Ika

Security Assessment

June 25th, 2025 — Prepared by OtterSec

Tuyết Dương tuyet@osec.io

Himanshu Sheoran deuterium@osec.io

Robert Chen r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

dWallet Labs engaged OtterSec to assess the `ika` and the cryptography (`inkrypto`) programs. This assessment was conducted between March 1st and June 17th, 2025. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 14 findings throughout this audit engagement.

In particular, we identified several critical vulnerabilities. Specifically, asynchronous message handling may create inconsistent MPC states, where some parties process messages from a now-flagged malicious actor while others ignore them, resulting in state divergence (OS-Ika-ADV-00). It is also possible for anyone to consume available presigns without validating dWallet ownership, resulting in a race condition enabling attackers to drain presigns and create a denial-of-service scenario for legitimate users (OS-Ika-ADV-01).

We also identified multiple high-risk issues, including the calculation of share size limits utilizing the current access structure instead of the upcoming one, the potential to produce undersized shares when the upcoming quorum is larger (OS-Ika-ADV-02), the possibility of a validator sending messages with ever-increasing round numbers, bloating serialized messages and preventing the quorum (OS-Ika-ADV-04), and the failure to verify imported encrypted shares, allowing malicious validators to inject an invalid encrypted centralized secret share and proof (OS-Ika-ADV-03).

Additionally, the majority vote functionality incorrectly determines consensus based on the number of tangible parties rather than their cumulative weights, enabling low-weight colluding parties to override high-weight honest participants (OS-Ika-ADV-06), and the multiplication logic mistakenly checks the bit length of the input instead of the result, allowing out-of-bound values to bypass validation (OS-Ika-ADV-09).

Furthermore, a rounding error while withdrawing stake may result in the withdrawal of zero shares for a small but non-zero stake, allowing attackers to manipulate the staking pool's share-to-asset ratio and extract unearned value (OS-Ika-ADV-07), and the reward distribution logic only updates current validators, skipping those in the pending active set and resulting in them missing reward updates and state transitions after epoch advancement (OS-Ika-ADV-11).

All the vulnerabilities outlined above were addressed by the dWallet team. Additionally, the team independently identified and reported several other issues, including critical vulnerabilities, which we have verified were successfully resolved.

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/dwallet-labs/ika and https://github.com/dwallet-labs/inkrypto.

This audit was performed against https://github.com/dwallet-labs/ika (commit 9a53629), and https://github.com/dwallet-labs/inkrypto (commit 29d2bcf).

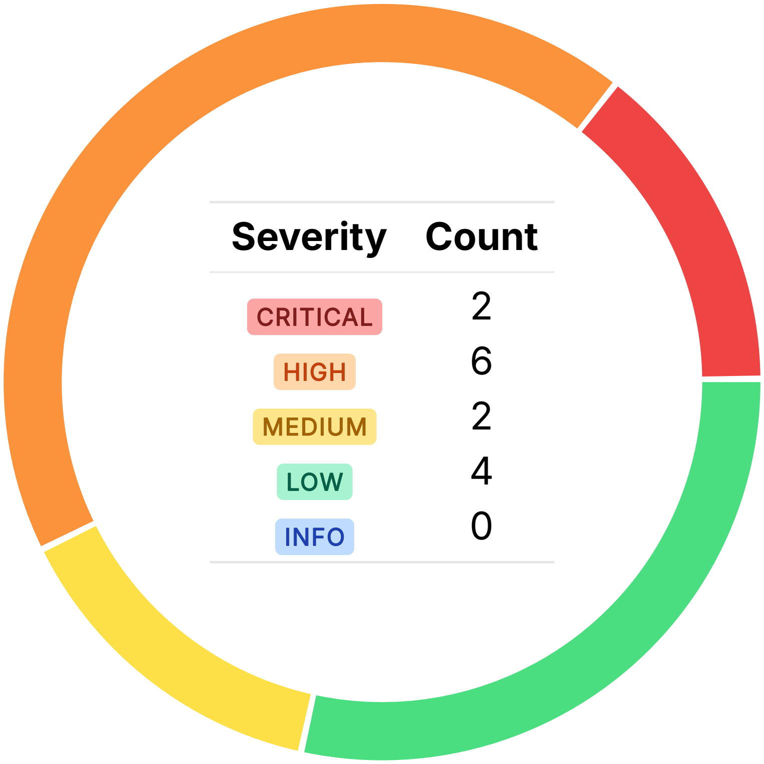
A brief description of the programs is as follows:

