OpenZeppelin | security

Oxorio SAMM Module Audit

OXORIO

March 7, 2025

Table of Contents

Table of Contents 2

Summary 4

Scope 5

System Overview 6

Usage Workflow 6

Security Model, Roles, and Trust Assumptions 6

Additional Information 7

High Severity 8

H-01 Unconstrained Domain in DKIM Signature 8

H-02 Valid Emails May Be Unprovable 8

H-03 Front-Running Can Lead to Multisig Wallet DoS 9

Medium Severity 9

M-01 Participants Are Partially De-Anonymized 9

Low Severity 10

L-01 Insufficient Setup Validation Can Render SAMM Unusable 10

L-02 Incomplete Transaction Validation 11

L-03 Inconsistent Field Sizes 11

L-04 Floating Compiler Version in Noir Codebase 11

L-05 Allowance System Ignores Operation Type 12

L-06 Potential Use of Revoked Public Key Hashes With Updated DKIM Registry 12

L-07 Use of Non-Audited Base64 Library 12

L-08 No Event Emitted for Allowed Transactions Added During Setup 13

L-09 Duplicate Transaction IDs Can Overwrite Each Other During Setup 13

L-10 Misleading and Incomplete Documentation 14

L-11 The Codebase Contains Magic Numbers 15

Notes & Additional Information 15

N-01 Code Simplifications 15

N-02 Code Inefficiencies 16

N-03 Code Redundancy 18

N-04 Unused Contract 18

N-05 Event Parameter for DKIM Registry Is Not Indexed 18

N-06 Typographical Errors 19

N-07 Naming Suggestions 19

Conclusion 20

Summary

Type	ZK	Total Issues	22 (0 resolved)
Timeline	From 2024-12-09 To 2025-01-03	Critical Severity Issues	0 (0 resolved)
Languages	Solidity, Noir	High Severity Issues	3 (0 resolved)
		Medium Severity Issues	1 (0 resolved)
		Low Severity Issues	11 (0 resolved)
		Notes & Additional Information	7 (0 resolved)

Scope

We audited the oxor-io/samm-circuits repository at commit 240b94a and the oxor-io/samm-contracts repository at commit c2d255d.

In scope were the following files:

```
samm-circuits
└── lib
    ├── src
    │   ├── lib.nr
    │   ├── merkle_tree.nr
    │   └── utils.nr
    └──  builds
        ├── samm_1024
        │   └── src
        │       └── main.nr
        └── samm_2048
            └── src
                └── main.nr
```

and

```
samm-contracts
└── src
    ├── SAMM.sol
    ├── ModuleGuard.sol
    ├──  libraries
    │   ├── Enum.sol
    │   └── PubSignalsConstructor.sol
    └── utils
        └── DKIMRegistry.sol
```

System Overview

The Safe Anonymous Mail Module (SAMM) by Oxorio is a privacy-tech module for the Safe multisig wallet. It enables anonymous transaction initiation and approval through email, ensuring that participant identities remain confidential to external parties throughout the process. It makes use of zero-knowledge (ZK) proofs by employing custom Noir circuits and autogenerated Verifiers derived from the Barretenberg proving library.

Usage Workflow

SAMM integrates with Safe multisig wallets, allowing users to initiate and approve transactions via email. The workflow is as follows:

1. **Transaction Initiation**: A SAMM member sends an email to the relayer's address containing transaction details such as recipient (`to`), data, value, and deadline.
2. **Approval**: Other members approve the transaction by sending emails with the `msgHash` as the subject, each including a DKIM signature in the header.
3. **Relayer Processing**: The relayer stores approval data and, upon reaching the voting threshold, generates a ZK proof.
4. **On-Chain Execution**: The relayer submits the transaction data and proof to the SAMM smart contract, which verifies the proof and executes the transaction if the threshold is met. This process ensures that all participants can manage transactions without revealing their identities to entities outside the relayer infrastructure.

