value.

Listing 57:

```
+function getCommunityGuard() external view override returns (
    ↪ bool) {
```

3.58 CVF-58

- **Severity** Minor
- **Category** Bad naming
- **Status** Fixed
- **Source** GovernanceCommunityGuarded.sol

Description The names of these modifies look more like names of getter functions.
Recommendation More convertional names would be “onlyCommunityKey”, “onlyCommunityGuardDisabled” etc.

Listing 58:

```
+modifier isCommunityKey() {

+modifier isCommunityGuardDisabled() {

+modifier isCommunityGuardEnabled() {

+modifier isGovernor() {

+modifier isSuspended() {

+modifier isNotSuspended() {
```

3.59 CVF-59

- **Severity** Minor
- **Category** Procedural
- **Status** Info
- **Source** DepositToken.sol

Description String error messages are suboptimal.
Recommendation Consider using named errors instead.
Client Comment It is fine as is.

Listing 59:

```
+require(token.transfer(msg.sender, token.balanceOf(address(this
↪ ))), "DepositToken: transfer failed");
```

3.60 CVF-60

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** AggKeyNonceConsumer.sol

Description This function should emit some event with the new key manager address. The same event should be emitted from the constructor.

Listing 60:

```
+function updateKeyManager(SigData calldata sigData, IKeyManager
↪  keyManager)
```

3.61 CVF-61

- **Severity** Minor
- **Category** Procedural
- **Status** Info
- **Source** AggKeyNonceConsumer.sol

Description Here a custom format of a signed message is used.
Recommendation Consider using a standard format as described in EIP-712: https://eips.ethereum.org/EIPS/eip-712
Client Comment We are OK with that.

Listing 61:

```
+keccak256(
+    abi.encodeWithSelector(
+        this.updateKeyManager.selector,
+        SigData(sigData.keyManAddr, sigData.chainID, 0, 0,
↪ sigData.nonce, address(0)),
+        keyManager
+    )
+)
```

3.62 CVF-62

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** AggKeyNonceConsumer.sol

Description This function is redundant, just declare the "getKeyManager" function as public rather than external, so child contracts could call it internally.

Listing 62:

```
72 +function _getKeyManager() internal view returns (IKeyManager) {
```

3.63 CVF-63

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** IGovernanceCommunityGuarded.sol

Description This import is not used.

Listing 63:

```
4 +import "./IAggKeyNonceConsumer.sol";
```

3.64 CVF-64

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** IGovernanceCommunityGuarded.sol

Description This functions should emit some events and these events should be declared in this interface.

Listing 64:

```
20 +function enableCommunityGuard() external;

25 +function disableCommunityGuard() external;

31 +function suspend() external;

36 +function resume() external;
```

3.65 CVF-65

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** IFLIP.sol

Recommendation Should be "^0.8.0" unless there is something special about this particular version.

Listing 65:

```
+pragma solidity ^0.8.7;
```

3.66 CVF-66

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** IFLIP.sol

Description The "oldSupply" parameter is redundand as its value could be derived from the previous event.
Client Comment It is fine as it is.

Listing 66:

```
+event FlipSupplyUpdated(uint256 oldSupply, uint256 newSupply,
    ↪ uint256 stateChainBlockNumber);
```

3.67 CVF-67

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** IKeyManager.sol

Description The old key parameters are redundant as they could be derived from the previous events.
Client Comment It is fine as it is.

Listing 67:

```
+event AggKeySetByAggKey(Key oldAggKey, Key newAggKey);
+event AggKeySetByGovKey(Key oldAggKey, Key newAggKey);
+event GovKeySetByAggKey(address oldGovKey, address newGovKey);
+event GovKeySetByGovKey(address oldGovKey, address newGovKey);
+event CommKeySetByAggKey(address oldCommKey, address newCommKey
    ↪ );
+event CommKeySetByCommKey(address oldCommKey, address
    ↪ newCommKey);
```

3.68 CVF-68

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

Recommendation This function should be declared as “view”.

Listing 68:

```
34 +function balanceOf(address) external returns (uint256);
```

3.69 CVF-69

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Fixed
- **Source** IAggKeyNonceConsumer.sol

Description This function should emit some event and this event should be declared in this interface.

Listing 69:

```
23 +function updateKeyManager(SigData calldata sigData, IKeyManager
    ↪ keyManager) external;
```

3.70 CVF-70

- **Severity** Minor
- **Category** Bad naming
- **Status** Info
- **Source** IAggKeyNonceConsumer.sol

Description The name is confusing.
Recommendation More conventional name would be “setKeyManager”.
Client Comment It is fine as it is.

Listing 70:

```
23 +function updateKeyManager(SigData calldata sigData, IKeyManager
    ↪ keyManager) external;
```

3.71 CVF-71

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** IAggKeyNonceConsumer.sol

Recommendation This function should be declared as “view”.

Listing 71:

```
35 +function getKeyManager() external returns (IKeyManager);
```