HOPR Audit

Côme du Crest

2024-08-26

Table of contents

HOPR Audit

This document presents the finding of a smart contract audit conducted by Côme du Crest for Gnosis.

Scope

The scope includes selected contracts within hoprnet/hoprnet/smart-contract-v3 as of commit a55fa71.

As specified in the overview document, contracts in scope are:

```
|---- Channels.sol # payment channels between nodes in the HOPR network
|---- Crypto.sol # cryptographic primitives used by the HOPR protocol
|---- MultiSig.sol # abstraction of interaction between nodes and Safes in the HOPR network
|---- interfaces
|   |---- IAvatar.sol # interface for Avatar (Safe).
|   |---- INetworkRegistryRequirement.sol # interface for logics used in "network registry"
|   |---- INodeManagementModule.sol # interface for node management Module
|   |---- INodeSafeRegistry.sol # interface for node safe registry
|---- node-stake
|   |---- NodeSafeRegistry.sol # registry for nodes and Safes in the HOPR network
|   |---- NodeStakeFactory.sol # factory contract to deploy Safe and node management Module for node runners
|   |---- permissioned-module
|       |---- CapabilityPermissions.sol # library for capability management of node management Module
|       |---- NodeManagementModule.sol # implementation logics for node management Module
|       |---- SimplifiedModule.sol # simplified implementation of Module
|---- utils
|   |---- EnumerableStringSet.sol # enumerable sets for String type
|   |---- EnumerableTargetSet.sol # enumerable sets for Target type
|   |---- SafeSuiteLib.sol # deployment addresses for Safe v.1.4.1
|   |---- TargetUtils.sol # utilities for Target type
|---- Announcements.sol # node announcement scheme which is independent from staking
|---- Ledger.sol # snapshot-based indexing of HOPR Channels
|---- NetworkRegistry.sol # implements network gate which will be removed eventually
|---- TicketPriceOracle.sol # standalone oracle to change HOPR ticket price network-wide
|---- proxy # implementations of adapters between network registry and staking
    |---- DummyProxyForNetworkRegistry.sol
    |---- SafeProxyForNetworkRegistry.sol
    |---- StakingProxyForNetworkRegistry.sol
```

Context

The repository implements three sets of features. The first one is a permission and access control module using Safe and a NodeManagementModule to enable / disable actions for a chain key. The second one is a set of cryptographic primitives using secp256k1 to expose a VRF. The third one implements payment channels with probabilistic payments using the VRF.

Status

The report has been sent to the core developer.

The report has been reviewed and fixes implemented in branch q/response-to-audit-20240826/.

The fixes have been reviewed and every issue has been responded to. Response comments have been added to the corresponding issues.

Legal Information And Disclaimer

1. This report is based solely on the information provided by hopr (the "Company"), with the assumption that the information provided to Gnosis is authentic, accurate, complete, and not misleading as of the date of this report. Gnosis has not conducted any independent enquiries, investigations or due diligence in respect of the Company, its business or its operations.
2. Changes to the information contained in the documents, repositories and any other materials referenced in this report might affect or change the analysis and conclusions presented. Gnosis is not responsible for monitoring, nor will we be aware of, any future additions, modifications, or deletions to the audited code. As such, Gnosis does not assume any responsibility to update any information, content or data contained in this report following the date of its publication.
3. This report does not address, nor should it be interpreted as addressing, any regulatory, tax or legal matters, including but not limited to: tax treatment, tax consequences, levies, duties, data privacy, data protection laws, issues relating to the licensing of information technology, intellectual property, money laundering and countering the financing of terrorism, or any other legal restrictions or prohibitions. Gnosis disclaims any liability for omissions or errors in the findings or conclusions presented in this report.
4. The views expressed in this report are solely our views regarding the specific issues discussed within this report. This report is not intended to be exhaustive, nor should it be construed as an assurance, guarantee or warranty that the code is free from bugs, vulnerabilities, defects or deficiencies. Different use cases may carry different risks, and integration with third-party applications may introduce additional risks.
5. This report is provided for informational purposes only and should not be used as the basis for making investment or financial decisions. This report does not constitute investment research and should not be viewed as an invitation, recommendation, solicitation or offer to subscribe for or purchase any securities, investments, products or services. Gnosis is not a financial advisor, and this report does not constitute financial or investment advice.
6. The statements in this report should be considered as a whole, and no individual statement should be extracted or referenced independently.
7. This report is addressed exclusively to the Company, and except with prior written consent from Gnosis, it may not be shared, disclosed, transmitted, or relied upon by any other person or entity for any purpose.
8. To the fullest extent permitted by applicable laws, Gnosis disclaims any and all other liability, whether in contract, tort, or otherwise, that may arise from this report or the use thereof.