Security Model, Roles, and Trust Assumptions

In the overall system surrounding the SAMM module, we identified the following roles and necessary trust assumptions:

- **EOA Owners**: EOAs with full control over the Safe multisig, including managing SAMM permissions.

- **SAMM Module**: Holds restricted permissions, limited to a specific transaction allowlist. Allowlisted transactions are additionally limited by an allowance value. The SAMM cannot execute administrative actions (e.g., it cannot change the Safe multisig owners).
- **Relayer**: Responsible for processing approval emails, generating ZK proofs, and submitting transactions to the `SAMM` contract. The transaction information of the SAMM members and their full email addresses are revealed to the relayer.
- **DKIM Registry contract**: The DKIM Registry is assumed to be securely administered with accurate information about valid and revoked public key hashes. We assume that the centralization of revocation rights is not abused by the DKIM Registry admin (e.g., to censor approval transactions observed in the mempool).
- **Sending Mail Server**: The mail server(s) of the SAMM member is responsible for the DKIM signatures. We assume it never spoofs emails, but instead securely authenticates each sender before an email with DKIM signature is sent. Furthermore, the mail server is assumed to be configured with a relaxed Header Canonicalization Algorithm as the Noir circuit code relies on lower-case email fields (e.g., `from:`, `to:` and `subject:`, and stripped whitespaces).

Additional Information

For more detailed technical requirements and specifications, refer to the SAMM Technical Requirements.

High Severity

H-01 Unconstrained Domain in DKIM Signature

The circuit ensures that the header's signature matches the specified public key, while the `SAMM` contract validates that the corresponding public key hash is contained in the DKIM registry. However, neither the circuit nor the contract ensures that the particular public key corresponds to the participant's email domain.

This means that a malicious DKIM signer can forge a message from participants who use a different email domain. To complete the forged proof, they would need the participant's `secret` value, which likely requires colluding with the relayer. Nevertheless, this could significantly reduce the multisig threshold.

Consider validating that the DKIM signature matches the email sender's domain.

Update: *The Oxorio team implemented a fix in PR #3. However, this fix has not been evaluated by the OpenZeppelin team.*

H-02 Valid Emails May Be Unprovable

The `check_from_field` function looks for a `<` character and parses the email address differently depending on whether it is found. The `check_to_field` function follows the same pattern. However, the search function introduces an offset at the start of the field and then uses the same offset again in the search pointer. The offset is intended to skip over the name of the field and the subsequent `:` character, but since it is double-counted, the function actually skips the start of the field contents. If this causes the search to skip the `<` character, depending on how it is called, this will either fail because of an index mismatch with the email address location within the buffer or because the field does not match the expected email address. In either case, this means that some valid emails cannot be proven.

Consider correcting the offset to not double-count the field name skip. Consider applying this patch in a local copy of the `constrain_header_field_detect_last_angle_bracket` function until the ZKEmail.nr repository has fixed this issue.

Update: *Acknowledged, not resolved. The Oxorio team stated*:

> We do not want to fork and modify the library; we will wait for an official fix and update the version accordingly.

H-03 Front-Running Can Lead to Multisig Wallet DoS

The `executeTransaction` and `executeTransactionReturnData` functions of the `SAMM` contract currently do not validate the caller. As such, anyone can invoke them with the required number of correct proofs. This allows an external party to observe the relayer's transaction in the mempool and resubmit it with a different gas amount. This amount could be chosen so that when executing the desired operation, the forwarded gas is insufficient to complete the operation but the EIP-150 retained amount is enough to complete the overall transaction, which would increment the `SAMM`'s nonce. This would invalidate the relayer's transaction, effectively discarding the off-chain coordination and computation required to produce the proof.

Consider implementing some combination of the following countermeasures against the scenarios outlined above:

- Integration of the relayer's address as a public signal and subsequent on-chain verification.
- Explicit checks of the call's success condition.
- Integration of the required gas amount as a public signal and subsequent on-chain verification

Update: *The Oxorio team implemented a fix in PR #3. Relayer's address as a public input in the circuit was added. However, this fix has not been evaluated by the OpenZeppelin team.*

Medium Severity

M-01 Participants Are Partially De-Anonymized

For each valid ZK proof, the caller must specify the domain and public key hash. This partially de-anonymizes each participant, since their email domain is revealed. This means that the relevant anonymity set is the other participants that share the same email domain.

