SHERLOCK

Security Review For Symbiotic

Public Audit Contest Prepared For: **Symbiotic**
Lead Security Expert: **0x73696d616f**
Date Audited: **June 19 - July 10, 2025**
Final Commit: **cb0c4e2**

Introduction

Symbiotic Core (https://github.com/symbioticfi/core) provides a modular onchain framework for creating flexible staking solutions, including collateral choice (native tokens, restaked assets, or multi-asset), as well as reward, slashing and redistribution logic.

Symbiotic Relay serves as an extension to the Symbiotic Core that radically simplifies integrating Symbiotic's universal staking primitives and enables leveraging stake across any execution environment, expanding the design space for multichain-native decentralized protocols.

Scope

Repository: symbioticfi/middleware-sdk

Audited Commit: 57f80a92a614f4df812cd0495e3b214bc5d954ec

Final Commit: cb0c4e22963b3bc5f532b6250e6891fa077c6069

Files:

- examples/MyKeyRegistry.sol
- examples/MyNetwork.sol
- examples/MySettlement.sol
- examples/MyValSetDriver.sol
- examples/MyVotingPowerProvider.sol
- pkg/proof/circuit.go
- src/contracts/libraries/keys/KeyBlsBn254.sol
- src/contracts/libraries/keys/KeyEcdsaSecp256k1.sol
- src/contracts/libraries/sigs/SigBlsBn254.sol
- src/contracts/libraries/sigs/SigEcdsaSecp256k1.sol
- src/contracts/libraries/structs/Checkpoints.sol
- src/contracts/libraries/structs/PersistentSet.sol
- src/contracts/libraries/utils/InputNormalizer.sol
- src/contracts/libraries/utils/KeyTags.sol
- src/contracts/libraries/utils/ValSetVerifier.sol
- src/contracts/modules/base/NetworkManager.sol
- src/contracts/modules/base/OzEIP712.sol
- src/contracts/modules/base/PermissionManager.sol

- src/contracts/modules/common/permissions/OzAccessControl.sol
- src/contracts/modules/common/permissions/OzAccessManaged.sol
- src/contracts/modules/common/permissions/OzOwnable.sol
- src/contracts/modules/key-registry/KeyRegistry.sol
- src/contracts/modules/network/Network.sol
- src/contracts/modules/settlement/Settlement.sol
- src/contracts/modules/settlement/sig-verifiers/libraries/ExtraDataStorageHelper.sol
- src/contracts/modules/settlement/sig-verifiers/SigVerifierBlsBn254Simple.sol
- src/contracts/modules/settlement/sig-verifiers/SigVerifierBlsBn254ZK.sol
- src/contracts/modules/valset-driver/EpochManager.sol
- src/contracts/modules/valset-driver/ValSetDriver.sol
- src/contracts/modules/voting-power/base/VotingPowerCalcManager.sol
- src/contracts/modules/voting-power/common/voting-power-calc/EqualStakeVPCalc.sol
- src/contracts/modules/voting-power/extensions/BaseRewards.sol
- src/contracts/modules/voting-power/extensions/BaseSlashing.sol
- src/contracts/modules/voting-power/extensions/logic/BaseRewardsLogic.sol
- src/contracts/modules/voting-power/extensions/logic/BaseSlashingLogic.sol
- src/contracts/modules/voting-power/extensions/logic/OpNetVaultAutoDeployLogic.sol
- src/contracts/modules/voting-power/extensions/MultiToken.sol
- src/contracts/modules/voting-power/extensions/OperatorsBlacklist.sol
- src/contracts/modules/voting-power/extensions/OperatorsJail.sol
- src/contracts/modules/voting-power/extensions/OperatorsWhitelist.sol
- src/contracts/modules/voting-power/extensions/OperatorVaults.sol
- src/contracts/modules/voting-power/extensions/OpNetVaultAutoDeploy.sol
- src/contracts/modules/voting-power/extensions/SharedVaults.sol
- src/contracts/modules/voting-power/logic/VotingPowerProviderLogic.sol
- src/contracts/modules/voting-power/VotingPowerProvider.sol
- src/interfaces/modules/base/INetworkManager.sol
- src/interfaces/modules/base/IOzEIP712.sol

