ABDK CONSULTING

PROPOSAL
SMART CONTRACT AND CIRCUIT AUDIT

TORNADO

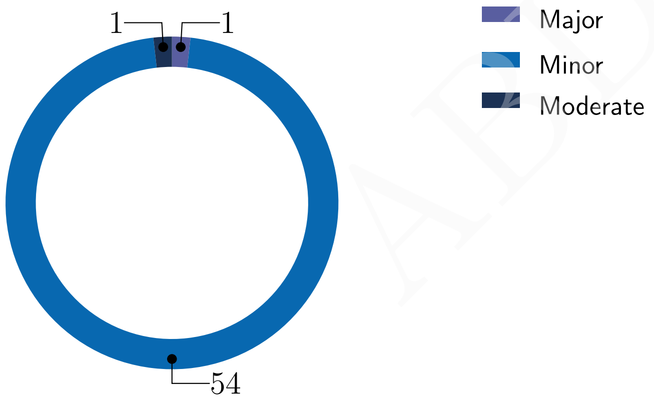
abdk.consulting

AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
20th March 2021

We've been asked to review the Tornado smart contracts and circuits related to the upgrade of the Tornado contract to a new one. We have identified only two significant issues.

ABDK



Findings

ID	Severity	Subject	Status
CVF-1	Minor	Bad naming	Opened
CVF-2	Minor	Index missing	Opened
CVF-3	Minor	Improper approach	Opened
CVF-4	Minor	Improper approach	Opened
CVF-5	Minor	Improper approach	Opened
CVF-6	Minor	Improper access specifiers	Opened
CVF-7	Minor	Redundant code	Opened
CVF-8	Moderate	Underflow	Opened
CVF-9	Minor	Complicated code	Opened
CVF-10	Minor	Code duplication	Opened
CVF-11	Minor	Complicated code	Opened
CVF-12	Minor	Complicated code	Opened
CVF-13	Minor	Complicated code	Opened
CVF-14	Minor	Redundant code	Opened
CVF-15	Minor	Comment missing	Opened
CVF-16	Minor	Improper type	Opened
CVF-17	Minor	Redundant code	Opened
CVF-18	Minor	Improper approach	Opened
CVF-19	Minor	Redundant code	Opened
CVF-20	Minor	Improper approach	Opened
CVF-21	Minor	Redundant code	Opened
CVF-22	Minor	Complicated code	Opened
CVF-23	Minor	Dublicated code	Opened
CVF-24	Minor	Redundant code	Opened
CVF-25	Minor	Improper type	Opened
CVF-26	Minor	Redundant code	Opened
CVF-27	Minor	Event missing	Opened

ABDK

ID	Severity	Subject	Status
CVF-28	Minor	Comment missing	Opened
CVF-29	Minor	Improper approach	Opened
CVF-30	Minor	Complicated code	Opened
CVF-31	Minor	Comment missing	Opened
CVF-32	Minor	Redundant code	Opened
CVF-33	Minor	Comment missing	Opened
CVF-34	Minor	Improper approach	Opened
CVF-35	Minor	Bad naming	Opened
CVF-36	Minor	Improper approach	Opened
CVF-37	Minor	Complicated code	Opened
CVF-38	Minor	Complicated code	Opened
CVF-39	Minor	Out of scope file	Opened
CVF-40	Minor	Complicated code	Opened
CVF-41	Minor	Bad naming	Opened
CVF-42	Minor	Bad naming	Opened
CVF-43	Major	Check missing	Opened
CVF-44	Minor	Improper approach	Opened
CVF-45	Minor	Improper approach	Opened
CVF-46	Minor	Redundant code	Opened
CVF-47	Minor	Improper access specifiers	Opened
CVF-48	Minor	Improper access specifiers	Opened
CVF-49	Minor	Bad naming	Opened
CVF-50	Minor	Improper type	Opened
CVF-51	Minor	Improper type	Opened
CVF-52	Minor	Event missing	Opened
CVF-53	Minor	Improper type	Opened
CVF-54	Minor	Bad naming	Opened
CVF-55	Minor	Redundant code	Opened
CVF-56	Minor	Improper approach	Opened

ABDK

Contents

ABDK

ABDK

1 Document properties

Version

Version	Date	Author	Description
0.1	Mar. 19, 2021	D. Khovratovich	Initial Draft
0.2	Mar. 20, 2021	D. Khovratovich	Minor revision
1.0	Mar. 20, 2021	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

ABDK

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. We were given access to three repositories with the common tag `proposal_audit` and reviewed the following files.