To achieve general anonymity, consider keeping the domain private as well. This would likely involve replacing the public key hash circuit input with a commitment or Merkle root of all valid (and unrevoked) public keys.

Update: *Acknolwedged, not resolved. The Oxorio team stated:*

> Since this issue only affects anonymity and requires significant changes to the codebase, we decided not to fix it in this development iteration.

Low Severity

L-01 Insufficient Setup Validation Can Render SAMM Unusable

The `SAMM` contract is deployed using a proxy factory, with its `setup` function being responsible for initializing instances with specific input data. One of these inputs, the `relayer` argument, represents an email address of the relayer. The contract assumes that this email address will not exceed 124 characters, as enforced in the `setRelayer` function. However, this assumption is not explicitly validated during the `setup` function, leaving room for unintended behavior. During the `setup` process, the `relayer` argument is only checked to ensure that it is not empty, and that no validation exists to restrict its maximum length. This becomes problematic in the `getPubSignals` function, which constructs a fixed-length array of size 172 for public signals. The email address begins writing at index 1, but if its length exceeds the array's capacity, an out-of-bounds write attempt triggers a contract panic error (with code `0x32`).

Another input is the `membersRoot` argument, which can currently be set to any value greater than zero without verification against the BN254 prime field order. If this value exceeds the field order, it prevents transaction execution through the `SAMM` contract. While both states can be resolved if the associated multisig updates the variables [1, 2], the lack of input validation introduces an initial DoS risk.

Consider implementing checks in the `setup` function to ensure that the `relayer` and `membersRoot` arguments are constrained to valid inputs.

Update: *The Oxorio team implemented a fix in PR #7. However, this fix has not been evaluated by the OpenZeppelin team.*

L-02 Incomplete Transaction Validation

The `ModuleGuard` interprets the first 4 bytes of the `data` buffer as a function signature to decide whether the transaction is allowed. The `SAMM` contract follows the same pattern. In both cases, if the `data` buffer has less than four bytes, the eventual function invocation will not match the validation.

In practice, this means the `SAMM` contract (and any other module protected by the `ModuleGuard`) will be able to execute the `fallback` or `receive` function on any valid destination contract. There is also a more obscure possibility: the module could send a prefix (for example, the first two bytes) of a valid selector, which may have a different effect on the destination contract if it were constructed manually without using the standard Solidity or Vyper compilers.

Consider checking the length of the `data` buffer so that the validation exactly matches the eventual function invocation.

Update: *Acknolwedged, not resolved. The Oxorio team stated:*

> We decided not to add this check in the current development iteration.

L-03 Inconsistent Field Sizes

Within `lib.nr`, the `MAX_EMAIL_HEADER_LENGTH` and `MAX_EMAIL_FIELD_LENGTH` constants are defined, specifying the maximum email header size and the maximum header field size, respectively. However, the field size is defined to be greater than the total header size, which is inconsistent.

Consider reducing the `MAX_EMAIL_FIELD_LENGTH` size.

Update: *The Oxorio team implemented a fix in PR #3. However, this fix has not been evaluated by the OpenZeppelin team.*

L-04 Floating Compiler Version in Noir Codebase

The `>=0.35.0` compiler version specified throughout the Noir codebase allows the protocol to be compiled with various compiler versions. This variability can lead to inconsistencies in behavior during testing and the generation of verifier contracts, as different compiler versions might introduce changes or bugs that affect the protocol.

Consider explicitly specifying a fixed compiler version in the codebase to ensure consistent compilation and predictable behavior across environments.