- src/interfaces/modules/base/IPermissionManager.sol
- src/interfaces/modules/common/permissions/IOzAccessControl.sol
- src/interfaces/modules/common/permissions/IOzAccessManaged.sol
- src/interfaces/modules/common/permissions/IOzOwnable.sol
- src/interfaces/modules/key-registry/IKeyRegistry.sol
- src/interfaces/modules/network/INetwork.sol
- src/interfaces/modules/network/ISetMaxNetworkLimitHook.sol
- src/interfaces/modules/settlement/ISettlement.sol
- src/interfaces/modules/settlement/sig-verifiers/ISigVerifierBlsBn254Simple.sol
- src/interfaces/modules/settlement/sig-verifiers/ISigVerifierBlsBn254ZK.sol
- src/interfaces/modules/settlement/sig-verifiers/ISigVerifier.sol
- src/interfaces/modules/settlement/sig-verifiers/zk/IVerifier.sol
- src/interfaces/modules/valset-driver/IEpochManager.sol
- src/interfaces/modules/valset-driver/IValSetDriver.sol
- src/interfaces/modules/voting-power/base/IVotingPowerCalcManager.sol
- src/interfaces/modules/voting-power/common/voting-power-calc/IEqualStakeVPCalc.sol
- src/interfaces/modules/voting-power/extensions/IBaseRewards.sol
- src/interfaces/modules/voting-power/extensions/IBaseSlashing.sol
- src/interfaces/modules/voting-power/extensions/IMultiToken.sol
- src/interfaces/modules/voting-power/extensions/IOperatorsBlacklist.sol
- src/interfaces/modules/voting-power/extensions/IOperatorsJail.sol
- src/interfaces/modules/voting-power/extensions/IOperatorsWhitelist.sol
- src/interfaces/modules/voting-power/extensions/IOperatorVaults.sol
- src/interfaces/modules/voting-power/extensions/IOpNetVaultAutoDeploy.sol
- src/interfaces/modules/voting-power/extensions/ISharedVaults.sol
- src/interfaces/modules/voting-power/IVotingPowerProvider.sol

Final Commit Hash

cb0c4e22963b3bc5f532b6250e6891fa077c6069

Findings

Each issue has an assigned severity:

- Medium 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.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues Found

High	Medium
1	7

Issues Not Fixed and Not Acknowledged

High	Medium
0	0

Security experts who found valid issues

00xJi
0x73696d616f
0xShoonya
0xapple
0xmaverick
0xpetern
Cybrid
Drynooo
Jeffy
MaCree
Mimis
Mishkat6451
PASCAL
Raihan
X0sauce
Ziusz
albahaca0000
befree3x
coin2own
francoHacker
harry
hunt1
j3x
kangaroo
katz
klaus
maigadoh
maxim371
montecristo
pashap9990
patitonar
redbeans
roadToWatsonN101
roshark
snjax
snowflake30518
themartto
vinica_boy
zark

Issue H-1: Malicious operator can alone, with any voting power smaller than quorum forge a proof

Source:
https://github.com/sherlock-audit/2025-06-symbiotic-relay-judging/issues/452

Found by

0x73696d616f, vinica_boy

Summary

A malicious operator with any voting power (considering a minimum inclusion voting power if present) smaller than the quorum can forge a proof and bypass verification, setting the next epoch header to any value, taking over the network. Firstly, in `circuit.go`, operators with keys X, Y equal to 0 are not part of the validator set hash and are skipped whenever they are last.

```
valsetHash = api.Select(
    api.And(fieldFpApi.IsZero(&circuit.ValidatorData[i].Key.X),
     ↪  fieldFpApi.IsZero(&circuit.ValidatorData[i].Key.Y)),
    valsetHash,
    valsetHashTemp,
)
```

The reason the X = 0, Y = 0 operators need to be last to be skipped, is because the `valsetHash` takes the temp value when the key is not null. Hence, if we have an empty key (0,0), followed by a non empty key, valsetHash will take the value of valsetHashTemp again, which includes the full mimc hash, which is cumulative:

```
hashAffineG1(&mimcApi, &circuit.ValidatorData[i].Key)
mimcApi.Write(circuit.ValidatorData[i].VotingPower)
valsetHashTemp := mimcApi.Sum()
```

Hence, for this to work, the operators must be sent as [OP1, ..., OPn, (0,0)], so `valsetHash` takes the hash of the set up until OPn. It needs to exclude the (0,0) key from the validator set because the MIMC hash is checked against the real hash, which doesn't contain this (0,0) key.