- Tornado-trees-proposal:
  - Proposal.sol
- Tornado-anonymity-mining:
  - TornadoProxy.sol
- Tornado-trees:
  - TornadoTrees.sol
  - BatchTreeUpdate.circom
  - Utils.circom

The audit goal is a general review of the smart contract and circuit structure, critical/major bugs detection and issuing the general recommendations.

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 About Customer

Tornado Cash is a decentralized Ethereum Mixer. ABDK had audited previous versions of Tornado Cash, and is now reviewing certain changes only.

2.3 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.

2.4 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.

ABDK

3 Detailed Results

3.1 CVF-1 Bad naming

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

Recommendation The suffix 'SIZE' is ambiguous. Better call it 'IN_BYTES'.

Listing 1: Bad naming

```
uint256 public constant ITEM_SIZE = 32 + 20 + 4;
uint256 public constant BYTES_SIZE = 32 + 32 + 4 + CHUNK_SIZE *
   ↪ ITEM_SIZE;
```

3.2 CVF-2 Index missing

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

Recommendation The "instance" parameters should be indexed.

Listing 2: Index missing

```
event DepositData(address instance, bytes32 indexed hash,
   ↪ uint256 block, uint256 index);
event WithdrawalData(address instance, bytes32 indexed hash,
   ↪ uint256 block, uint256 index);
```

ABDK

3.3 CVF-3 Improper approach

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

Description These strings are used multiple times to calculated 4-byte function selectors.
Recommendation Consider passing precomputed selectors instead of full signatures.

Listing 3: Improper approach

```
"deposits(uint256)",

"withdrawals(uint256)",
```

3.4 CVF-4 Improper approach

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

Recommendation It is probably safe to just round 'lastDepositLeaf' up to the nearest multiple of 'CHUNK_SIZE'.

Listing 4: Improper approach

```
require(lastDepositLeaf % CHUNK_SIZE == 0, "Incorrect
    ↪ TornadoTrees state");

require(lastWithdrawalLeaf % CHUNK_SIZE == 0, "Incorrect
    ↪ TornadoTrees state");
```

3.5 CVF-5 Improper approach

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

Recommendation This function could have been made cheaper if the caller also provides the correct length and the function just has to verify it. Maybe it is not possible if the function is called automatically and can not accept parameters.

ABDK

Listing 5: Improper approach

```
function findArrayLength(
```

3.6 CVF-6 Improper access specifiers

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

Recommendation This function should be made internal.

Listing 6: Improper access specifiers

```
function findArrayLength(
```

3.7 CVF-7 Redundant code

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** TornadoTrees.sol (proposal_audit)

Description It is a bad practice to leave test-only stuff in a production code. If you want this function to just return 0 in code or all tests, just inherit another smart contract from this smart contract, override the "findArrayLength" function, and test this inherited smart contract.

Listing 7: Redundant code

```
if (_from == 0 && _step == 0) {
  return 0; // for tests
}
```

3.8 CVF-8 Underflow

- **Severity** Moderate
- **Category** Overflow/Underflow
- **Status** Opened
- **Source** TornadoTrees.sol

Description "_from - _step" may cause underflow in case the "_step" value is greater then the remaining number of elements.

Listing 8: Underflow

```
_from = direction ? _from + _step : _from - _step;
```

ABDK

3.9 CVF-9 Complicated code

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

Recommendation These lines could be rewritten as: "(uint low, uint high) = direction ? (_from - step, _from) : (_from, _from + _step);".

Listing 9: Complicated code

```
uint256 high = direction ? _from : _from + _step;
uint256 low = direction ? _from - _step : _from;
```

3.10 CVF-10 Code duplication

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

Recommendation This code duplication could be avoided by calculating mid in the beginning of the loop body.

Listing 10: Code duplication

```
uint256 mid = (high + low) / 2;

  mid = (low + high) / 2;
```

3.11 CVF-11 Complicated code

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

Description The signature is hashed on every invocation.
Recommendation Consider hashing it once and reusing.