Update: *The Oxorio team implemented a fix in PR #3. However, this fix has not been evaluated by the OpenZeppelin team.*

L-05 Allowance System Ignores Operation Type

The `_checkTxAllowance` function is designed to validate the execution of calls within the `SAMM` contract. However, it does not differentiate between operation types, such as a `delegatecall` or a regular `call`. This lack of specificity means that the validation process may not align with the intended restrictions, as different types of calls have distinct implications for contract security and behavior.

Consider modifying the `_checkTxAllowance` function to allow users to specify the permitted types of operations (e.g., `call`, `delegatecall`) for each `txId`.

Update: *The Oxorio team implemented a fix in PR #7. However, this fix has not been evaluated by the OpenZeppelin team.*

L-06 Potential Use of Revoked Public Key Hashes With Updated DKIM Registry

In the `SAMM` contract, the `setDKIMRegistry` function allows updating the DKIM registry through the Safe multisig. This function changes the registry used for DKIM-related operations. However, if the DKIM registry is updated, there is a risk that revoked public key hashes from the previous DKIM registry could still be utilized.

Consider documenting this scenario to ensure that users are informed about the importance of verifying the new DKIM registry contract, particularly regarding any canceled public key hashes for domains.

Update: *Acknolwedged, not resolved.*

L-07 Use of Non-Audited `Base64` Library

The codebase uses a `Base64` library which does not appear to have been audited and, therefore, could introduce potential risks.

Consider using an audited solution instead, such as OpenZeppelin's `Base64` library.

***Update:** Acknolwedged, not resolved. The Oxorio team stated:*

> We decided not to modify the library in the current development iteration.

L-08 No Event Emitted for Allowed Transactions Added During Setup

In the `setup` function of the `SAMM` contract, there is functionality to whitelist contract addresses and function signatures that the `SAMM` module can interact with on behalf of the Safe multisig. The `setup` function does not emit events for added `allowedTxs`, unlike the `setTxAllowed` function, which properly logs updates to allowed transactions. This may result in discrepancies between the on-chain state and the off-chain components.

Consider moving the `setTxAllowed` function logic into an `internal` function which emits the `TxAllowanceChanged` event and can be utilized in the `setup` function.

***Update:** The Oxorio team implemented a fix in PR #7. However, this fix has not been evaluated by the OpenZeppelin team.*

L-09 Duplicate Transaction IDs Can Overwrite Each Other During Setup

In the `setup` function of the `SAMM` contract, the `txAllowances` array does not enforce uniqueness for its elements. As a result, duplicate `txId` entries can cause allowances to be overridden, potentially leading to an unintended protocol state.

Consider checking the Boolean return of `s_allowedTxs.add()` to ensure unique values. Furthermore, consider implementing an `internal` function for the set transaction allowance logic that is reused during setup and in the `setTxAllowed` function.

***Update:** The Oxorio team implemented a fix in PR #7. However, this fix has not been evaluated by the OpenZeppelin team.*

L-10 Misleading and Incomplete Documentation

Throughout the SAMM-contracts codebase, the following instances of misleading and incomplete comments were identified:

- In `SAMM.sol`:
  - The documentation of `txAllowances` only refers to the `address`/`to` field and a `selector` field. However, the corresponding code makes use of an additional `amount` field.
  - The `executeTransaction` and `executeTransactionReturnData` functions refer to the undefined concept of `hash approval amount` in addition to the number of proofs threshold. This part of the comment does not seem to correspond to the code.
  - Additionally, in the `executeTransaction` function, the deadline [1, 2] has been described as an exclusive bound (to execute it before the deadline), whereas it is an inclusive bound (can be executed on the deadline).
- The `DKIMRegistry` contract comments describe two mechanisms for generating a public key hash. Neither of them exactly matches the Noir code:
  - The 1024-bit keys are split into 9 chunks of 31 bytes or 248 bits, but not 242 bits.
  - The 2048-bit keys are split into 16 chunks and not 17 chunks.
  - Additionally, the referenced `EmailVerifier.sol` file does not exist.
- The `ModuleGuard` states that the `isTxAllowed` mapping is for the SAMM module. However, the usage with a single SAMM module contradicts the inclusion of a `module` term since there will be only one module.