Issues

[High] Incorrect implementation of ecAdd() when a is not 0

Summary

The function to add two elliptic curve points together `ecAdd()` does not implement point doubling correctly. The used formula is incorrect but does not differentiate with the correct formula in the case where `a = 0` so the result only differs when `a != 0`.

Vulnerability Detail

The formula for doubling a point `(x, y)` on an elliptic curve `y^2 = x^3 + ax + b` is as follows:

```
lambda = (3 * x^2 + a) / (2 * y)
xr = lamba ^ 2 - 2 * x
yr = lambda * (x - xr) - y
```

The code incorrectly implements:

```
lambda = (3 * x^2) / (2 * y + a)
xr = lamba ^ 2 - 2 * x
yr = lambda * (x - xr) - y
```

See code:

```
    function ecAdd(
        uint256 pX,
        uint256 pY,
        uint256 qX,
        uint256 qY,
        uint256 a
    )
        internal
        view
        returns (uint256 rx, uint256 ry)
    {
        // solhint-disable-next-line no-inline-assembly
        assembly {
            ...
            let lambda
            let toInvert
            switch and(eq(pX, qX), eq(pY, qY))
            // P == Q ?
            case true {
                // Point double
                toInvert := addmod(mulmod(2, pY, SECP256K1_BASE_FIELD_ORDER), a, SECP256K1_BASE_FIELD_ORDER) // 2 * p.y  // @audit 2*p.y + a

                // compute (2 * p.y) ^ -1 using expmod precompile
                let payload := mload(0x40)
                mstore(payload, 0x20) // Length of Base
                mstore(add(payload, 0x20), 0x20) // Length of Exponent
                mstore(add(payload, 0x40), 0x20) // Length of Modulus
                mstore(add(payload, 0x60), toInvert) // Base
                mstore(add(payload, 0x80), 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2D) // p - 1
                mstore(add(payload, 0xa0), SECP256K1_BASE_FIELD_ORDER) // Modulus
                if iszero(staticcall(not(0), 0x05, payload, 0xC0, payload, 0x20)) {
                    // 0x05 == expmod precompile
                    revert(0, 0)
                }
                lambda :=
                    mulmod( // (3 * p.x ^ 2) * (2 * p.y) ^ -1
                        mulmod( // 3 * p.x ^ 2
                        3, mulmod(pX, pX, SECP256K1_BASE_FIELD_ORDER), SECP256K1_BASE_FIELD_ORDER), // @audit lambda should be (3*p.x ^2 + a)/(2*p.y)
                        mload(payload),
                        SECP256K1_BASE_FIELD_ORDER
                    )
            }
            case false {
                ...
            }
            ...
        }
    }
```

Impact

`ecAdd()` is used with `a != 0` in the `hashToCurve()` function which is used by `vrfVerify()` to produce a pseudo-random point. Incorrectly implementing this cryptographic primitive could lead to manipulable randomness resulting in biased probabilistic payment tickets.