Listing 11: Complicated code

```
(success, ) = address(_tornadoTreesV1).staticcall{ gas: 2500 }(abi.encodeWithSignature(_type, index));
```

3.12 CVF-12 Complicated code

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

Recommendation These two lines could be combined as: "uint256 _depositsLength = depositsLength++";

Listing 12: Complicated code

```
uint256 _depositsLength = depositsLength;

depositsLength = _depositsLength + 1;
```

ABDK

3.13 CVF-13 Complicated code

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

Recommendation These lives could be combined as "uint256 _withdrawalsLength = withdrawalsLength++;".

Listing 13: Complicated code

```
uint256 _withdrawalsLength = withdrawalsLength;

withdrawalsLength = _withdrawalsLength + 1;
```

3.14 CVF-14 Redundant code

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

Description Emitting block number is probably redundant as it can be easily obtained from the event metadata.

Listing 14: Redundant code

```
emit WithdrawalData(_instance, _nullifierHash, blockNumber(), _withdrawalsLength);
```

ABDK

3.15 CVF-15 Comment missing

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

Recommendation There must be a comment on what exactly is proven.

Listing 15: Comment missing

```
bytes calldata _proof,
```

3.16 CVF-16 Improper type

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

Recommendation This can be type uint256.

Listing 16: Improper type

```
bytes32 _argsHash,
```

3.17 CVF-17 Redundant code

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** TornadoTrees.sol (proposal_audit)

Description These parameters are redundant, as the contract already knows the current root and the path indices could be derived from the offset already known to the smart contract.

Listing 17: Redundant code

```
bytes32 _currentRoot,

uint32 _pathIndices,

bytes32 _currentRoot,

uint256 _pathIndices,
```

ABDK

3.18 CVF-18 Improper approach

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

Description This probably can never happen as long as the proof verifies that the new root is an update with a non-zero entry and thus can not equal a previous root. If this check is just a sanity check, then a range check for new root should be there too.

Listing 18: Improper approach

```
require(_newRoot != previousDepositRoot, "Outdated deposit root
    ↪ ");
```

3.19 CVF-19 Redundant code

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

Description This check looks redundant.

Listing 19: Redundant code

```
168 require(_newRoot != previousDepositRoot, "Outdated deposit root");

213 require(_newRoot != previousWithdrawalRoot, "Outdated withdrawal root");
```

3.20 CVF-20 Improper approach

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

Description The ability to handle V1 deposits and withdrawals is needed for a few first chunks only, but the corresponding logic will consume extra gas forever. One solution would be to implement two versions of the "updateDepositsTree"/"updateWithdrawalsTree" functions: one that does support V1 deposits/withdrawals, and another that doesn't support them and just ensures that all the V1' deposits/withdrawals are already processed. Another solution would be to allocate in memory an array of 256 bytes32 values, copy there as much as 256 remaining V1 deposits/withdrawals. Then, if the array is not full yet, fill the rest with V2 deposits/withdrawals deleting them from the storage. Then perform the main loop over this in-memory array rather then on in-storage data structures.

Listing 20: Improper approach

```
181 bytes32 deposit = offset + i >= depositsV1Length ? deposits[offset + i] : tornadoTreesV1.deposits(offset + i);

188 if (offset + i >= depositsV1Length) {

227 bytes32 withdrawal = offset + i >= withdrawalsV1Length ? withdrawals[offset + i] : tornadoTreesV1.withdrawals(offset + i);

235 if (offset + i >= withdrawalsV1Length) {
```

3.21 CVF-21 Redundant code

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

Description The value "add (data, mul (ITEM_SIZE, i))" is calculated three times on every iteration.
Recommendation Consider using a single pointer initialized before the loop to the value "add (data, 0x64)" and incremented by "ITEM_SIZE" at the end of every loop iteration.

Listing 21: Redundant code

```
mstore(add(add(data, mul(ITEM_SIZE, i)), 0x7c), blockNumber)
mstore(add(add(data, mul(ITEM_SIZE, i)), 0x78), instance)
mstore(add(add(data, mul(ITEM_SIZE, i)), 0x64), hash)

mstore(add(add(data, mul(ITEM_SIZE, i)), 0x7c), blockNumber)
mstore(add(add(data, mul(ITEM_SIZE, i)), 0x78), instance)
mstore(add(add(data, mul(ITEM_SIZE, i)), 0x64), hash)
```

3.22 CVF-22 Complicated code

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

Description Moving the current root into the previous root and assigning a new value to the current root is suboptimal. More efficient way would be to have two variables: 'oddRoot' and 'evenRoo' whose roles would flip every time new chunk was validated.