Now, the (0,0) key is not a real operator, so their voting power contribution must be 0. However, it's actually possible to set any voting power (up until var size constraints), exceeding the quorum, of this fake (0,0) validator, and the proof still goes through.

The `IsNonSigner` flag is set to false of this (0,0) validator, so the voting power counts. As a result, their (0,0) key is also added to the signing aggregated key. However, the null (0,0) key point property is that its addition to the aggregate key has no effect, which means that effectively no signature is required from the null (0,0) key. Thus, having validators [OP1, ..., OPn, (0,0)], with an aggregated signature of validators 1 to n, will pass the signature verification.

This effectively means that any operator with any minimal voting power can add this (0,0) operator with a voting power that exceeds the quorum and lets the message go through. As a result, they can manipulate whatever data they want and take full control of the network, more precisely the valSetHeader for the next epoch in `Settlement.sol`, fully compromising the network.

Root Cause

In `circuit.go:101`, the voting power of an operator with null key must be null.

Internal Pre-conditions

None

External Pre-conditions

None

Attack Path

1. Operator with 1 voting power (or any minimal amount) calls `Settlement::commitValSetHeader()` with a malicious header for next epoch to compromise the network. They send a proof with only them as signer, all other operators are non signers and add at the end a null operator (0,0) with voting power bigger than the quorum.

Impact

Network is compromised and attacker can do whatever they want. Operator role is permissionless for some networks (depending on extensions) and even if it wasn't, they would still be able to completely bypass the quorum which is high severity.

PoC

Change `proof_test.go` to:

```
func genValset(numValidators int, nonSigners []int) []ValidatorData {
    valset := make([]ValidatorData, numValidators)
    for i := 0; i < numValidators; i++ {
        pk := big.NewInt(int64(i + 1000000000000000000))
        valset[i].PrivateKey = pk
```

```
        valset[i].Key = getPubkeyG1(pk)
        valset[i].KeyG2 = getPubkeyG2(pk)
        valset[i].VotingPower = big.NewInt(1) // this has to be the real voting
         ↪  power of the rest of the set but it doesn't really matter for this poc
         ↪  as they won't sign. Only validator i == 0 is signing.
        valset[i].IsNonSigner = false
        if i != 0 {
            valset[i].IsNonSigner = true
        }
    }

    for _, nonSigner := range nonSigners {
        valset[nonSigner].IsNonSigner = true
    }

    return valset
}
```

Change `helpers.go` to the following. Note that n is 11 (set has length 10) to add the null key (0,0).

```
func NormalizeValset(valset []ValidatorData) []ValidatorData {
    // Sort validators by key in ascending order
    sort.Slice(valset, func(i, j int) bool {
        // Compare keys (lower first)
        return valset[i].Key.X.Cmp(&valset[j].Key.X) < 0 ||
         ↪  valset[i].Key.Y.Cmp(&valset[j].Key.Y) < 0
    })
    n := 11 //getOptimalN(len(valset))
    normalizedValset := make([]ValidatorData, n)
    for i := range n {
        if i < len(valset) {
            normalizedValset[i] = valset[i]
        } else {
            zeroPoint := new(bn254.G1Affine)
            zeroPoint.SetInfinity()
            zeroPointG2 := new(bn254.G2Affine)
            zeroPointG2.SetInfinity()
            normalizedValset[i] = ValidatorData{PrivateKey: big.NewInt(0), Key:
             ↪  *zeroPoint, KeyG2: *zeroPointG2, VotingPower:
             ↪  big.NewInt(30000000000000 * 10), IsNonSigner: false}
        }
    }
    return normalizedValset
}
```

In `proof.go` set `MaxValidators = []int{11}`. If the 3 verifiers 10, 100, 1000 were used this wouldn't be needed but the current commit defaults to 10 only and it's easier to allow 11 for this POC.

