Skale Network

Date	October 2020
Lead Auditor	Sergii Kravchenko
Co-auditors	Shayan Eskandari

1 Executive Summary

This report presents the results of our second engagement with **Skale** to review Skale Manager code that manages the nodes and schain grouping and rotations. The delegation, validator and token functions of the SKALE Manager system were previously audited by ConsenSys Diligence.

The review was conducted over two weeks, from **Oct 19, 2020** to **Nov 6, 2020** by Sergii Kravchenko and Shayan Eskandari.

During the first week, we got a walk through on the code, reviewed the overall design and started the cryptographic review in parallel.

During the second week, the code was updated by Skale team, adding `BountyV2.sol` and its implementation to the rest of the Skale code base. We had to review some of the changed code through out the rest of the files, and started working on the new Bounty system.

During the third week the cryptography audit was completed and we drafted a report for the Skale team to review.

2 Scope

Our review focused on the commit hash `cf4a07a5198b387fd128e1863a26bbfc789856c4`.

2.1 Documentation

- Whitepaper: https://skale.network/whitepaper

Overview:

- Trusted validators register "Nodes" in the Network.
- Each node is allocated "Spaces" (up to 128 units per node) and this is managed by SKALE Manager, and allocated out as users request "skale chains" of 3 sizes: small ($1/128$), medium ($1/32$) and large (1).
- Skale Chains are requested by users who stake/deposit SKL tokens. Successful requests form "SKALE Chain groups" of 16 randomly chosen nodes with appropriate allocated space (can be any 3n+1, default is 16 nodes, each node with 1/x space allocated)
- Nodes in SKALE Chain groups may be rotated over time.
- SKALE Chain initialization and rotation perform DKG so that a threshold of nodes sign messages in the Network and on each schain.

Blog posts:

- https://skale.network/blog/the-skale-network-primer/
- https://skale.network/blog/containerization-the-future-of-decentralized-infrastructure/
- https://skale.network/blog/containerization-the-future-of-decentralized-infrastructure-2/

Cryptography related articles:

- https://medium.com/skale/bls-deep-dive-793a4e8a6f4e
- https://link.springer.com/content/pdf/10.1007%2F3-540-48910-X_21.pdf
- https://www.iacr.org/archive/asiacrypt2001/22480516.pdf
- https://www.iacr.org/archive/pkc2003/25670031/25670031.pdf

2.2 Objectives

Together with the Skale team, we identified the following priorities for our review:

1. Ensure that the system is implemented consistently with the intended functionality, and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.
3. Cryptography implementations

3 Recommendations

During the course of our review, we made the following recommendations:

3.1 Reorder functions with hardcoded values

Description

There are many initialized functions in different contracts that have many hardcoded values for critical system variables. It is suggested to move these functions to top of the contract for better visibility and reducing the possibility of future error on deployment.

Examples

code/contracts/BountyV2.sol:L173-L180

```
function initialize(address contractManagerAddress) public override initiali
    Permissions.initialize(contractManagerAddress);
    _nextEpoch = 0;
    _epochPool = 0;
    _bountyWasPaidInCurrentEpoch = 0;
    bountyReduction = false;
    nodeCreationWindowSeconds = 3 * SECONDS_PER_DAY;
}
```

code/contracts/ConstantsHolder.sol:L210-L225

```
function initialize(address contractsAddress) public override initializer {
    Permissions.initialize(contractsAddress);

    msr = 0;
    rewardPeriod = 2592000;
    allowableLatency = 150000;
    deltaPeriod = 3600;
    checkTime = 300;
    launchTimestamp = uint(-1);
    rotationDelay = 12 hours;
    proofOfUseLockUpPeriodDays = 90;
    proofOfUseDelegationPercentage = 50;
    limitValidatorsPerDelegator = 20;
    firstDelegationsMonth = 0;
    complaintTimelimit = 1800;
}
```

code/contracts/Nodes.sol:L676-L682

```
function initialize(address contractsAddress) public override initializer {
    Permissions.initialize(contractsAddress);

    numberOfActiveNodes = 0;
    numberOfLeavingNodes = 0;
    numberOfLeftNodes = 0;
}
```

3.2 Test suite with gas usage estimation

Description

As also mentioned in issue 5.3, many functionalities in the code bases has high gas usage. We recommend implementation of proper gas usage estimation in the test suite to make sure none of the tested work flows will exceed the block gas limit.