Name	Description
ika	A decentralized platform that enables secure, user-consented, multi-chain transaction signing using a novel 2PC-MPC protocol. It eliminates the need for trusted bridges by allowing smart contracts (e.g., on Sui) to control decentralized wallets that can natively sign on any blockchain.
inkrypto	Implementation of a novel 2PC-MPC protocol that enables secure, non-collusive ECDSA signing between a user and a decentralized network.

03 — Findings

Overall, we reported 14 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	2
HIGH	6
MEDIUM	2
LOW	4
INFO	0

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-Ika-ADV-00	CRITICAL	RESOLVED ✓	Asynchronous message handling may create inconsistent MPC states, where some parties process messages from a now-flagged malicious actor while others ignore them, resulting in state divergence.
OS-Ika-ADV-01	CRITICAL	RESOLVED ✓	It is possible for anyone to consume available presigns without validating dWallet ownership, resulting in a race condition enabling attackers to drain presigns and create a denial-of-service scenario for legitimate users.
OS-Ika-ADV-02	HIGH	RESOLVED ✓	The current logic calculates share size bounds utilizing the current access structure instead of the upcoming one, potentially producing undersized shares when the upcoming quorum is larger.
OS-Ika-ADV-03	HIGH	RESOLVED ✓	`advance_specific_party` skips verifying imported encrypted shares, allowing malicious validators to inject an invalid encrypted centralized secret share and proof.
OS-Ika-ADV-04	HIGH	RESOLVED ✓	A malicious validator may send messages with ever-increasing `round_numbers` , bloating `serialized_messages` and preventing quorum.

OS-Ika-ADV-05	HIGH	RESOLVED ✓	The `2pc-mpc` protocol relies on message count to determine progression. However, the `store_message` function permits messages from round 0, even though no messages should exist at that stage.
OS-Ika-ADV-06	HIGH	RESOLVED ✓	`majority_vote` incorrectly determines consensus based on the number of tangible parties rather than their cumulative weights, enabling low-weight colluding parties to override high-weight honest participants.
OS-Ika-ADV-07	HIGH	RESOLVED ✓	A rounding error while withdrawing stake may result in the withdrawal of zero shares for a small but non-zero stake, allowing attackers to manipulate the staking pool's share-to-asset ratio and extract unearned value.
OS-Ika-ADV-08	MEDIUM	RESOLVED ✓	The refactored logic, after the pull request to re-utilize Sui's `StakeAggregator`, fails to add malicious actors to the malicious actor list after quorum is reached.
OS-Ika-ADV-09	MEDIUM	RESOLVED ✓	`mul` mistakenly checks the bit length of the input instead of the result, allowing out-of-bound values to bypass validation.
OS-Ika-ADV-10	LOW	RESOLVED ✓	The verifier does not ensure that all votes agree on the `SessionInfo`, allowing malicious parties to submit outputs with mismatched session data.
OS-Ika-ADV-11	LOW	RESOLVED ✓	The reward distribution logic only updates current validators, skipping those in the `pending_active_set` , resulting in them missing reward updates and state transitions after epoch advancement.

OS-Ika-ADV-12	LOW	RESOLVED ✓	`handle_threshold_not_reached_report` accepts votes from malicious validators, allowing them to falsely influence quorum.
OS-Ika-ADV-13	LOW	RESOLVED ✓	`active_sessions_counter` is never decremented after session completion, resulting in the manager blocking new sessions indefinitely once the limit is reached, adding them to the pending queue.