Go to `pkg/proof` and run `go test .`, it passes with 1 real voting power.

Mitigation

If the key is null, voting power must be null.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/symbioticfi/relay-contracts/pull/33/commits/8d0d70bd47afa5029a0cf72ef82c49754fd2201c

Issue M-1: Attacker will manipulate voting power calculations as `getOperatorVotingPower()` and `getOperatorVotingPowerAt()` functions lack vault validation

Source: https://github.com/sherlock-audit/2025-06-symbiotic-relay-judging/issues/196

This issue has been acknowledged by the team but won't be fixed at this time.

Found by

0xapple, 0xmaverick, Drynooo, Jeffy, PASCAL, Ziusz, katz, patitonar, roshark, zark

Summary

The missing vault validation in `VotingPowerProvider::getOperatorVotingPower()` and `VotingPowerProvider::getOperatorVotingPowerAt()` functions that receives the `vault` external parameter will cause incorrect voting power calculations as an attacker can provide unregistered or invalid vault addresses to gain unauthorized voting influence

Root Cause

In `VotingPowerProviderLogic::getOperatorVotingPower(address operator, address vault, bytes memory extraData)` and `VotingPowerProviderLogic::getOperatorVotingPowerAt(address operator, address vault, bytes memory extraData, uint48 timestamp, bytes memory hints)` the `vault` parameter is not validated to be a registered vault before processing voting power calculations.

https://github.com/sherlock-audit/2025-06-symbiotic-relay/blob/main/middleware-sdk/src/contracts/modules/voting-power/VotingPowerProvider.sol#L273-L295

The functions only validate that the vault's collateral token is registered via `isTokenRegistered(IVault(vault).collateral())`, but they do not verify that the vault itself is properly registered in the system.

A vault can be unregistered by:

- SharedVaults::unregisterSharedVault()
- OperatorVaults::unregisterOperatorVault()

Internal Pre-conditions

A vault with valid collateral token exists but is not registered in the `VotingPowerProvider` contract, OR the vault was registered but unregistered later.

External Pre-conditions

N/A

Attack Path

1. Attacker identifies an unregistered vault that has a registered collateral token
2. Attacker calls some contract that calls `VotingPowerProvider::getOperatorVotingPower()` with the unregistered vault address
3. Function calculates voting power using the retrieved stake and returns it as valid voting power
4. Attacker uses this voting power, gaining influence they should not have

Impact

The protocol using `VotingPowerProvider` suffers incorrect voting power calculations as attackers gain unauthorized voting influence through unregistered vaults.

PoC

No response

Mitigation

Add vault validation to both `getOperatorVotingPower()` and `getOperatorVotingPowerAt()` functions by checking if the vault is registered before processing voting power calculations:

```
// Add vault validation
if (!isSharedVaultRegistered(vault) && !isOperatorVaultRegistered(vault)) {
    return 0;
}
```

Issue M-2: Enabling the whitelist can grant a malicious operator a temporary whitelisted status

Source: https://github.com/sherlock-audit/2025-06-symbiotic-relay-judging/issues/361

Found by

Ziusz, klaus, maigadoh, zark

Summary

N/A

Root Cause

The root cause of this vulnerability is that in order for `unwhitelist` call to success, the operator must be whitelisted.

```
function _unwhitelistOperator(
    address operator
) internal virtual {
    if (!isOperatorWhitelisted(operator)) {
        revert OperatorsWhitelist_OperatorNotWhitelisted();
    }
    _getOperatorsWhitelistStorage()._whitelisted[operator] = false;
    if (isWhitelistEnabled() && isOperatorRegistered(operator)) {
        _unregisterOperator(operator);
    }

    emit UnwhitelistOperator(operator);
}
```

Link to code

Internal Pre-conditions

N/A

External Pre-conditions

N/A

Attack Path

1. Whitelist mode is off.
2. Owner enables whitelist mode.
3. Malicious operator front runs owner by registering.
4. Now, if owner wants to unregister him, he must grant him the whitelist role and then unwhitelist him.
5. This would mean that the malicious operator would be granted the whitelist role for at least one block.
6. Since this is an SDK, it is very possible that the whitelist role will be connected with further abilities.