Our suggestion is to use eth-gas-reporter as it is easy to integrate it with current Truffle test suite.

3.3 [Optimization] Reduce gas cost for error messages

Description

There are some instances in the codebase that string concatenation `strConcat` is used to populate a string for the error messages. Specially in the scenarios that these are in the for loops, it can become expensive to run, specially given that these concatenations will happen in every round and not just the failure workflow.

Examples

code/contracts/NodeRotation.sol:L87-L104

```
for (uint i = 0; i < schains.length; i++) {
    Rotation memory rotation = rotations[schains[i]];
    if (rotation.nodeIndex == nodeIndex && now < rotation.freezeUntil) {
        continue;
    }
    string memory schainName = schainsInternal.getSchainName(schains[i]);
    string memory revertMessage = "Node cannot rotate on Schain ";
    revertMessage = revertMessage.strConcat(schainName);
    revertMessage = revertMessage.strConcat(", occupied by Node ");
    revertMessage = revertMessage.strConcat(rotation.nodeIndex.uint2str());
    string memory dkgRevert = "DKG process did not finish on schain ";
    ISkaleDKG skaleDKG = ISkaleDKG(contractManager.getContract("SkaleDKG"));
    require(
        skaleDKG.isLastDKGSuccessful(keccak256(abi.encodePacked(schainName))),
        dkgRevert.strConcat(schainName));
    require(rotation.freezeUntil < now, revertMessage);
    _startRotation(schains[i], nodeIndex);
}
```

Recommendation

For the error messages, it is suggested to either use error codes (with UI translation to error messages), or in the example above you can also use `abi.encodePacked(str1, str2, str3)`.

3.4 Review the Code Quality recommendations in Appendix 1

Other comments related to readability and best practices are listed in
endix 1

4 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

4.1 Actors

Overall Skale manager is currently a permissioned system (with intention to progressively decentralize over time), meaning that only the actors below have the ability to configure and interact with the system. The rest of the users, can read and use the verification methods on the contracts to make sure the nodes are running as they should be.

The relevant actors are listed below with their respective abilities:

- Owner
    - Many permissions in out of scope contracts (e.g. `ValidatorService`, `TokenState`) that can directly affect the functionalities of the overall system.
    - Call any external function with the modifier `allow`.
    - Set penalty for a given offence `setPenalty()`
    - Populate Bounty for all specific nodes `populateBountyV2()`.
    - Enable and Disable Bounty reduction `enableBountyReduction()`.
    - Rotate an schain, skipping the set delay `skipRotationDelay()`.
    - Can set and replace (upgrade) any smart contract in the system `setContractsAddress()`.
    - Can set and replace any constant (System-wide variables) in the system `ConstantHolder.sol`.
- Admin
    - Some permissions in out of scope contracts (e.g. `ValidatorService`) that can directly affect the functionalities of the overall system.
    - Delete schain entirely by root `deleteSchainByRoot()`.
    - Can set/remove a node from maintenance.

    - Can release slashed tokens by calling `forgive()`.
- Node
    - Create a node if possible `CreateNode()` -> `checkPossibilityCreatingNode()`. Note that the node details (IP, Port, etc) cannot be changed after initial creation.
    - Can get a bounty.
    - Can set/remove the node from maintenance.
    - Exit the node.
    - Participate in DKG process.
- Validator
    - Can link the node's address.
    - Can exit a node that belongs to the validator.
    - Can set/remove a node from maintenance.
    - Note that the node details (IP, Port, etc) cannot be changed after initial creation
- Everyone
    - Add/remove schain.

4.2 Important Security Properties

The following is a non-exhaustive list of security properties that were discussed in this audit:

- A random number is used to select a node for a group and generate the groups. It should be noted that this random number uses a hash of the previous block with some other variables, and it could potentially be manipulated by the miners. Although, as it does not have any obvious security implications, it was not reported as an issue.
- The control over the system is currently assigned to multiple parties by the `owner` address, which is a multi-sig. This address can change anything in the system: call any external function with the modifier `allow` and change any contract addresses (See Actors for more details).