Asynchronous Message Handling Inconsistency CRITICAL OS-Ika-ADV-00

Description

There is a potential race condition between synchronous and asynchronous handling of MPC (Multi-Party Computation) messages. When `process_dwallet_mpc_output` in `authority_per_epoch_store` (shown below) verifies an output and flags an authority as malicious, it does so in a synchronous, consensus-driven context. If verification fails, it flags the authority as malicious. However, `handle_dwallet_db_message` processes incoming messages asynchronously. As a result, some fast-moving parties may have already processed a malicious authority's messages before the actor is flagged, while others, receiving the same messages later, will ignore them in `mpc_manager::handle_message`.

>_ *dwallet-network/crates/ika-core/src/authority/authority_per_epoch_store.rs* RUST

```rust
async fn process_dwallet_mpc_output(
    &self,
    origin_authority: AuthorityName,
    session_info: SessionInfo,
    output: Vec<u8>,
) -> IkaResult<ConsensusCertificateResult> {
    [...]
    let mut manager = self.get_dwallet_mpc_manager().await;
    manager.flag_authorities_as_malicious(&output_verification_result.malicious_actors);
    [...]
}
```

Thus, once a party is flagged malicious in the consensus-driven `process_dwallet_mpc_output`, subsequent rejections in `handle_message` (shown below) may result in state divergence if some parties have already acted on those now-invalid messages. This discrepancy creates inconsistent MPC session states across validators, corrupting MPC session outcomes and affecting protocol correctness.

>_ *dwallet-network/crates/ika-core/src/dwallet_mpc/mpc_manager.rs* RUST

```rust
pub(crate) fn handle_message(&mut self, message: DWalletMPCMessage) -> DwalletMPCResult<()> {
    if self
        .malicious_handler
        .get_malicious_actors_names()
        .contains(&message.authority)
    {
        // Ignore a malicious actor's messages.
        return Ok(());
    }
    [...]
}
```

Remediation

Ensure consistency in message-handling logic.

Patch

Resolved in PR#749.

No Access Control for Future Sign CRITICAL OS-Ika-ADV-01

Description

The issue arises from improper access control in `request_ecdsa_future_sign` within `dwallet_2pc_mpc_secp256k1_inner`. It allows any actor to consume a presign without proving dWallet ownership. A presign is removed from the available presign table and placed inside the `ECDSAPartialUserSignature` object without a `PresignCap` check.

>_ *dwallet-network/crates/ika-move-packages/packages/ika_system/dwallet_2pc_mpc_secp256k1_inner.move* RUST

```rust
public(package) fun request_ecdsa_future_sign(
    self: &mut DWalletCoordinatorInner,
    dwallet_id: ID,
    message: vector<u8>,
    presign_id: ID,
    hash_scheme: u8,
    message_centralized_signature: vector<u8>,
    payment_ika: &mut Coin<IKA>,
    payment_sui: &mut Coin<SUI>,
    ctx: &mut TxContext
): UnverifiedECDSAPartialUserSignatureCap {
    let (dwallet, _) = self.get_active_dwallet_and_public_output_mut(dwallet_id);
    let dwallet_network_decryption_key_id = dwallet.dwallet_network_decryption_key_id;

    // TODO: Change error
    assert!(dwallet.ecdsa_presigns.contains(presign_id), EPresignNotExist);

    let presign = dwallet.ecdsa_presigns.remove(presign_id);
}
```

This introduces a race condition between legitimate dWallet users and malicious actors for the same presign. A malicious user may repeatedly call the function and remove all presigns from the `ecdsa_presigns` store, even without legitimate signing intent.

Remediation

Require the caller to provide a `PresignCap` proving ownership of the presign.

Patch

Resolved in PR#786.

Incorrect Share Bits Calculation HIGH OS-Ika-ADV-02

Description

In `part::advance` within `class-groups::reconfiguration`, the bit-size upper bound for secret shares (`decryption_key_share_bits` / `randomizer_share_bits`) is incorrectly computed utilizing the current access structure `(n, t)` instead of the upcoming one `(n', t')`. This value is utilized as `randomizer_share_bits` for the PVSS targeting the upcoming quorum (within `randomizer_contribution_to_upcoming_pvss_party`).