Impact

Malicious actor can frontrun the whitelist mode enabled and in this way force the owners to whitelist him (in order to unwhitelist him) for at least one block. So the malicious actor can gain any abilities that a whitelisted operator can do without this being the goal of the owners of the network. If we take into account that the owner of the `Middleware` may be the `Network` which has time delays in the calls, this worsen the situation.

PoC

N/A

Mitigation

Allow the `unwhitelist` of an operator even if he is not whitelisted but the whitelist mode is on.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/symbioticfi/relay-contracts/pull/33/commits/e655574ab282e2c42065166f1d6f8ef4a5004665

Issue M-3: `autoDeployedVault` mapping is not updated after `unregisterOperatorVault`

Source:
https://github.com/sherlock-audit/2025-06-symbiotic-relay-judging/issues/362

Found by

0xShoonya, snjax, zark

Summary

`unregisterOperatorVault` deletes a vault in `VotingPowerProvider` but it doesn't check the `_autoDeployedVault` mapping in `OpNetVaultAutoDeploy` extension, so an operator whose auto-deployed vault is removed remains flagged as owning that vault and can neither receive a fresh auto deployment nor rely on the old address.

Root Cause

The vault unregistration in `VotingPowerProviderLogic` updates only `_operatorVaults` and `_allOperatorVaults`. When the vault had originally been created by `OpNetVaultAutoDeploy.createVault`, its address was also stored in `_autoDeployedVault[operator]`. Because `unregisterOperatorVault` is not overridden in the extension and holds no hook back into it, that mapping key is left unchanged.

```
// OpNetVaultAutoDeploy.sol
    function _registerOperatorImpl(
        address operator
    ) internal virtual override {
        super._registerOperatorImpl(operator);
        if (isAutoDeployEnabled() && getAutoDeployedVault(operator) == address(0)) {
            (address vault, address delegator,) =
            ↪  OpNetVaultAutoDeployLogic.createVault(operator);
            _registerOperatorVault(operator, vault);
            if (isSetMaxNetworkLimitHookEnabled()) {
                ISetMaxNetworkLimitHook(NETWORK()).setMaxNetworkLimit(
                    delegator, SUBNETWORK_IDENTIFIER(), type(uint256).max
                );
            }
        }
    }
```

[Link to code]

```
// VotingPowerProviderLogic.sol
    function unregisterOperatorVault(address operator, address vault) public {
        IVotingPowerProvider.VotingPowerProviderStorage storage $ =
        ↪  _getVotingPowerProviderStorage();
        if (!$._operatorVaults[operator].remove(Time.timestamp(), vault)) {
            revert IVotingPowerProvider.VotingPowerProvider_OperatorVaultNotRegiste⌋
            ↪  red();
        }
        $._allOperatorVaults.remove(Time.timestamp(), vault);

        emit IVotingPowerProvider.UnregisterOperatorVault(operator, vault);
    }
```

Link to code

So after all, calls to `getAutoDeployedVault(operator)` therefore return a vault that the core module now considers unregistered, and `_registerOperatorImpl` skips auto-deployment because it sees a non-zero pointer.

Internal Pre-conditions

Auto deployment must be enabled, the configuration valid and an operator has previously registered, triggering `createVault` and populating `_autoDeployedVault`.

External Pre-conditions

`unregisterOperatorVault(operator, vault)` being invoked for that vault created with auto deployment.

Attack Path

1. An operator registers while auto-deployment is enabled, causing `createVault` to store vault in `_autoDeployedVault[operator]` and register it in the provider storage.
2. A caller executes `unregisterOperatorVault(operator, vault)`, which removes the vault from `_operatorVaults` and `_allOperatorVaults` but leaves `_autoDeployedVault [operator]` unchanged.
3. Because `getAutoDeployedVault(operator)` still returns a non-zero address, the system assumes the operator already has a vault and skips creating a new one, while the active vault lists no longer include that old address. Also, if someone queries the `getAutoDeployedVault` of the operator, an incorrect unregistered vault would be returned.

