



SMART CONTRACT SECURITY AUDIT

Clique

SigDispersal Airdrop Contract

CLIENT

Clique

LANGUAGE

Solidity

REVIEW

July 2025

DOCUMENT

Document information

Project	Clique, SigDispersal Airdrop Contract
Client	Clique
Platform	Not recorded in the original report
Language	Solidity
Repository	Not recorded in the original report
Commit	Not recorded in the original report
Methods	Not recorded in the original report
Review dates	3/06/2025 to 05/07/2025 (as written in the original)
Report date	4 July 2025

Document history

VERSION	DATE	CHANGE
1.0	4 July 2025	Original report issued View original report
1.1	October 2026	Reformatted into the Safe Edges standard template. Findings unchanged

This report was reformatted into the Safe Edges standard template in October 2026. Findings, severities, recommendations and code references are unchanged from the original report, which is preserved unchanged in the [Safe Edges Audit Portfolio on GitHub](#).

CONTENTS

Contents

01	Executive summary	3
02	Scope	4
03	Methodology	5
04	Severity and status	6
05	Findings summary	7
06	Detailed findings	8
	H-01 Missing batchEnabled Check Prior to Claiming in claim Function	8
	L-01 Mistakenly Sent ETH to claim Function Will Be Stuck Forever	9
	I-01 Reorder Checks to Save Gas	9
	I-02 Lack of Deadline or Expiry Mechanism for Airdrop Batches	10
	I-03 Missing Zero Address Check	11
	Disclaimer	12

Executive summary

About Clique

SigDispersal is Clique's signature-based airdrop contract. Users claim allocations from batches using a signature from an authorised signer, and tokens are transferred to them from a vault.

Summary of the review

Safe Edges reviewed Clique's SigDispersal airdrop contract. Five issues were identified: one High, one Low and three Informational. The High issue was a missing batchEnabled check in claim, which let users claim from batches the admin had disabled. All five were resolved.

Findings by severity



Findings by status

STATUS	ISSUES
Resolved	5

Scope

Repository	Not recorded in the original report
Commit	Not recorded in the original report
Platform	Not recorded in the original report
Language	Solidity

IN-SCOPE FILES

SigDispersal.sol

File hash: 3caeb0e4aeb0afc085af1cf713ed5f1f13b539b7689db9ea10a8b73619325c6

Only the code at the commit above was reviewed. Changes made after this commit are not covered by this report.

Methodology

Methods recorded for this engagement: Not recorded in the original report.

Manual review

Auditors read the in-scope code line by line, tracing control flow, state changes and trust boundaries against the intended behaviour described by the client.

Threat modelling

We map privileged roles, external dependencies and value flows, then test the assumptions the code makes about each of them.

Functional and automated testing

Where tooling exists for the language and chain, we run static analysis and targeted tests to confirm suspected issues and check edge cases.

Fix review

After the client addresses the findings, we review the updated code and record the final status of each issue.

Severity and status

Severity levels

SEVERITY	DEFINITION
Critical	Direct loss or permanent lock of funds, or full compromise of the protocol, with few or no preconditions. Must be fixed before deployment.
High	Significant loss of funds, broken core functionality or manipulation of critical state under realistic conditions. Should be fixed before deployment.
Medium	Limited loss or degraded functionality, or issues that need specific conditions or privileged access to exploit. Should be fixed.
Low	Minor impact, low likelihood, or deviations from best practice. Fix when practical.
Informational	Code quality, documentation, gas and style observations with no direct security impact.

Status definitions

STATUS	DEFINITION
Resolved	The issue was fixed and the fix was verified.
Partially resolved	The risk was reduced but not fully removed.
Acknowledged	The client accepted the issue without changing the code.
Open	The issue was not addressed at the time of the report.
Not recorded	The original report did not record a resolution status for this issue.

Findings summary

ID	TITLE	SEVERITY	STATUS
H-01	Missing batchEnabled Check Prior to Claiming in claim Function	High	● Resolved
L-01	Mistakenly Sent ETH to claim Function Will Be Stuck Forever	Low	● Resolved
I-01	Reorder Checks to Save Gas	Informational	● Resolved
I-02	Lack of Deadline or Expiry Mechanism for Airdrop Batches	Informational	● Resolved
I-03	Missing Zero Address Check	Informational	● Resolved

Detailed findings

H-01 Missing batchEnabled Check Prior to Claiming in claim Function

Severity **High** Status ● Resolved

DESCRIPTION

The claim function is designed to allow users to claim their airdropped tokens. However, it does not include a check for the batchEnabled mapping, which is intended to verify whether a particular airdrop batch is currently active or not. This omission may allow users to successfully claim tokens from a disabled batch, provided they possess a valid signature from the authorized signer. As a result, the batchEnabled mapping becomes redundant (dead code) and offers no protection.