>_ *cryptography-private/class-groups/src/reconfiguration/party.rs* RUST

```rust
fn advance([...]) -> Result<AsynchronousRoundResult<Self::Message, Self::PrivateOutput,
    ↪  Self::PublicOutput>>
{
    [...]
    let decryption_key_bits = setup_parameters.decryption_key_bits();
    let decryption_key_share_bits = secret_key_share_size_upper_bound(
        u32::from(current_access_structure.number_of_virtual_parties()),
        u32::from(current_access_structure.threshold),
        decryption_key_bits,
    );
    [...]
}
```

If `n'` or `t'` is significantly larger, and `n` and `t` are small, this will result in a smaller share upper bound for the upcoming access structure.

Remediation

The `decryption_key_share_bits` / `randomizer_share_bits` for `randomizer_contribution_to_upcoming_pvss_party` should be derived utilizing the upcoming access parameters rather than the current values.

Patch

The `randomizer_share_bits` is now based on the `upcoming_access_structure`'s size.

>_ *cryptography-private/class-groups/src/reconfiguration/party.rs* RUST

```rust
    let randomizer_share_bits = secret_key_share_size_upper_bound(Add commentMore actions
            u32::from(
                public_input
                    .upcoming_access_structure
                    .number_of_virtual_parties(),
            ),
            u32::from(public_input.upcoming_access_structure.threshold),
            decryption_key_bits,
        );
}
```

Missing Imported Key Verification HIGH OS-Ika-ADV-03

Description

`advance_specific_party` in `mpc_session` is responsible for progressing MPC sessions. For certain session types such as `DKGSecond`, it calls `verify_encrypted_share` to validate that the centralized encrypted share is indeed a correct encryption of the user's decentralized DKG (Distributed Key Generation) output.

>_ *dwallet-network/crates/ika-core/src/dwallet_mpc/encrypt_user_share.rs* RUST

```rust
/// Verifies that the given encrypted secret key share matches the encryption of the dWallet's
/// secret share, validates the signature on the dWallet's public share,
/// and ensures the signing public key matches the address that initiated this transaction.
pub(crate) fn verify_encrypted_share(
    verification_data: &StartEncryptedShareVerificationEvent,
) -> DwalletMPCResult<()> {
    verify_centralized_secret_key_share_proof(
        &verification_data.encrypted_centralized_secret_share_and_proof,
        &verification_data.decentralized_public_output,
        &verification_data.encryption_key,
    )
    .map_err(|_| DwalletMPCError::EncryptedUserShareVerificationFailed)
}
```

However, in the case of `DWalletImportedKeyVerificationRequest`, it fails to invoke `verify_encrypted_share`, which is critical for verifying that the `encrypted_centralized_secret_share_and_proof` correctly corresponds to the public DKG output. Without this check, a malicious validator may generate an incorrect `encrypted_centralized_secret_share_and_proof` for users, compromising user integrity.

Remediation

Ensure `advance_specific_party` calls `verify_encrypted_share` for `DWalletImportedKeyVerificationRequest` to verify the `encrypted_centralized_secret_share_and_proof` and public output.

Patch

Resolved in PR#1083.

Current Round Number Manipulation HIGH OS-Ika-ADV-04

Description

`store_message` in `mpc_session` allows a message to be appended if its `round_number` equals the current number of rounds, which is based on the `serialized_messages.len`. Consequently, a malicious validator may exploit this by repeatedly sending messages with incrementally higher `round_numbers`, resulting in unbounded growth of the `serialized_messages` vector. This bloats memory and prevents the MPC session from obtaining the correct set of `serialized_messages`, effectively stalling the `advance` calls, resulting in a denial-of-service scenario.

>_ dwallet-network/crates/ika-core/src/dwallet_mpc/mpc_session.rs RUST

```rust
pub(crate) fn store_message(&mut self, message: &DWalletMPCMessage) -> DwalletMPCResult<()> {
    [...]
    let current_round = self.serialized_messages.len();
    match self.serialized_messages.get_mut(message.round_number) {
        Some(party_to_msg) => {[...]}
        // If next round.
        None if message.round_number == current_round => {
            let mut map = HashMap::new();
            map.insert(source_party_id, message.message.clone());
            self.serialized_messages.push(map);
        }
        None => {
            // Unexpected round number; rounds should grow sequentially.
            return Err(DwalletMPCError::MaliciousParties(vec![source_party_id]));
        }
    }
    Ok(())
}
```

Remediation

Introduce a hard upper bound for the round number.