Listing 22: Complicated code

```
previousDepositRoot = _currentRoot;
depositRoot = _newRoot;

previousWithdrawalRoot = _currentRoot;
withdrawalRoot = _newRoot;
```

ABDK

3.23 CVF-23 Dublicated code

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

Description This function's code is almost a duplicate of 'updateDepositTree'.
Recommendation Consider extracting the shared code to some utility.

Listing 23: Dublicated code

```
function updateWithdrawalTree(
```

3.24 CVF-24 Redundant code

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

Description There is no corresponding check in 'updateDepositsTree'. Probable this check is redundant here as well.

Listing 24: Redundant code

```
require(uint256(_newRoot) < SNARK_FIELD, "Proposed root is out
  ↪ of range");
```

3.25 CVF-25 Improper type

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

Description Does this function have to be public?

Listing 25: Improper type

```
function validateRoots(bytes32 _depositRoot, bytes32
  ↪ _withdrawalRoot) public view {
```

ABDK

3.26 CVF-26 Redundant code

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

Description These functions are redundant, as all the storage variables used in them are already public.

Listing 26: Redundant code

```
function getRegisteredDeposits() external view returns (bytes32[] memory _deposits) {

function getRegisteredWithdrawals() external view returns (bytes32[] memory _withdrawals) {
```

3.27 CVF-27 Event missing

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

Description This function should probably log some event.

ABDK

Listing 27: Event missing

```
function setTornadoProxyContract(address _tornadoProxy) external onlyGovernance {

function setVerifierContract(IBatchTreeUpdateVerifier _treeUpdateVerifier) external onlyGovernance {
```

3.28 CVF-28 Comment missing

- **Severity** Minor
- **Category** Documentation
- **Status** Opened
- **Source** BatchTreeUpdate.circom

Recommendation Some comment on the functionality and expected input range is recommended.

Listing 28: Comment missing

```
TreeLayer(height) {
```

3.29 CVF-29 Improper approach

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** BatchTreeUpdate.circom

Recommendation This template should be in its own file named "TreeLayer.circom".

Listing 29: Improper approach

```
TreeLayer(height) {
```

3.30 CVF-30 Complicated code

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** BatchTreeUpdate.circom

Recommendation Consider extracting "1 « height" into a variable to make the code easier to read.

Listing 30: Complicated code

```
signal input ins[1 << (height + 1)];
signal output outs[1 << height];

component hash[1 << height];
for(var i = 0; i < (1 << height); i++) {
```

ABDK

3.31 CVF-31 Comment missing

- **Severity** Minor
- **Category** Documentation
- **Status** Opened
- **Source** BatchTreeUpdate.circom

Recommendation Some comment on the admissible input range is recommended.

Listing 31: Comment missing

```
BatchTreeUpdate(levels, batchLevels, zeroBatchLeaf) {
```

3.32 CVF-32 Redundant code

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** BatchTreeUpdate.circom

Description As everything is anyway hardcoded in the same file, the function 'nthZero' is redundant.
Recommendation Just use a hardcoded value of 'nthZero(8)' to initialize the "MatchTree-Update" template.

Listing 32: Redundant code

```
nthZero(n) {

    if (n == 8) return
      ↪ 1727866832365266488142020977399598876819599857462961459339516246314
      ↪

CHUNK_TREE_HEIGHT = 8
main = BatchTreeUpdate(20, CHUNK_TREE_HEIGHT, nthZero(
   ↪ CHUNK_TREE_HEIGHT))
```

ABDK

3.33 CVF-33 Comment missing

- **Severity** Minor
- **Category** Documentation
- **Status** Opened
- **Source** Utils.circom

Recommendation Some comment on the functionality of the template would be helpful.

Listing 33: Comment missing

```
TreeUpdateArgsHasher(nLeaves) {
```

3.34 CVF-34 Improper approach

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** Utils.circom

Recommendation This template should be in a file named "TreeUpdateArgsHasher.circom" to make code navigation easier.

Listing 34: Improper approach

```
TreeUpdateArgsHasher(nLeaves) {
```

3.35 CVF-35 Bad naming

- **Severity** Minor
- **Category** Bad naming
- **Status** Opened
- **Source** Utils.circom

Recommendation For readability, constants are usually named in CAPS.