Code Snippets

https://github.com/hoprnet/hoprnet/blob/a55fa71461851d0e5d5a3cb090a5bfcc6da11fcb/ethereum/contracts/src/Crypto.sol#L211-L317

Recommendation

Update the `ecAdd()` function to implement correct point doubling. Add a test with a simple point doubling on a curve where `a != 0` for example on `y^2 = x^3 + 2*x + 2 mod 17` where `(5, 1)+ (5, 1)= (6, 3)` which fails to be verified by this implementation.

Response

Fixed by commit 1e8b9cf.

[Low] Channels does not completely follow ERC1820

Summary

The function `canImplementInterfaceForAddress()` must return the `ERC1820_ACCEPT_MAGIC` value when called on the implementer with the given address and interface hash. That is `HoprChannels.canImplementInterfaceForAddress(TOKENS_RECIPIENT_INTERFACE_HASH, address(HoprChannels))= keccak256(abi.encodePacked("ERC1820_ACCEPT_MAGIC"))` which is not the case currently.

Vulnerability Detail

`HoprChannels` inherits from `ERC1820Implementer`:

```
contract HoprChannels is
    IERC777Recipient,
    ERC1820Implementer,
    Multicall,
    HoprLedger(INDEX_SNAPSHOT_INTERVAL),
    HoprMultiSig,
    HoprCrypto,
    HoprChannelsEvents
{
    ...
}
```

In its constructor it calls the ERC1820 registry to register itself as implementer for the interface:

```
    constructor(address _token, Timestamp _noticePeriodChannelClosure, HoprNodeSafeRegistry _safeRegistry) {
        ...
        _ERC1820_REGISTRY.setInterfaceImplementer(address(this), TOKENS_RECIPIENT_INTERFACE_HASH, address(this));
        ...
    }
```

It fails to call `_registerInterfaceForAddress()` to register the interface on itself, which would return the proper magic value when `canImplementInterfaceForAddress()` is called:

```
contract ERC1820Implementer is IERC1820Implementer {
    bytes32 private constant _ERC1820_ACCEPT_MAGIC = keccak256("ERC1820_ACCEPT_MAGIC");

    mapping(bytes32 => mapping(address => bool)) private _supportedInterfaces;

    ...
    function canImplementInterfaceForAddress(
        bytes32 interfaceHash,
        address account
    ) public view virtual override returns (bytes32) {
        return _supportedInterfaces[interfaceHash][account] ? _ERC1820_ACCEPT_MAGIC : bytes32(0x00);
    }

    ...
    function _registerInterfaceForAddress(bytes32 interfaceHash, address account) internal virtual {
        _supportedInterfaces[interfaceHash][account] = true;
    }
}
```

Impact

Discrepancy with ERC1820.

Code Snippets

https://github.com/hoprnet/hoprnet/blob/a55fa71461851d0e5d5a3cb090a5bfcc6da11fcb/ethereum/contracts/src/Channels.sol#L84

https://github.com/hoprnet/hoprnet/blob/a55fa71461851d0e5d5a3cb090a5bfcc6da11fcb/ethereum/contracts/src/Channels.sol#L248

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.9/contracts/utils/introspection/ERC1820Implementer.sol

Recommendation

Call `_registerInterfaceForAddress(TOKENS_RECIPIENT_INTERFACE_HASH, address(this))` in constructor.

Response

Acknowledge the issue. The eip actually state that `canImplementInterfaceForAddress()` must return the magic value only when the address of the implementer is different from the address for the which the interface is set.

[Info] Cryptographic implementation states using p - 1 but uses p - 2

Summary

In multiple places of `Crypto.sol` the comments state using `p - 1` to compute the inverse of an element of the field of order `p` but uses `p - 2`. To my understanding `p - 2` is the correct value to use following Fermat's little theorem `a^p = a mod p` which means `a^(p-2)= a^(-1)mod p`. That means the comment is incorrect and the code is correct.

