



SMART CONTRACT SECURITY AUDIT

Clique

SVM Merkle Distributor

CLIENT

Clique

PLATFORM

SVM

LANGUAGE

Rust (Anchor)

REVIEW

November 2025

DOCUMENT

Document information

Project	Clique, SVM Merkle Distributor
Client	Clique
Platform	SVM
Language	Rust (Anchor)
Repository	Not recorded in the original report
Commit	Not recorded in the original report
Methods	Not recorded in the original report
Assessed	November 2025

Document history

VERSION	DATE	CHANGE
1.0	November 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 Claimer Can Be Overcharged During Claim	8
	M-01 Base Account Constraint Prevents Protocol Integration	8
	M-02 Accepting Mints With Freeze Authority Enables Post-Claim Freezes	9
	M-03 Missing Chain Identifier Enables Cross-Chain Signature Replay Attacks	10
	M-04 Mutable Mint Allows Token Swap After Root Publication	12
	L-01 No Fee Caps Allow Unbounded Fee Extraction	12
	L-02 Misleading comment	13
	L-03 set_mint Lacks Mint Account Validation and Legacy Token Enforcement	14
	L-04 Dynamic Recipient Toggle Bricks Pending Claims	14
	L-05 Missing event emission for a critical state update function	15
	I-01 Unused Imports	16
	I-02 Missing State Change Guards	17
	I-03 Vault Parameter Should Use SystemAccount Type	17
	I-04 Misspelled Helper Function receover_signer Hurts Clarity	18
	Disclaimer	19

Executive summary

About Clique

Clique's merkle distributor program lets users claim token allocations on SVM chains using Merkle proofs and a signer's signature, with configurable claim fees.

Summary of the review

Safe Edges reviewed Clique's merkle distributor program for SVM chains, written in Rust with Anchor, in November 2025. Fourteen issues were identified: one High, four Medium, five Low and four Informational. The High issue charged the claim fee on the gross allocation rather than the claimable amount, so incremental claims were overcharged. All fourteen were fixed.

Findings by severity



Findings by status

STATUS	ISSUES
● Resolved	14

Scope

Repository	Not recorded in the original report
------------	-------------------------------------

Commit	Not recorded in the original report
--------	-------------------------------------

Platform	SVM
----------	-----

Language	Rust (Anchor)
----------	---------------

IN-SCOPE FILES

```
programs/merkle-distributor
```

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	Claimer Can Be Overcharged During Claim	High	● Resolved
M-01	Base Account Constraint Prevents Protocol Integration	Medium	● Resolved
M-02	Accepting Mints With Freeze Authority Enables Post-Claim Freezes	Medium	● Resolved
M-03	Missing Chain Identifier Enables Cross-Chain Signature Replay Attacks	Medium	● Resolved
M-04	Mutable Mint Allows Token Swap After Root Publication	Medium	● Resolved
L-01	No Fee Caps Allow Unbounded Fee Extraction	Low	● Resolved
L-02	Misleading comment	Low	● Resolved
L-03	set_mint Lacks Mint Account Validation and Legacy Token Enforcement	Low	● Resolved
L-04	Dynamic Recipient Toggle Bricks Pending Claims	Low	● Resolved
L-05	Missing event emission for a critical state update function	Low	● Resolved
I-01	Unused Imports	Informational	● Resolved
I-02	Missing State Change Guards	Informational	● Resolved
I-03	Vault Parameter Should Use SystemAccount Type	Informational	● Resolved
I-04	Misspelled Helper Function receover_signer Hurts Clarity	Informational	● Resolved

Detailed findings

H-01 Claimer Can Be Overcharged During Claim

Severity **High** Status ● Resolved

DESCRIPTION

In the claim function, users can redeem tokens they are entitled to via Merkle proof verification and signature validation. The function is designed to deduct a service fee from the claim process by calling `calculate_fee(amount, &ctx.accounts.fee_config)?`. However, this fee is computed based on the gross amount parameter rather than the actual claimable value (`claimable = released - claim_status.claimed`).

As a result, if the user has previously claimed a portion of their airdrop, subsequent claims will still be charged the full fee again, even though they are only receiving the smaller, remaining portion.