Listing 35: Bad naming

```
var header = 256 + 256 + 32;
```

3.36 CVF-36 Improper approach

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** Utils.circom

Recommendation Should be 256 + 160 + 32 to reflect the actual order of fields in a leaf.

Listing 36: Improper approach

```
var bitsPerLeaf = 160 + 256 + 32;
```

ABDK

3.37 CVF-37 Complicated code

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** Utils.circom

Description This pattern repeats several times in the code.
Recommendation Consider implementing it as a template. Inlining it every time is error-prone, as it is hard to notice a mistake in an index.

Listing 37: Complicated code

```
hasher.in[0] <== 0;
hasher.in[1] <== 0;
for(var i = 0; i < 254; i++) {
    hasher.in[i + 2] <== bitsOldRoot.out[253 - i];
}
```

3.38 CVF-38 Complicated code

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** Utils.circom

Recommendation Using a single counter for the number of data bits already populated would make code less error-prone and easier to read.

Listing 38: Complicated code

```
hasher.in[0] <== 0;
hasher.in[1] <== 0;

    hasher.in[i + 2] <== bitsOldRoot.out[253 - i];

hasher.in[256] <== 0;
hasher.in[257] <== 0;

    hasher.in[i + 258] <== bitsNewRoot.out[253 - i];

    hasher.in[i + 512] <== bitsPathIndices.out[31 - i];

    hasher.in[header + leaf * bitsPerLeaf + 0] <== 0;
    hasher.in[header + leaf * bitsPerLeaf + 1] <== 0;

        hasher.in[header + leaf * bitsPerLeaf + i + 2] <== bitsHash[leaf].out[253 - i];

        hasher.in[header + leaf * bitsPerLeaf + i + 256] <== bitsInstance[leaf].out[159 - i];

        hasher.in[header + leaf * bitsPerLeaf + i + 416] <== bitsBlock[leaf].out[31 - i];
```

ABDK

3.39 CVF-39 Out of scope file

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

Description We did not review these files.

Listing 39: Out of scope file

```
"tornado-trees/contracts/interfaces/ITornadoTreesV1.sol";
"tornado-trees/contracts/interfaces/IBatchTreeUpdateVerifier.sol
  ↪ ";
"tornado-trees/contracts/TornadoTrees.sol";
"tornado-trees/contracts/AdminUpgradeableProxy.sol";
"tornado-anonymity-mining/contracts/TornadoProxy.sol";
"./interfaces/ITornadoProxyV1.sol";
"./interfaces/IMiner.sol";
"./verifiers/BatchTreeUpdateVerifier.sol";
```

ABDK

3.40 CVF-40 Complicated code

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

Recommendation Consider passing all these addresses as constructor parameters and saving them into immutable variables (that are cheaper than storage variables). This would make it possible to test the code in testnet where addresses are different.

Listing 40: Complicated code

```
ITornadoTreesV1 public constant tornadoTreesV1 = ITornadoTreesV1
    ↪ (0x43a3bE4Ae954d9869836702AFd10393D3a7Ea417);
ITornadoProxyV1 public constant tornadoProxyV1 = ITornadoProxyV1
    ↪ (0x905b63Fff465B9fFBF41DeA908CEb12478ec7601);

      address(0x12D66f87A04A9E220743712cE6d9bB1B5616B8Fc),
      address(0x47CE0C6eD5B0Ce3d3A51fdb1C52DC66a7c3c2936),
      address(0x910Cbd523D972eb0a6f4cAe4618aD62622b39DbF),
      address(0xA160cdAB225685dA1d56aa342Ad8841c3b53f291)

      address(0xD4B88Df4D29F5CedD6857912842cff3b20C8Cfa3),
      address(0xFD8610d20aA15b7B2E3Be39B396a1bC3516c7144),
      address(0x22aaA7720ddd5388A3c0A3333430953C68f1849b),
      address(0xBA214C1c1928a32Bffe790263E38B4Af9bFCD659),
      address(0xd96f2B1c14Db8458374d9Aca76E26c3D18364307),
      address(0x4736dCf1b7A3d580672CcE6E7c65cd5cc9cFBa9D),
      address(0x169AD27A470D064DEDE56a2D3ff727986b15D52B),
      address(0x0836222F2B2B24A3F36f98668Ed8F0B38D1a872f)
```

3.41 CVF-41 Bad naming

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

Recommendation The name is ambiguous, consider using a more descriptive name.