Detail

One example in the code would be:

```
    function ecAdd(
        uint256 pX,
        uint256 pY,
        uint256 qX,
        uint256 qY,
        uint256 a
    )
        internal
        view
        returns (uint256 rx, uint256 ry)
    {
        // solhint-disable-next-line no-inline-assembly
        assembly {
            ...
            let lambda
            let toInvert
            switch and(eq(pX, qX), eq(pY, qY))
            // P == Q ?
            case true {
                // Point double
                toInvert := addmod(mulmod(2, pY, SECP256K1_BASE_FIELD_ORDER), a, SECP256K1_BASE_FIELD_ORDER) // 2 * p.y

                // compute (2 * p.y) ^ -1 using expmod precompile
                let payload := mload(0x40)
                mstore(payload, 0x20) // Length of Base
                mstore(add(payload, 0x20), 0x20) // Length of Exponent
                mstore(add(payload, 0x40), 0x20) // Length of Modulus
                mstore(add(payload, 0x60), toInvert) // Base
                mstore(add(payload, 0x80), 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2D) // p - 1  // @audit not p-1
                mstore(add(payload, 0xa0), SECP256K1_BASE_FIELD_ORDER) // Modulus
                if iszero(staticcall(not(0), 0x05, payload, 0xC0, payload, 0x20)) {
                    // 0x05 == expmod precompile
                    revert(0, 0)
                }
                ...
            }
            case false {
                ...
            }
            ...
        }
    }
```

Code Snippets

https://github.com/hoprnet/hoprnet/blob/a55fa71461851d0e5d5a3cb090a5bfcc6da11fcb/ethereum/contracts/src/Crypto.sol#L243

https://github.com/hoprnet/hoprnet/blob/a55fa71461851d0e5d5a3cb090a5bfcc6da11fcb/ethereum/contracts/src/Crypto.sol#L272

https://github.com/hoprnet/hoprnet/blob/a55fa71461851d0e5d5a3cb090a5bfcc6da11fcb/ethereum/contracts/src/Crypto.sol#L393

https://github.com/hoprnet/hoprnet/blob/a55fa71461851d0e5d5a3cb090a5bfcc6da11fcb/ethereum/contracts/src/Crypto.sol#L431

Recommendation

Make sure you indeed want to use $p - 2$ and not $p - 1$ and fix comment, or better define and use a constant.

Response

The faulty comments have been updated in commit 943c01c.

[Info] Revoking a target does not delete its capabilities

Summary

In the `HoprNodeManagementModule` contract, revoking a target does not remove its granular capabilities. If a target is added and setup with custom capabilities, removed, and added once again, it will retain its custom granular capabilities which may be unexpected to the user.

Vulnerability Detail

The function `revokeTarget()` only removes the target from the set `TargetSet`. It does not revoke `GranularPermission`:

```
contract HoprNodeManagementModule {

    function revokeTarget(address targetAddress) external onlyOwner {
        HoprCapabilityPermissions.revokeTarget(role, targetAddress);
    }
}

struct Role {
    TargetSet targets; // target addresses that can be called
    mapping(address => bool) members; // eligible caller. May be able to receive native tokens (e.g. xDAI), if set to
        // allowed
    // For CHANNELS target: capabilityKey (bytes32) => channel Id (keccak256(src, dest)) => GranularPermission
    // For TOKEN target: capabilityKey (bytes32) => pair Id (keccak256(node address, spender address)) =>
    // GranularPermission
    // For SEND target:  bytes32(0x00) => pair Id (keccak256(node address, spender address)) => GranularPermission
    mapping(bytes32 => mapping(bytes32 => GranularPermission)) capabilities;
}

library HoprCapabilityPermissions {

    function revokeTarget(Role storage role, address targetAddress) internal {
        bool result = role.targets.remove(targetAddress);
        if (result) {
            emit RevokedTarget(targetAddress);
        } else {
            revert TargetIsNotScoped();
        }
    }
}
```

Impact

If a target with custom permissions is removed and added once again, it will retain its custom permissions.