IMPACT

A claimer performing incremental claims is overcharged on every subsequent claim, paying a disproportionately higher fee than intended.

RECOMMENDED MITIGATION

Calculate the fee using the actual claimable amount rather than the gross amount.

M-01 Base Account Constraint Prevents Protocol Integration

Severity **Medium** Status ● Resolved

DESCRIPTION

The base account in the Claim instruction is constrained to SystemAccount, which restricts it to EOAs owned by the System Program. This prevents using PDAs, multisigs, or governance-controlled accounts as the distributor controller and blocks integrations where another protocol or DAO should own and operate the distributor. The account is also marked mut and is the recipient of system transfers, further limiting composability.

IMPACT

Reduces composability and decentralization by preventing PDA- or governance-controlled distributors, limiting integrations with protocols/DAOs and forcing EOA-controlled deployments.

RECOMMENDED MITIGATION

Allow any account type for base by switching to `AccountInfo<'info>` (or a more flexible account type). If specific authority constraints are needed (e.g., PDA seeds or multisig checks), enforce them explicitly rather than via SystemAccount.

```
/// CHECK: Base account that controls the distributor (PDA, multisig, or EOA)
#[account(mut)]
pub base: AccountInfo<'info>,
```

M-02 Accepting Mints With Freeze Authority Enables Post-Claim Freezes

Severity **Medium** Status ● Resolved

DESCRIPTION

If the distributor uses a mint with a configured freeze authority, that authority can freeze recipients token accounts (including ATAs) after claims, creating a denial-of-service and custody risk outside the programs control. The program does not restrict or validate the mints freeze_authority at initialization, and set_mint accepts a raw Pubkey without on-chain validation.

```
// new_distributor.rs
pub mint: Account<'info, Mint>, // no check on freeze_authority
distributor.mint = ctx.accounts.mint.key();

// set_mint.rs
pub fn set_mint<'info>(ctx: Context<SetMint>, mint: Pubkey) -> Result<()> {
    ctx.accounts.distributor.mint = mint; // no validation of freeze authority
    Ok(())
}
```

IMPACT

Recipients ATAs can be frozen post-claim, enabling external censorship/DoS by the freeze authority.

RECOMMENDED MITIGATION

Enforce no freeze authority at initialization and require validation when updating the mint. Accept the full Mint account in set_mint and verify freeze_authority.is_none() before assignment. Document that distributions must use mints without a freeze authority; if retention is necessary, make the distributor PDA the freeze authority and clearly disclose the risks.

```
// new_distributor.rs
require!(
    ctx.accounts.mint.freeze_authority.is_none(),
    ErrorCode::MintHasFreezeAuthority
);

// set_mint accounts and handler
#[derive(Accounts)]
pub struct SetMint<'info> {
    #[account(mut, seeds = [MerkleDistributor::SEED, base.key().to_bytes().as_ref()], bump)]
    pub distributor: Account<'info, MerkleDistributor>,
    pub base: Signer<'info>,
    pub mint: Account<'info, Mint>, // validated mint account
    pub system_program: Program<'info, System>,
}

pub fn set_mint(ctx: Context<SetMint>) -> Result<> {
    require!(
        ctx.accounts.mint.freeze_authority.is_none(),
        ErrorCode::MintHasFreezeAuthority
    );
    ctx.accounts.distributor.mint = ctx.accounts.mint.key();
    Ok(())
}
```

M-03 Missing Chain Identifier Enables Cross-Chain Signature Replay Attacks

Severity **Medium** Status ● Resolved

DESCRIPTION

The claim signature lacks a mandatory domain separator (chain identifier). The signed message includes the distributor PDA but omits any chain-unique identifier (e.g., program ID or cluster/chain ID). Since the distributor PDA is deterministically derived from `[MerkleDistributor::SEED, base.key()]`, the same PDA can exist across deployments on different SVM chains when using the same base key. A signature valid on one chain (e.g., devnet) can be replayed on another (e.g., mainnet) if roots and signer are reused. The extra parameter is optional and unconstrained, making it unreliable as a domain separator without explicit on-chain enforcement.

```
// claim.rs excerpt
let msg = [
    handler.as_ref(),
    claimer.key().to_bytes().as_ref(),
    amount.to_le_bytes().as_ref(),
    released.to_le_bytes().as_ref(),
    root.as_ref(),
    ctx.accounts.distributor.key().to_bytes().as_ref(),
    extra.as_ref(),
].concat();
```

IMPACT

Enables cross-chain signature replay, allowing double-claims across deployments before operators notice, potentially draining vaults and undermining the distributions economic model.