4.3 Cryptography Audit

- In addition to the issues reported below, here is the cryptography audit report as an attachment.

5 Issues

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

5.1 Every node gets a full validator's bounty Major ✓ Fixed

Resolution

This issue is addressed in Bug/skale 3273 formula fix 435 and SKALE-3273 Fix BountyV2 populating error 438.

The main change is related to how bounties are calculated for each validator. Below are a few notes on these pull requests:

- `nodesByValidator` mapping is no longer used in the codebase and the non-zero values are deleted when `calculateBounty()` is called for a specific validator. The mapping is kept in the code for compatible storage layout in upgradable proxies.
- Some functions such as `populate()` was developed for the transition to the upgraded contracts (rewrite `_effectiveDelegatedSum` values based on the new calculation formula). This function is not part of this review and will be removed in the future updates.

- Unlike the old architecture, `nodesByValidator[validatorId]` is no longer used within the system to calculate `_effectiveDelegatedSum` and bounties. This is replaced by using overall staked amount and duration.
- If a validator does not claim their bounty during a month, it is considered as a misbehave and her bounty goes to the bounty pool for the next month.

Description

To get the bounty, every node calls the `getBounty` function of the `SkaleManager` contract. This function can be called once per month. The size of the bounty is defined in the `BountyV2` contract in the `_calculateMaximumBountyAmount` function:

code/contracts/BountyV2.sol:L213-L221

```
return epochPoolSize
    .add(_bountyWasPaidInCurrentEpoch)
    .mul(
        delegationController.getAndUpdateEffectiveDelegatedToValidator(
            nodes.getValidatorId(nodeIndex),
            currentMonth
        )
    )
    .div(effectiveDelegatedSum);
```

The problem is that this amount actually represents the amount that should be paid to the validator of that node. But each node will get this amount. Additionally, the amount of validator's bounty should also correspond to the number of active nodes, while this formula only uses the amount of delegated funds.

Recommendation

Every node should get only their parts of the bounty.

5.2 A node exit prevents some other nodes from exiting for some period Medium Pending

Resolution

Skale team's comment:

```
known issue, acknowledged, assigned as the work for the next few months as an improve
```

Description

When a node wants to exit, the `nodeExit` function should be called as many times, as there are schains in the node. Each time one schain is getting removed from the node. During every call, all the active schains are getting frozen for 12 hours.

code/contracts/NodeRotation.sol:L84-L105

```
function freezeSchains(uint nodeIndex) external allow("SkaleManager") {
    SchainsInternal schainsInternal = SchainsInternal(contractManager.getCor
    bytes32[] memory schains = schainsInternal.getActiveSchains(nodeIndex);
    for (uint i = 0; i < schains.length; i++) {
        Rotation memory rotation = rotations[schains[i]];
        if (rotation.nodeIndex == nodeIndex && now < rotation.freezeUntil) {
            continue;
        }
        string memory schainName = schainsInternal.getSchainName(schains[i])
        string memory revertMessage = "Node cannot rotate on Schain ";
        revertMessage = revertMessage.strConcat(schainName);
        revertMessage = revertMessage.strConcat(", occupied by Node ");
        revertMessage = revertMessage.strConcat(rotation.nodeIndex.uint2str(
        string memory dkgRevert = "DKG process did not finish on schain ";
        ISkaleDKG skaleDKG = ISkaleDKG(contractManager.getContract("SkaleDKG
        require(
            skaleDKG.isLastDKGSuccessful(keccak256(abi.encodePacked(schainNa
            dkgRevert.strConcat(schainName));
        require(rotation.freezeUntil < now, revertMessage);
        _startRotation(schains[i], nodeIndex);
    }
}
```

Because of that, no other node that is running one of these schains can exit during that period. In the worst-case scenario, one malicious node has 128 Schains and calls `nodeExit` every 12 hours. That means that some nodes will not be able to exit for 64 days.

Recommendation

Make node exiting process less synchronous.