Patch

Resolved in PR#762.

Improper Advancement Due to Round 0 Messages HIGH OS-Ika-ADV-05

Description

The `2pc-mpc` protocol utilizes the length of messages to determine the `advance`, and in `mpc_session::store_message`, incoming MPC messages are stored based on their `round_number`, with round 0 initialized by default. However, in the `2pc-mpc` protocol, no messages are expected in round 0 as it is a setup phase, not an interactive one. Accepting messages for round 0 may result in the session incorrectly assuming a quorum has been reached, triggering an unintended call to `advance`.

Remediation

Ensure that round 0 messages are explicitly rejected.

Patch

Resolved in PR#864.

Consensus Manipulation via Improper Vote Calculation HIGH OS-Ika-ADV-06

Description

The vulnerability concerns consensus manipulation due to the improper implementation of majority vote calculation logic. `majority_vote` in the MPC module determines the majority decision based on the number of parties, ignoring their assigned weights. This contradicts the principles of weighted threshold MPC protocols, where each party has a weight and the protocol's correctness depends on the majority of total weight, not just party count. As a result, a group of colluding low-weight parties may outvote fewer high-weight honest parties, overriding their decisions and enabling manipulation of consensus outcomes.

>_ *dwallet-labs/cryptography-private/mpc/src/lib.rs* RUST

```rust
  fn majority_vote(self) -> Result<(Vec<PartyID>, T)> {
    [...]
    self.into_iter().for_each(|(party_id, candidate)| {
        let mut parties_voting_candidate = candidates
            .get(&candidate)
            .cloned()
            .unwrap_or(HashSet::new());
        parties_voting_candidate.insert(party_id);
        candidates.insert(candidate, parties_voting_candidate);
    });
    let (majority_vote, majority_voters) = candidates
        .clone()
        .into_iter()
        .max_by(
            |(_, first_parties_verifying_candidate),
             (_, second_parties_verifying_candidate)| {
                first_parties_verifying_candidate
                    .len()
                    .cmp(&second_parties_verifying_candidate.len())
            },
        )
        .ok_or(Error::InvalidParameters)?;
    [...]
}
```

We note that that this would not have effected the network as reconfigurations now are using fixed-stake (1 per validator).

Remediation

Modify the assertion in `majority_vote` such that it considers the majority of virtual parties.

Patch

The code now utilizes the following `weighted_majority_vote` function that considers weights.

```rust
  fn weighted_majority_vote(
        self,
        access_structure: &WeightedThresholdAccessStructure,
    ) -> Result<(Vec<PartyID>, T)> {
        let tangible_parties_that_voted: HashSet<PartyID> = self.keys().copied().collect();

        let votes_by_virtual_parties: HashMap<_, _> = self
            .into_iter()
            .map(|(tangible_party_id, vote)| {
                let virtual_subset =
                    access_structure.virtual_subset(HashSet::from([tangible_party_id]))?;

                Ok(virtual_subset
                    .into_iter()
                    .map(|virtual_party_id| (virtual_party_id, vote.clone()))
                    .collect::<Vec<_>>())
            })
            .collect::<Result<Vec<_>>>()?
            .into_iter()
            .flatten()
            .collect();

        let (malicious_virtual_voters, majority_vote) =
            ↪   votes_by_virtual_parties.majority_vote()?;

        let malicious_tangible_voters = malicious_virtual_voters
            .into_iter()
            .flat_map(|virtual_party_id|
                ↪   access_structure.to_tangible_party_id(virtual_party_id))
            .collect();

        // Check that the parties that voted honestly form an authorized subset.
        let honest_tangible_parties = tangible_parties_that_voted
            .difference(&malicious_tangible_voters)
            .copied()
            .collect();

        access_structure.is_authorized_subset(&honest_tangible_parties)?;

        Ok((
            malicious_tangible_voters.deduplicate_and_sort(),
            majority_vote,
        ))
    }
}
```

Manipulation of Staking Pool Ratio via Rounding HIGH OS-Ika-ADV-07

Description

In `request_withdraw_stake`, due to rounding in `get_token_amount`, the `pool_token_withdraw_amount` may be zero while `principal_withdraw > 0`. This rounding issue may result in `convert_to_share_amount` returning zero shares for a small non-zero principal when the staked amount is too small relative to the pool size, and integer division rounds the result to zero. Consequently, this allows a malicious user to manipulate the staking pool's share-to-asset ratio by withdrawing a small principal without reducing their share count, violating the share-to-asset ratio invariant.