RECOMMENDED MITIGATION

Add explicit domain separation by binding signatures to a specific deployment. Include the program ID and a chain/cluster identifier in the signed message, or enforce a structured extra that contains and validates a chain ID committed on-chain (e.g., stored in distributor state). Prefer a non-optional, validated field for reliability.

```
// Example: include program ID and chain identifier in the signed message
let chain_id: u64 = /* retrieve from distributor state or config constant */;
let msg = [
  handler.as_ref(),
  claimer.key().to_bytes().as_ref(),
  amount.to_le_bytes().as_ref(),
  released.to_le_bytes().as_ref(),
  root.as_ref(),
  ctx.accounts.distributor.key().to_bytes().as_ref(),
  &crate::ID.to_bytes(),           // bind to this program deployment
  &chain_id.to_le_bytes(),         // bind to this chain/cluster
  extra.as_ref(),
].concat();

// Alternative: enforce structured `extra` carrying chain_id (first 8 bytes)
require!(extra.len() >= 8, ErrorCode::InvalidExtra);
let provided_chain_id = u64::from_le_bytes(extra[0..8].try_into().unwrap());
let expected_chain_id: u64 = /* retrieve from state or constant */;
require!(provided_chain_id == expected_chain_id, ErrorCode::InvalidChainId);
```

M-04 Mutable Mint Allows Token Swap After Root Publication

Severity **Medium** Status ● Resolved

DESCRIPTION

distributor.mint can be changed after a Merkle root is published, but claim derives ATAs using the current distributor.mint. This allows operators to swap the asset mid-drop so that valid proofs pay out a different mint than the one intended when the root was created. The Merkle root does not bind the mint, enabling accidental or malicious asset substitution.

IMPACT

Medium-High. Users may receive an unintended or worthless asset while proofs still verify, undermining trust. Conversely, valuable assets in the vault can be drained in exchange for junk assets if the mint is swapped.

RECOMMENDED MITIGATION

Make the mint immutable after initialization, or bind the expected mint to each published root and enforce equality during claim. Additionally, block set_mint once a root is active or claims have begun unless the distributor is paused and no dynamic roots are active. Add tests that attempt mint swaps mid-drop and ensure they are rejected.

L-01 No Fee Caps Allow Unbounded Fee Extraction

Severity **Low** Status ● Resolved

DESCRIPTION

The set_fixed_fee and set_fee_per_tier functions do not enforce maximum fee limits, allowing the base controller to set arbitrarily high fees that could exceed claim amounts.

```
// set_fixed_fee.rs - no cap on fee parameter
pub fn set_fixed_fee(ctx: Context<SetFixedFee>, configurator: [u8; 32], fee: u64)

// set_fee_per_tier.rs - no cap on rate, max_fee, or min_fee
pub fn set_fee_per_tier(
    ctx: Context<SetFeePerTier>,
    configurator: [u8; 32],
    rate: u64,
    max_fee: u64,
    min_fee: u64,
)
```

This allows fees that exceed claim amounts, preventing users from claiming their allocations.

RECOMMENDED MITIGATION

Add fee validation constraints:

```
const MAX_FEE_LAMPORTS: u64 = 1_000_000_000; // 1 SOL max
const MAX_FEE_RATE: u64 = 100_000_000; // 10% max (out of 1B basis)

require!(fee <= MAX_FEE_LAMPORTS, ErrorCode::FeeTooHigh);
require!(rate <= MAX_FEE_RATE, ErrorCode::RateTooHigh);
require!(max_fee <= MAX_FEE_LAMPORTS, ErrorCode::FeeTooHigh);
require!(min_fee <= max_fee, ErrorCode::InvalidFeeRange);
```

L-02 Misleading comment

Severity Low Status Resolved

DESCRIPTION

The docs/comments suggest the vault stores tokens, but in practice tokens live in the vaults Associated Token Account (ATA), and claims transfer from that ATA. The ATAs authority is distributor.vault, while the CPI uses the distributor PDA as the transfer authority, implying a delegate-based flow (owner is vault, delegate is distributor PDA). This mismatch between comments and actual design can confuse integrators.

```
/// The vault to store the tokens in.
pub vault: SystemAccount<'info>,
```

IMPACT

Misleading comment.