Impact

The impact of this issue is that the `OpNetVaultAutoDeploy::getAutoDeployedVault` of the operator would return an incorrect unregistered vault while in the same time the operator would never be able to create a new auto deployed vault.

PoC

N/A

Mitigation

In order to mitigate this code "asymmetry", it is recommended to overwrite the `_unregisterOperator` in `OpNetVaultAutoDeploy` (as it is done with the `_registerOperatorImpl`) and if the operator and the vaults matches in `_autoDeployedVault`, then unregister it from there as well.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/symbioticfi/relay-contracts/pull/33/commits/84fa428dbd8b113c56f92bf91481997d6eedc288

Issue M-4: Most KeyRegistry, VotingPowerProvider functions can be DoSed

Source:
https://github.com/sherlock-audit/2025-06-symbiotic-relay-judging/issues/403

This issue has been acknowledged by the team but won't be fixed at this time.

Found by

00xJi, 0x73696d616f, 0xShoonya, 0xpetern, Cybrid, Drynooo, MaCree, Mimis, Mishkat6451, Raihan, X0sauce, Ziusz, albahaca0000, befree3x, coin2own, francoHacker, harry, hunt1, j3x, kangaroo, klaus, maigadoh, maxim371, montecristo, pashap9990, redbeans, roadToWatsonN101, snowflake30518, themartto, zark

Summary

`VotingPowerProvider::registerOperator()` is permissionless, so anybody can call it with any number of accounts to DoS all functions that gather operator information, such as `getOperatorsAt()`, `getOperators()`. Same for the vaults of the operators `getOperatorVaults()`, which can be registered via `_registerOperatorVault()` (part of the `OperatorVaults` extension or the auto deployment `OpNetVaultAutoDeploy` extension).

The `KeyRegistry` also has this issue, as setKey() is permissionless, and getKeys() will be DoSed as anybody can create any number of ethereum addresses as operators and keys to DoS the function.

As part of the SDK, these functions are key and should be paginated. They will be DoSed on chain due to the gas limit and off chain due to rpc timeout limits.

Root Cause

In `KeyRegistry`, `VotingPowerProvider`, there is no limit on some array elements nor pagination.

Internal Pre-conditions

None.

External Pre-conditions

None.

Attack Path

1. Attacker creates multiple wallets to DoS the function calls.

Impact

Key view functions for key, operators are DoSed.

PoC

No response

Mitigation

Add pagination.

Issue M-5: Changing the epoch duration will completely break the vault and the slashers

Source: https://github.com/sherlock-audit/2025-06-symbiotic-relay-judging/issues/410

This issue has been acknowledged by the team but won't be fixed at this time.

Found by

0x73696d616f, zark

Summary

The vault and slashers in scope of the protocol in `OpNetVaultAutoDeploy` are `BASE_VAULT_VERSION` and `TOKENIZED_VAULT_VERSION`, SlasherType.INSTANT and SlasherType.VETO.

The issue is that the epoch duration can not be updated in these contracts, they always use the same value. As a result, whenever it is changed in the epoch manager via `EpochManager::setEpochDuration()`, it will completely ruin the vault's staking and slashing mechanism, as they rely on the epoch duration. For example, the instant slasher requires the slashing to be within 1 epoch duration, but if the epoch duration is now bigger/smaller, slashing will fail for a significant period of time. Slashing also has several checks which don't allow slashing whenever the operator, token or vault is not registered in that epoch, which can make slashing impossible via `slashVault()` in case of epoch duration change.

Additionally, the slashing window must be bigger than the epoch duration, but the slasher's and vault's epoch duration can't be changed, so this will also not work properly. The same happens for the veto duration of the veto slasher.

`Vault::onSlash()` requires the epoch to be no older than `currentEpoch_ - 1`, but a different duration means this will fail. For example, if the duration is halved, calling at duration / 2 + 1 after the current epoch will revert as this is already 2 epochs in the past.

Root Cause

In `EpochManager:115`, setting a new epoch duration will always break the vault and their slasher.

Internal Pre-conditions

None

External Pre-conditions

None

Attack Path

1. `EpochManager::setEpochDuration()` is called.

Impact

Epoch duration is out of sync between the vaults, slashers and the epoch manager, so slashing will fail or be impossible.