Listing 41: Bad naming

```
event Deployed(address _contract);
```

3.42 CVF-42 Bad naming

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

Recommendation Events are usually named via nouns, such as "Deployment" or "Contract".

Listing 42: Bad naming

```
event Deployed(address _contract);
```

3.43 CVF-43 Check missing

- **Severity** Major
- **Category** Suboptimal
- **Status** Opened
- **Source** Proposal.sol

Description This function can be called multiple times.
Recommendation Consider using some protection.

Listing 43: Check missing

```
function executeProposal() public {
```

3.44 CVF-44 Improper approach

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

Description The same event is emitted 4 times with distinct parameters and probably different semantics. It would make more sense to have four different events or one with four parameters.

Listing 44: Improper approach

```
emit Deployed(address(verifier));

emit Deployed(address(tornadoTreesImpl));

emit Deployed(address(upgradeableProxy));

emit Deployed(address(proxy));
```

ABDK

3.45 CVF-45 Improper approach

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

Description The same event is used to log deployments of different components. This makes is hard to know what address belongs to what component.
Recommendation Consider either declaring a single event with four type-safe parameters, or four different events, each having one type safe-parameter, where for different events the parameter types are different.

Listing 45: Improper approach

```
emit Deployed(address(verifier));

emit Deployed(address(tornadoTreesImpl));

emit Deployed(address(upgradeableProxy));

emit Deployed(address(proxy));
```

ABDK

3.46 CVF-46 Redundant code

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

Recommendation The type case to 'IBatchTreeUpdateVerifier' is redundant.

Listing 46: Redundant code

```
tornadoTrees.initialize(address(proxy), IBatchTreeUpdateVerifier
    ↪ (address(verifier)));
```

3.47 CVF-47 Improper access specifiers

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

Recommendation This function doesn't have to be public. It also could be made pure.

Listing 47: Improper access specifiers

```
function getSearchParams() public view returns (TornadoTrees.
    ↪ SearchParams memory) {
```

3.48 CVF-48 Improper access specifiers

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

Recommendation These functions don't have to be public.

Listing 48: Improper access specifiers

```
function getEthInstances() public pure returns (address[4]
    ↪ memory) {

function getErc20Instances() public pure returns (address[8]
    ↪ memory) {

function getInstances() public pure returns (TornadoProxy.
    ↪ Instance[] memory instances) {
```

3.49 CVF-49 Bad naming

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

Recommendation Enum constants are usually named IN_CAPITAL_CASE.

Listing 49: Bad naming

```
num InstanceState { Disabled, Enabled, Mineable }
```

3.50 CVF-50 Improper type

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

Recommendation The type of this field should be "ITornadoInstance".

Listing 50: Improper type

```
address instance;
```

ABDK

3.51 CVF-51 Improper type

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** TornadoProxy.sol (proposal_audit)

Recommendation Should be 'ITornadoTrees'.

Listing 51: Improper type

```
ddress _tornadoTrees,
```

3.52 CVF-52 Event missing

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

Recommendation This function should probably emit some event.

Listing 52: Event missing

```
unction updateInstance(ITornadoInstance _instance, InstanceState
  ↪ _state) external onlyGovernance {

unction setTornadoTreesContract(address _instance) external
  ↪ onlyGovernance {
```

3.53 CVF-53 Improper type

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

Recommendation The type should be 'ITornadoTrees'.

Listing 53: Improper type

```
unction setTornadoTreesContract(address _instance) external
  ↪ onlyGovernance {
```

ABDK

3.54 CVF-54 Bad naming

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

Recommendation The 'amount' would be a better name.

Listing 54: Bad naming

```
int256 _balance
```

3.55 CVF-55 Redundant code

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

Recommendation Treating zero balance as "all" is redundant, as _balance=2Ì‚256-1 would basically do the same.

Listing 55: Redundant code

```
int256 balance = _balance == 0 ? totalBalance : Math.min(
    ↪ totalBalance, _balance);

int256 balance = _balance == 0 ? totalBalance : Math.min(
    ↪ totalBalance, _balance);
```

3.56 CVF-56 Improper approach

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

Recommendation Consider using "send" instead of "transfer", as using transfer is discouraged nowadays.

Listing 56: Improper approach

```
to.transfer(balance);
```

ABDK