Code Snippets

https://github.com/hoprnet/hoprnet/blob/a55fa71461851d0e5d5a3cb090a5bfcc6da11fcb/ethereum/contracts/src/node-stake/permissioned-module/NodeManagementModule.sol#L222-L224

https://github.com/hoprnet/hoprnet/blob/a55fa71461851d0e5d5a3cb090a5bfcc6da11fcb/ethereum/contracts/src/node-stake/permissioned-module/CapabilityPermissions.sol#L491-L498

Recommendation

Acknowledge the issue and document the behaviour.

Response

The issue has been acknowledged and comments have been added in commit 943c01c.

[Info] ensureNodeIsSafeModuleMember() can be tricked

Summary

The function `NodeSafeRegistry.ensureNodeIsSafeModuleMember()` can be tricked to return true by a safe enabling a simple dummy module that returns true for both `isHoprNodeManagementModule()` and `isNode(address)`.

Vulnerability Detail

The function `ensureNodeIsSafeModuleMember()` relies on the modules installed by the safe which are controlled by the safe owners:

```
    function ensureNodeIsSafeModuleMember(address safeAddress, address nodeChainKeyAddress) internal view {
        // nodeChainKeyAddress must be a member of the enabled node management module
        address nextModule;
        address[] memory modules;
        // there may be many modules, loop through them. Stop at the end point of the linked list
        while (nextModule != SENTINEL_MODULES) {
            // get modules for safe
            (modules, nextModule) = IAvatar(safeAddress).getModulesPaginated(SENTINEL_MODULES, pageSize);
            for (uint256 i = 0; i < modules.length; i++) {
                if (
                    IHoprNodeManagementModule(modules[i]).isHoprNodeManagementModule()
                        && IHoprNodeManagementModule(modules[i]).isNode(nodeChainKeyAddress)
                ) {
                    return;
                }  // @audit this can be faked by installing an attack module
            }
        }

        // if nodeChainKeyAddress is not a member of a valid HoprNodeManagementModule to the safe, revert
        revert NodeNotModuleMember();
    }
```

This function is used in when registering and deregistering a node by a safe to ensure the node has the safe module endabled and the chain key address is registered as a member of the module.

Impact

I am not sure why this check is important as registering a node requires signature from the node in any case. I don't see the impact deregistering a node by a safe that had the module enabled and chain key address as a member and no longer does would have.

Code Snippets

https://github.com/hoprnet/hoprnet/blob/a55fa71461851d0e5d5a3cb090a5bfcc6da11fcb/ethereum/contracts/src/node-stake/NodeSafeRegistry.sol#L266-L286

https://github.com/hoprnet/hoprnet/blob/a55fa71461851d0e5d5a3cb090a5bfcc6da11fcb/ethereum/contracts/src/node-stake/NodeSafeRegistry.sol#L177-L189

https://github.com/hoprnet/hoprnet/blob/a55fa71461851d0e5d5a3cb090a5bfcc6da11fcb/ethereum/contracts/src/node-stake/NodeSafeRegistry.sol#L227-L258

Recommendation

Ensure that this check is not critical if manipulated by the owners of the safe and acknowledge the issue.

Response

The function has been removed in commit f6a1700.

Optimisations and miscellaneous

This part lists minor gas/code optimizations that shouldn't make the code less readable or improve overall readability. It also lists questions about unclear code segments.

AccessControl._setupRole() is deprecated

Openzeppelin deprecated `_setupRole()` in favour of `_grantRole()` (same arguments, same logic):

```
    /*
     * ...
     * NOTE: This function is deprecated in favor of {_grantRole}.
     */
    function _setupRole(bytes32 role, address account) internal virtual {
        _grantRole(role, account);
    }
```

https://github.com/hoprnet/hoprnet/blob/a55fa71461851d0e5d5a3cb090a5bfcc6da11fcb/ethereum/contracts/src/NetworkRegistry.sol#L96-L98

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/dc44c9f1a4c3b10af99492eed84f83ed244203f6/contracts/access/AccessControl.sol#L204

[Response] deprecated function replaced in commit 6334cfb.