PoC

No response

Mitigation

Non trivial.

Issue M-6: BlsBn254 is not available in certain chains due to hardcoded gas limit

Source:
https://github.com/sherlock-audit/2025-06-symbiotic-relay-judging/issues/422

Found by

0x73696d616f, Drynooo, klaus

Summary

EcParing precompile always fails on certain chains due to the hardcoded gas limit, so BlsBn254 is not available on certain chains.

Root Cause

- contracts/libraries/sigs/SigBlsBn254.sol#L39
- contracts/libraries/sigs/SigBlsBn254.sol#L65

`SigBlsBn254.verify` uses the hardcoded `PAIRING_CHECK_GAS_LIMIT(= 120_000)` when calling `BN254.safePairing`. This is the gas cost of EcParing based on `34000 * k + 45000` when k=2, as defined in EIP-1108.

```
@>  uint256 internal constant PAIRING_CHECK_GAS_LIMIT = 120_000;

    function verify(
        bytes memory keyBytes,
        bytes memory message,
        bytes memory signature,
        bytes memory extraData
    ) internal view returns (bool) {
        ...

        (bool success, bool result) = BN254.safePairing(
            signatureG1.plus(keyG1.scalar_mul(alpha)),
            BN254.negGeneratorG2(),
            messageG1.plus(BN254.generatorG1().scalar_mul(alpha)),
            keyG2,
@>          PAIRING_CHECK_GAS_LIMIT
        );
        return success && result;
    }

    ////////////////////////////
```

```
    // BN254.safePairing
    function safePairing(
        G1Point memory a1,
        G2Point memory a2,
        G1Point memory b1,
        G2Point memory b2,
@>      uint256 pairingGas
    ) internal view returns (bool, bool) {
        ...

        assembly {
@>          success := staticcall(pairingGas, 8, input, mul(12, 0x20), out, 0x20)
        }
        ...
    }
```

The gas cost for precompile may change or vary by chain. For example, ZKSync (included in the chain to be deployed) updated the EcAdd, EcMul, and EcPairing precompiles and changed the gas cost in the ZIP-11. V28 Precompile Upgrade upgrade in May 2025.

The above code cause problems in ZKSync. The following is the new EcPairing code updated at ZKSync V28. The gas cost is calculated via `80000 * k`. When k=2, the required gas is 160_000, which is higher than the `PAIRING_CHECK_GAS_LIMIT`. EcParing always fails when gas is insufficient, so the BlsBn254 signature check in ZKSync will always fail.

Internal Pre-conditions

1. Use BlsBn254 for signing.

External Pre-conditions

1. The EcPairing precompile gas cost at the deployed chain does not follow EIP-1108.

Attack Path

The issue is caused by a bug.

Impact

BlsBn254 is not available on some chains.

PoC

The ZIP-11. V28 Precompile Upgrade is only available on the ZKSync mainnet (I don't think it's applied to the testnet), and it is not reproducible with the foundry fork test, so you need to test it directly on the mainnet.

Deploy the following code to the ZKSync Era mainnet and run it to see the gas cost. If you put the correct `input` in the `verify` function and experiment with incrementing `pairingGas` from 120_000, you will see that at around 161_000, the signature verification succeeds with `success` and `out[0]` set to 1. This is consistent with ZKSync's `80000 * k (k = 2)`.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract Test {

    event Cost(uint256);
    event Out(bool, uint256);

    function test (uint256[12] memory input, uint256 pairingGas) public {
        (uint256 gasBefore, uint256 gasAfter, bool success, uint256 out) =
        ↪  verify(input, pairingGas);
        emit Cost(gasBefore - gasAfter);
        emit Out(success, out);
    }

    function verify (uint256[12] memory input, uint256 pairingGas) public view
    ↪  returns (uint256, uint256, bool, uint256) {
        uint256[1] memory out;
        bool success;

        uint256 gasBefore = gasleft();
        // solium-disable-next-line security/no-inline-assembly
        assembly {
            success := staticcall(pairingGas, 8, input, mul(12, 0x20), out, 0x20)
        }
        uint256 gasAfter = gasleft();
        return (gasBefore, gasAfter, success, out[0]);
    }
}
```

Use the following as the `input` parameter, created with the correct signature and key value. This is the value from the test code.