RECOMMENDED MITIGATION

In new_distributor.rs, change `The vault to store the tokens in` to `Authority for the distributors token account (tokens reside in the vaults ATA)`.

L-03 set_mint Lacks Mint Account Validation and Legacy Token Enforcement

Severity Low Status Resolved

DESCRIPTION

The program uses legacy SPL Token (v1) throughout (e.g., Token, TokenAccount, CPIs via `anchor_spl::token`) and validates a legacy Mint at initialization, but `set_mint` function accepts a raw Pubkey without owner/layout checks. This allows setting a Token-2022 mint or even a non-mint, which will later mismatch legacy Token CPIs and ATAs during claim, causing failures or undefined behavior.

IMPACT

Low to core protocol logic but operationally disruptive: misconfiguration can DoS claimants (claims fail), strand distributions, and force redeployments/rollbacks. This is primarily a configuration risk rather than direct fund loss.

RECOMMENDED MITIGATION

Change `set_mint` to accept and validate a full Mint account alongside the legacy Token program, enforcing that the mints owner equals the provided legacy Token program and thereby ensuring a valid legacy mint. Keep initialization strict by continuing to accept `Account<'info, Mint>` and applying the same owner constraint there. This binds all ATAs and CPIs to Token v1, prevents Token-2022/non-mint keys from being set, and aligns on-chain checks with Anchors owner/layout guarantees.

L-04 Dynamic Recipient Toggle Bricks Pending Claims

Severity Low Status Resolved

DESCRIPTION

Flipping `distributor.dynamic_recipient` from true to false after publishing a dynamic-root breaks all pending dynamic claims. The handler derivation during claim is keyed off the mutable `dynamic_recipient` flag rather than data committed to the Merkle root, so toggling the flag forces recomputation as `keccak(claimer_pubkey)` and invalidates proofs that were valid at publication.

SETTER

```
pub fn set_is_dynamic_recipient(
    ctx: Context<SetIsDynamicRecipient>,
    is_dynamic_recipient: bool,
) -> Result<()> {
    ctx.accounts.distributor.dynamic_recipient = is_dynamic_recipient;
    emit!(IsDynamicRecipientUpdatedEvent { /* ... */ });
    Ok(())
}
```

IMPACT

A single toggle bricks all dynamic claims, halting distributions and trapping user funds until a fresh root (or distributor) is deployed.

RECOMMENDED MITIGATION

Bind the handler mode to the published root instead of global mutable state. Store `dynamic_recipient` alongside each root on creation and require claim to use the flag committed in the root metadata. Reject toggling while a dynamic-configured root remains active, or enforce a phased rollout (e.g., clearing outstanding dynamic roots before switching to static). Add integration tests covering dynamic-to-static transitions to prevent regressions.

```
pub struct RootConfig {
    pub merkle_root: [u8; 32],
    pub dynamic_recipient: bool,
}
```

L-05 Missing event emission for a critical state update function

Severity Low Status Resolved

DESCRIPTION

The `set_claim_root` instruction updates configuration (root and configurator) without emitting an event. This reduces transparency and makes it hard for indexers, explorers, and monitoring tools to track claim root rotations and who changed them.

```
pub fn set_claim_root<'info>(<
    ctx: Context<SetClaimRoot>,
    root: [u8; 32],
    configurator: [u8; 32],
) -> Result<()> {
    let claim_root = &mut ctx.accounts.claim_root;
    claim_root.distributor = ctx.accounts.distributor.key();
    claim_root.root = root;
    claim_root.configurator = configurator;
    Ok(())
}
```

IMPACT

off-chain systems cant reliably detect claim root/configurator updates, weakening analytics, alerts, and audits.