5.3 Removing a node require multiple transactions and may be very expensive Medium Pending

Resolution

Skale team's comment:

```
Acknowledged, assigned as the work for the next few months  (improvement). Please ass
```

Description

When removing a node from the network, the owner should redistribute all the schains that are currently on that node to the other nodes. To do so, the validator should call the `nodeExit` function of the `SkaleManager` contract. In this function, only one schain is going to be removed from the node. So the node would have to call the `nodeExit` function as many times as there are schains in the node. Every call iterates over every potential node that can be used as a replacement (like in issue 5.4).

In addition to that, the first call will iterate over all schains in the node, make 4 SSTORE operations and external calls for each schain:

code/contracts/NodeRotation.sol:L204-L210

```
function _startRotation(bytes32 schainIndex, uint nodeIndex) private {
    ConstantsHolder constants = ConstantsHolder(contractManager.getContract(
    rotations[schainIndex].nodeIndex = nodeIndex;
    rotations[schainIndex].newNodeIndex = nodeIndex;
    rotations[schainIndex].freezeUntil = now.add(constants.rotationDelay());
    waitForNewNode[schainIndex] = true;
}
```

This may hit the block gas limit even easier than issue 5.4.

If the first transaction does not hit the block's gas limit, the maximum price of deleting a node would be BLOCK_GAS_COST * 128. At the moment, it's around $50,000.

Recommendation

Optimize the process of deleting a node, so it can't hit the gas limit in one transaction, and the overall price should be cheaper.

5.4 Adding a new schain may potentially hit the gas limit

Medium Pending

Resolution

Skale team's comment:

```
Acknowledged, assigned as the work for the next few months (improvement) Please assig
```

Description

When adding a new schain, a group of random 16 nodes is randomly selected to run that schain. In order to do so, the `_generateGroup` function iterates over all the nodes that can be used for that purpose:

le/contracts/SchainsInternal.sol:L522-L541

```
function _generateGroup(bytes32 schainId, uint numberOfNodes) private returr
    Nodes nodes = Nodes(contractManager.getContract("Nodes"));
    uint8 space = schains[schainId].partOfNode;
    nodesInGroup = new uint[](numberOfNodes);

    uint[] memory possibleNodes = isEnoughNodes(schainId);
    require(possibleNodes.length >= nodesInGroup.length, "Not enough nodes t
    uint ignoringTail = 0;
    uint random = uint(keccak256(abi.encodePacked(uint(blockhash(block.numbe
    for (uint i = 0; i < nodesInGroup.length; ++i) {
        uint index = random % (possibleNodes.length.sub(ignoringTail));
        uint node = possibleNodes[index];
        nodesInGroup[i] = node;
        _swap(possibleNodes, index, possibleNodes.length.sub(ignoringTail).s
        ++ignoringTail;

        _exceptionsForGroups[schainId][node] = true;
        addSchainForNode(node, schainId);
        require(nodes.removeSpaceFromNode(node, space), "Could not remove sp
    }
```

If the total number of nodes exceeds around a few thousands, adding a schain may hit the block gas limit.

Recommendation

Avoid iterating over all nodes when selecting a random node for a schain.

5.5 Typos Minor

Description

There are a few typos in the contract source code. This could result in unforeseeable issues in the future development cycles.

Examples

succesful instead of **successful**:

code/contracts/SkaleDKG.sol:L77-L78

```
apping(bytes32 => uint) public lastSuccesfulDKG;
```

code/contracts/SkaleDKG.sol:L372-L373

```
    _setSuccesfulDKG(schainId);
}
```

and many other instances of `succesful` through out the code.

5.6 Redundant Checks in some flows

Description

The workflows in the Skale network are complicated and multi layers (multiple calls to different modules). Some checks are done in this process that are redundant and can be removed, based on the current code and the workflow.