Remediation

Ensure to always round up in conversions to favor the protocol over the users.

Patch

The issue was resolved with a new staking code PR#971.

Lack of Addition of Malicious Actors MEDIUM OS-Ika-ADV-08

Description

The vulnerability arises with the new pull request to re-utilize Sui's `StakeAggregator`. With this change, while `report_malicious_actor` detects when a quorum of validators reports a malicious actor, it no longer adds that actor to the internal `malicious_actors` list, allowing malicious validators to remain active even after being flagged.

>_ *dwallet-network/crates/ika-core/src/dwallet_mpc/malicious_handler.rs* RUST

```rust
    pub(crate) fn report_malicious_actor(
        &mut self,
        report: MaliciousReport,
        authority: AuthorityName,
    ) -> DwalletMPCResult<ReportStatus> {
        let report_votes = self
            .reports
            .entry(report)
            .or_insert(StakeAggregator::new(self.committee.clone()));
        if report_votes.has_quorum() {
            return Ok(ReportStatus::OverQuorum);
        }
        if report_votes
            .insert_generic(authority, ())
            .is_quorum_reached()
        {
            return Ok(ReportStatus::QuorumReached);
        }
        Ok(ReportStatus::WaitingForQuorum)
    }
}
```

Remediation

Ensure malicious actors are appropriately added to the malicious actor list when quorum is reached.

Patch

Resolved in PR#778.

Erroneous Bit Length Check MEDIUM OS-Ika-ADV-09

Description

`GroupElement::mul` in `bounded_natural_numbers_group` incorrectly checks the bit length of the input (`self.value`) instead of the multiplication result (`value`). This allows results that exceed the intended `upper_bound_bits` to pass unchecked, potentially resulting in overflows.

>_ *cryptography-private/group/src/bounded_natural_numbers_group.rs* RUST

```rust
impl<'r, const LIMBS: usize> Mul<Self> for &'r GroupElement<LIMBS> {
    type Output = GroupElement<LIMBS>;

    fn mul(self, rhs: Self) -> Self::Output {
        let value = self.value.mul(rhs.value);
        assert!(self.value.bits() <= self.upper_bound_bits);

        GroupElement::<LIMBS> {
            value,
            sample_bits: self.sample_bits,
            upper_bound_bits: self.upper_bound_bits,
        }
    }
}
```

Remediation

`Mul` should utilize `value.bits` instead of `self.value.bits` to check against the `upper_bound_bits`.

Patch

The functions now utilize `value.bits` instead of `self.value.bits`.

>_ *cryptography-private/group/src/bounded_natural_numbers_group.rs* RUST

```rust
impl<'r, const LIMBS: usize> Mul<Self> for &'r GroupElement<LIMBS> {
    type Output = GroupElement<LIMBS>;

    fn mul(self, rhs: Self) -> Self::Output {
        let value = self.value.mul(rhs.value);
        assert!(value.bits() <= self.upper_bound_bits);

        GroupElement::<LIMBS> {
            value,
            sample_bits: self.sample_bits,
            upper_bound_bits: self.upper_bound_bits,
        }
    }
}

impl<'r, const LIMBS: usize> Mul<&'r Self> for &'r GroupElement<LIMBS> {
    type Output = GroupElement<LIMBS>;

    fn mul(self, rhs: &'r Self) -> Self::Output {
        let value = self.value.mul(rhs.value);
        assert!(value.bits() <= self.upper_bound_bits);

        GroupElement::<LIMBS> {
            value,
            sample_bits: self.sample_bits,
            upper_bound_bits: self.upper_bound_bits,
        }
    }
}
```

Missing Session Info Agreement Check in Output Verifier LOW OS-Ika-ADV-10

Description

`try_verify_output` in `mpc_outputs_verifier` fails to validate that all votes for a given `MPCPublicOutput` also agree on the exact same `SessionInfo`. This omission allows malicious parties to submit outputs with valid signatures but tampered `SessionInfo` containing incorrect `mpc_round` values. Note that the only effect here is resource exhaustion.