RECOMMENDED MITIGATION

Add a event like this

```
#[event]
pub struct ClaimRootUpdatedEvent {
    pub distributor: Pubkey,
    pub root: [u8; 32],
    pub configurator: [u8; 32],
    // Optional but recommended for auditability:
    // pub authority: Pubkey,
}
```

I-01 Unused Imports

Severity Informational Status Resolved

DESCRIPTION

Several files contain unused imports that should be removed to improve code cleanliness and maintainability.

```
--> programs/merkle-distributor/src/instructions/new_distributor.rs:3:26
|
3 |     token::{Mint, Token, TokenAccount},
|               ^^^^^^^^^^^^^^^^^
4 |     token_interface::spl_token_metadata_interface::instruction::emit,
|     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
|
= note: `#[warn(unused_imports)]` on by default
```

IMPACT

Informational. No direct security impact, but unused imports add noise to reviews, increase warning volume, and may obscure dead code or lead to confusion during refactors.

RECOMMENDED MITIGATION

Run cargo clippy and remove unused imports across the codebase. Consider enabling CI to treat clippy warnings as errors for unused imports to prevent regressions.

I-02 Missing State Change Guards

Severity **Informational** Status ● Resolved

DESCRIPTION

Setter functions (e.g., `set_mint`, `set_vault`, `set_signer`) do not verify that the provided value differs from the current on-chain state, allowing no-op updates that consume compute and fees while producing redundant events.

IMPACT

Informational/Low. No security impact, but wastes compute/fees, increases write contention, and can clutter event logs with redundant updates.

RECOMMENDED MITIGATION

Add guards to reject or early-return on no-op updates. For example:

```
pub fn set_mint(ctx: Context<SetMint>, mint: Pubkey) -> Result<()> {
    require!(ctx.accounts.distributor.mint != mint, ErrorCode::NoStateChange);
    ctx.accounts.distributor.mint = mint;
    // ... emit event
    Ok(())
}
```

Apply analogous checks in `set_vault` and `set_signer` against their current values.

I-03 Vault Parameter Should Use SystemAccount Type

Severity **Informational** Status ● Resolved

DESCRIPTION

In `set_vault`, the vault parameter is provided as a raw `Pubkey` without on-chain type/owner validation. This allows setting an uninitialized or non-system account as the vault, which is a configuration hazard and can result in zero-initialized or incorrect destinations being recorded.

IMPACT

Primarily an operability/configuration risk rather than direct fund loss. An invalid or zero-initialized vault can cause later transfers, accounting, or withdrawals to fail or behave unexpectedly (DoS-like until corrected).

RECOMMENDED MITIGATION

Require a `SystemAccount<'info>` for the new vault and assign the distributors vault from that accounts key. This ensures the account exists and is owned by the system program. Optionally add rent-exemption checks if the vault must maintain minimum balance guarantees.

```
#[derive(Accounts)]
pub struct SetVault<'info> {
    #[account(mut, seeds = [MerkleDistributor::SEED, base.key().to_bytes().as_ref()], bump)]
    pub distributor: Account<'info, MerkleDistributor>,
    pub base: Signer<'info>,
    pub new_vault: SystemAccount<'info>,           // Ensure it's a valid system account
    pub system_program: Program<'info, System>,
}

pub fn set_vault(ctx: Context<SetVault>) -> Result<()> {
    ctx.accounts.distributor.vault = ctx.accounts.new_vault.key();
    Ok(())
}
```

I-04 Misspelled Helper Function receover_signer Hurts Clarity

Severity Informational

Status ● Resolved

DESCRIPTION

The helper function that recovers the signer from a secp256k1 signature is misspelled as `receover_signer` instead of `recover_signer`. While functionally harmless, this reduces readability and risks confusion or propagation of the typo in future refactors and integrations.

IMPACT

No runtime/security impact, but degrades code clarity and increases the chance of human error during maintenance or external referencing.

RECOMMENDED MITIGATION

Rename the function and all call sites from `receover_signer` to `recover_signer`. Keep the implementation unchanged.

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