```
[
```

```
17542794946843030738197687269502130768488764040084025709702018229683082027107,
↪  21243454333462907454433938848788936660904069824545612138480299027504168819393,
↪  11559732032986387107991004021392285783925812861821192530917403151452391805634,
↪  10857046999023057135944570762232829481370756359578518086990519993285655852781,
↪  17805874995975841540914202342111839520379459829704422454583296818431106115052,
↪  13392588948715843804641432497768002650278120570034223513918757245338268106653,
↪  6152845192698230377440204073057238033424791113774748884801148069022325658846,
↪  13760496706863554449593094343798996929546352261485265365831743695186162488392,
↪  10168917783125035928329339378130255896597415372015030444874307897081997728948,
↪  15338339620195733484325031668011173090672215643291231872576243132177438055881,
↪  10104509023153927337647655231628382133731833653099790728025758502925918550767,
↪  13448048280709447326318302930315758447948705837394892676501482696723894570897
]
```

Mitigation

The gas cost required to call Precompile may be changed in the future and can differ between chains. Therefore, instead of using `PAIRING_CHECK_GAS_LIMIT`, you should use a variable that can be set by an administrator.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/symbioticfi/relay-contracts/pull/33/commits/cb0c4e22963b3bc5f532b6250e6891fa077c6069

Issue M-7: A malicious operator will control consensus without risking stake (stake-exit lag exploit)

Source:
https://github.com/sherlock-audit/2025-06-symbiotic-relay-judging/issues/446

This issue has been acknowledged by the team but won't be fixed at this time.

Found by

0x73696d616f, hunt1, montecristo

Summary

The non-atomic nature of `setSigVerifier` and `commitValSetHeader` will cause a potential loss of security guarantees for networks as a malicious operator can manipulate validator sets after unstaking their funds, avoiding slashing penalties.

Root Cause

In `Settlement` contract the design choice to separate `setSigVerifier` and `commitValSetHeader` functions (latter being public) is a mistake as it allows for a time gap between validator selection and header commitment. This creates a window where an operator can withdraw their stake while still maintaining their voting power.

Specifically, `commitValSetHeader` in `Settlement` is marked as public:

https://github.com/sherlock-audit/2025-06-symbiotic-relay/blob/main/middleware-sdk/src/contracts/modules/settlement/Settlement.sol#L292-L324

While `setSigVerifier` is a separate function:

https://github.com/sherlock-audit/2025-06-symbiotic-relay/blob/main/middleware-sdk/src/contracts/modules/settlement/Settlement.sol#L267-L275

Attack Path

Epoch 1:

1. A malicious operator deposits a large amount of stake to their vault, ensuring it exceeds the `quorumThreshold` for voting power.
2. The off-chain relay calculates voting powers, creates a `sigVerifier` using validators belonging to the malicious operator, and calls `setSigVerifier` to assign these validators for the next epoch.

3. Near the end of Epoch, the operator calls `withdraw` on their vault to initiate the withdrawal of their stake.

Epoch 2:

1. The operator immediately calls `claim` on their vault to retrieve all their withdrawn stake, effectively removing their financial exposure.
2. Despite having withdrawn their stake, their voting power is active in the system's state for a short time-span.
3. The operator can craft their own proof using their validators' private keys, which will pass verification since they had enough voting power.
4. When the operator submits this crafted proof to `commitValSetHeader`, it will be accepted by the system, allowing the operator to control consensus without any stake at risk.

Impact

The network suffers a complete compromise of its security model. The malicious operator can perform any validator action (such as approving invalid transactions or censoring valid ones) without having any stake at risk of being slashed. This fundamentally breaks the economic security assumptions of the protocol, which relies on validators having skin in the game to behave honestly.

Mitigation

Redesign the `commitValSetHeader` function to be internal (`_commitValSetHeader`), and create a new public function that handles both setting the signature verifier (optional) and committing the header in a single atomic transaction.

Alternatively apply `checkPermission` to `commitValSetHeader` so it can only be executed by the Network's relay service.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.