We recommend using batchId as the key in the batchEnabled mapping instead of root. This will make the batch enable/disable functionality meaningful and ensure proper validation before claims.

IMPACT

High – Even if the PROJECT_ADMIN disables a batch, users can still claim tokens using valid signatures, rendering batch control ineffective.

MITIGATION

Add a check to ensure the batch is enabled before proceeding with the claim. Suggested code change:

```
function claim(bytes calldata _signature, bytes32 _batch, uint256 _amount, address _onBehalfOf)
    external
    payable
    whenActive
{
    + if (!batchEnabled[_batch]) {
    +     revert NotActive();
    + }
    address _signer = signer;
    _signatureCheck(_amount, _onBehalfOf, _signature, _signer, _batch);
    uint256 _claimHeight = claimed[_batch][_onBehalfOf];
    if (_claimHeight > 0) {
        revert AlreadyClaimed();
    }
    claimed[_batch][_onBehalfOf] = block.number;
    IERC20(token).safeTransferFrom(vault, _onBehalfOf, _amount);
    emit AirdropClaimed(_batch, _onBehalfOf, _amount);
}
```

L-01 Mistakenly Sent ETH to claim Function Will Be Stuck Forever

Severity **Low** Status ● Resolved

DESCRIPTION

The claim function is marked payable, but the function does not use or handle any native ETH. Moreover, the contract lacks a `receive()` or `fallback()` function. As a result, if users accidentally send ETH with their claim, the funds will become permanently stuck in the contract.

IMPACT

Low – Affects users only if they mistakenly send ETH during a claim call.

MITIGATION

Remove the payable modifier from the claim function, since it serves no purpose.

```
function claim(bytes calldata _signature, bytes32 _batch, uint256 _amount, address _onBehalfOf)
    external
-   payable
    whenActive
{ ... }
```

I-01 Reorder Checks to Save Gas

Severity **Informational** Status ● Resolved

DESCRIPTION

In the claim function, `_signatureCheck` is called before verifying whether the user has already claimed the airdrop. This means unnecessary gas is consumed on signature verification even when the user is ineligible due to an existing claim.

IMPACT

Informational, but relevant for gas optimization.

MITIGATION

Reorder the checks to first validate the claim eligibility, then verify the signature:

```
function claim(bytes calldata _signature, bytes32 _batch, uint256 _amount, address _onBehalfOf)
    external
    whenActive
{
+   uint256 _claimHeight = claimed[_batch][_onBehalfOf];
+   if (_claimHeight > 0) {
+       revert AlreadyClaimed();
+   }
    address _signer = signer;
    _signatureCheck(_amount, _onBehalfOf, _signature, _signer, _batch);
-   uint256 _claimHeight = claimed[_batch][_onBehalfOf];
-   if (_claimHeight > 0) {
-       revert AlreadyClaimed();
-   }
    claimed[_batch][_onBehalfOf] = block.number;
    IERC20(token).safeTransferFrom(vault, _onBehalfOf, _amount);
    emit AirdropClaimed(_batch, _onBehalfOf, _amount);
}
```

I-02 Lack of Deadline or Expiry Mechanism for Airdrop Batches

Severity **Informational** Status ● Resolved

DESCRIPTION

Currently, once a batch is enabled, it remains active indefinitely unless explicitly disabled by the PROJECT_ADMIN. However, the disabling process does not impact the claim function unless the missing batchEnabled check (see H-01) is implemented.

IMPACT

Informational, but affects overall token control and lifecycle.

MITIGATION

Introduce a deadline or expiration mechanism per batch (e.g., using a batchExpiry mapping), and integrate it into the claim logic.

I-03 Missing Zero Address Check

Severity Informational Status ● Resolved

DESCRIPTION

There is no check to ensure that signer, vault is not the zero address. This could lead to unintended behavior or misuse,

IMPACT

Informational, but recommended for robust contract safety.

MITIGATION

Add a validation check like:

```
if (signer == address(0)) {  
    revert ZeroAddress();  
}
```

Disclaimer

This report is based on the code at the commit listed in the Scope section and on information the client provided during the review. It reflects Safe Edges' assessment at the time of the review and is not a guarantee that the code is free of vulnerabilities.

Changes made after the reviewed commit, deployment configuration, off-chain components and third-party dependencies are outside the scope of this report unless stated otherwise.

Security is ongoing. We recommend continued testing, monitoring and bug bounty coverage alongside any audit.

This report is not investment advice and is not an endorsement of the project, its team or any token.



Safe Edges is a Web3 security firm and official security partner for the Fuel, ENI and XI EcoChain ecosystems. We work across smart contract audits, Rust and WASM reviews, ZK circuit review, AI and MCP security, and penetration testing.

200+

PROJECTS AUDITED

\$4B+

TVL SECURED

0

POST-AUDIT HACKS