Examples

An example of this redundancy, is when completing a node exit procedure. `completeExit()` checks if the node status is leaving and if so continues:

code/contracts/Nodes.sol:L309-L311

```
require(isNodeLeaving(nodeIndex), "Node is not Leaving");

_setNodeLeft(nodeIndex);
```

However, in `_setNodeLeft()` it has an if clause for the status being Active, which will never be true.

code/contracts/Nodes.sol:L795-L803

```
function _setNodeLeft(uint nodeIndex) private {
    nodesIPCheck[nodes[nodeIndex].ip] = false;
    nodesNameCheck[keccak256(abi.encodePacked(nodes[nodeIndex].name))] = fal
    delete nodesNameToIndex[keccak256(abi.encodePacked(nodes[nodeIndex].name
    if (nodes[nodeIndex].status == NodeStatus.Active) {
        numberOfActiveNodes--;
    } else {
        numberOfLeavingNodes--;
    }
```

Recommendation

To properly check the code flows for unreachable code and remove redundant checks.

5.7 Presence of empty function ✓ Fixed

Resolution
Implemented in Bug/skale 3273 formula fix 435.

Description

`estimateBounty()` is declared but neither implemented nor used in any part of the current code base.

Examples

code/contracts/BountyV2.sol:L142-L159

```
function estimateBounty(uint /* nodeIndex */) external pure returns (uint) {
    revert("Not implemented");
    // ConstantsHolder constantsHolder = ConstantsHolder(contractManager.getC
    // Nodes nodes = Nodes(contractManager.getContract("Nodes"));
    // TimeHelpers timeHelpers = TimeHelpers(contractManager.getContract("Tim

    // uint stagePoolSize;
    // uint nextStage;
    // (stagePoolSize, nextStage) = _getEpochPool(timeHelpers.getCurrentMonth

    // return _calculateMaximumBountyAmount(
    //     stagePoolSize,
    //     nextStage.sub(1),
    //     nodeIndex,
    //     constantsHolder,
    //     nodes
    // );
}
```

Recommendation

It is suggested to remove dead code from the code base, or fully implement it before the next step.

5.8 Presence of TODO tags in the codebase

Description

A few TODO tags are present in the codebase.

Examples

code/contracts/SchainsInternal.sol:L153-L160

```
// TODO:
// optimize
for (uint i = 0; i + 1 < schainsAtSystem.length; i++) {
    if (schainsAtSystem[i] == schainId) {
        schainsAtSystem[i] = schainsAtSystem[schainsAtSystem.length.sub(1)];
        break;
    }
}
```

code/contracts/SchainsInternal.sol:L294-L301

```
/**
 * @dev Checks whether schain name is available.
 * TODO Need to delete - copy of web3.utils.solioditySha3
 */
function isSchainNameAvailable(string calldata name) external view returns (
    bytes32 schainId = keccak256(abi.encodePacked(name));
    return schains[schainId].owner == address(0) && !usedSchainNames[schainI
}
```

code/contracts/Nodes.sol:L81-L89

```
// TODO: move outside the contract
struct NodeCreationParams {
    string name;
    bytes4 ip;
    bytes4 publicIp;
    uint16 port;
    bytes32[2] publicKey;
    uint16 nonce;
}
```

And a few others in test scripts.

Appendix 1 - Code Quality Recommendations

A.1.1 Inline Documentation

There are many instances of confusing function/variable names that require better documentation to increase code readability. It is recommended to add more inline documentations, preferably in NatSpec style.

Appendix 2 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any

token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

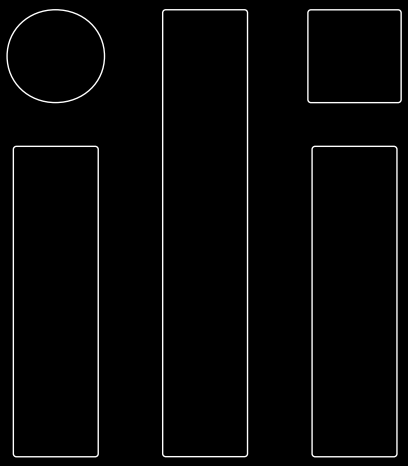
LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.

Request a Security Review Today

Get in touch with our team to request a quote for a smart contract audit.

CONTACT US



AUDITS
FUZZING
SCRIBBLE
BLOG
TOOLS
RESEARCH
ABOUT
CONTACT
CAREERS
PRIVACY POLICY

Subscribe to Our Newsletter

Stay up-to-date on our latest offerings, tools, and the world of blockchain security.

Email*

e-mail address

→

POWERED BY CONSENSYS