>_ *dwallet-network/crates/ika-core/src/dwallet_mpc/mpc_outputs_verifier.rs* RUST

```rust
pub async fn try_verify_output([...]) -> DwalletMPCResult<OutputVerificationResult> {
    [...]
    if let Some((agreed_output, _)) = agreed_output {
        let voted_for_other_outputs = session_output_data
            .session_output_to_voting_authorities
            .iter()
            .filter(|(output, _)| *output != agreed_output)
            .flat_map(|(_, voters)| voters)
            .cloned()
            .collect();
        session_output_data.current_result = OutputResult::AlreadyCommitted;
        [...]
    }
    [...]
}
```

Remediation

Ensure that `try_verify_output` enforces that validators' agree on the same `SessionInfo` when submitting outputs.

Patch

Resolved in PR#762.

Failure to Update Pending Active Set Post Epoch Advance LOW OS-Ika-ADV-11

Description

In `validator_set::distribute_reward`, after the epoch is advanced via the `advance_epoch` call, only active validators are updated, ignoring those in the `pending_active_set` who are scheduled to join in the new epoch. As a result, these pending validators may miss reward allocations and fail to transition their internal state.

>_ *ika-move-packages/packages/ika_system/sources/system/validator_set.move* RUST

```
/// Distribute rewards to validators and stakers
fun distribute_reward([...]) {
    [...]
    while (i < length) {
        let validator_id = members[i].validator_id();
        let validator = self.get_validator_mut(validator_id);
        let staking_reward_amount = adjusted_staking_reward_amounts[i];
        let validator_rewards = staking_rewards.split(staking_reward_amount);
        validator.advance_epoch(validator_rewards, new_epoch);
        i = i + 1;
    }
}
```

Remediation

Update the `pending_active_set` to account for the new rewards after an epoch is advanced.

Patch

Resolved in PR#993.

Votes Acceptance from Malicious Parties LOW OS-Ika-ADV-12

Description

`mpc_manager::handle_threshold_not_reached_report` lacks a check to exclude malicious authorities from voting. It accepts reports from all validators without verifying whether the sender (`origin_authority`) is flagged as malicious. This allows malicious authorities to influence quorum calculations by submitting dishonest reports, as validators flagged as malicious are still counted toward quorum in `StakeAggregator`, resulting in premature triggering of session retries.

Remediation

Update `handle_threshold_not_reached_report` such that it ignores votes from known malicious actors.

Patch

Resolved in PR#1048.

Failure to Decrement Active Sessions Counter LOW

OS-Ika-ADV-13

Description

`mpc_manager::push_new_mpc_session` increments the `active_sessions_counter` when activating a new session, but never decrements it when sessions complete. As a result, once the counter reaches the configured maximum (`max_active_mpc_sessions`), all subsequent sessions are pushed to the `pending_sessions_queue` and will never be added to `mpc_sessions`. Thus, without decrementing the counter, the system effectively stalls despite having available capacity.

>_ *dwallet-network/crates/ika-core/src/dwallet_mpc/mpc_manager.rs* RUST

```rust
/// Spawns a new MPC session if the number of active sessions is below the limit.
/// Otherwise, add the session to the pending queue.
pub(crate) fn push_new_mpc_session(
    &mut self,
    public_input: MPCPublicInput,
    private_input: MPCPrivateInput,
    session_info: SessionInfo,
) -> DwalletMPCResult<()> {
    [...]
    self.active_sessions_counter += 1;
    info!(
        "Added MPCSession to MPC manager for session_id {:?}",
        session_info.session_id
    );
    Ok(())
}
```

Remediation

Ensure to decrement `active_sessions_counter` when sessions complete.

Patch

Resolved in PR#745.

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL

Vulnerabilities that immediately result in irreparable damages with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.