Furthermore, in the SAMM circuits codebase, the main entry point of the `samm_2048` and the `samm_1024` circuits contains several public and private input signals as well as public output signals. For each signal, consider documenting the semantic meaning of each input and output and a short justification of why it is public or private.

Consider correcting and completing every instance of imprecise or misleading documentation.

Update: *The Oxorio team implemented a partial fix in PR #8. The Oxorio team stated:*

> We updated comments in the smart contract code, but did not add comments in the circuit code since a detailed description of public and private signals is already provided in the separate technical documentation.

L-11 The Codebase Contains Magic Numbers

Throughout the SAMM-circuits codebase, multiple instances of data being split into an arbitrary number of chunks for hashing were identified:

- The message hash is split into 2 22-byte chunks.
- The 1024-bit public key is split into 9 chunks.
- The 2048-bit public key is split into 16 chunks, although the key consists of 18 field elements.
- The emails are split into 4 chunks.

Consider thoroughly documenting the reasoning for deciding on the number of chunks.

Similarly, the SAMM-contracts codebase contains the following magic numbers:

- 124 as a limit to the email address length which is also used when allocating space in the pub signals array.
- 44 as a limit of the `msgsHash64` length.

In all the aforementioned instances of magic number usage, consider using constants and sufficiently documenting them.

Update: *Acknolwedged, not resolved. The Oxorio team stated:*

> We will add to the documentation for relayer operators (out of scope for this audit).

Notes & Additional Information

N-01 Code Simplifications

Throughout the codebase, the following opportunities for code simplification were identified:

- The `isDKIMPublicKeyHashValid` function can be rewritten as `return !revokedDKIMPublicKeyHashes[publicKeyHash] && dkimPublicKeyHashes[domainName][publicKeyHash]`.
- The `SAMM` contract inherits a `Singleton` contract which states that the singleton should occupy a full storage slot. Instead of relying on a comment, consider enforcing

this behavior by introducing a dummy `uint96` variable that consumes the rest of the slot.
- Within `compute_merkle_root`, given that the Merkle tree has a fixed `LEVELS` height, the loop can go to `LEVELS` instead of `siblings.len()`, which is of size `LEVELS`.
- Within `getPubSignals`, instead of the second type casting of `bytes(relayer)`, the existing `relayerBytes` variable can be used.
- The typecasting of `bytes(relayer).length` to `uint256` in `getPubSignals` is redundant because `bytes.length` is already a `uint256`.

Consider applying the above suggestions to improve the clarity of the codebase.

Update: *Acknolwedged, not resolved. The Oxorio team stated:*

> We decided not to fix code style issues in this development iteration.

N-02 Code Inefficiencies

Throughout the codebase, multiple instances of inefficient code were identified:

- The `setDKIMPublicKeyHashes` function calls `setDKIMPublicKeyHash` in a loop. This means that the `onlyOwner` modifier is invoked at the start of the function and then once per hash. Consider moving the logic to an `internal` function so it can be called without invoking the modifier.
- Within `_checkNProofs`, a loop checks the uniqueness of each commitment. Consider simplifying this by requiring that proofs are submitted sorted by the commitment. Then, each commitment just needs to be checked to be strictly greater than the previous one.
- The `_stringEq` is inefficient due to having a redundant memory expansion through the `abi.encodePacked` per arguments `a` and `b`. Instead, consider implementing the `equal` function of OpenZeppelin's `String` library. Furthermore, consider whether the function can be removed entirely as it does not appear to be used.
- The `SAMM` contract's storage layout is inefficient:
  - `s_threshold` has type `uint64`. However, it is packed with `s_safe`, which is 20 bytes, leaving 32 bits unused. It could be beneficial to increase its size to `uint96` to utilize the entire available space.
  - The `nonce` variable has size `uint256`. However, it can be packed with the `s_dkimRegistry` address if its size is reduced to `uint96`, which would still allow for up to about $7.9 \cdot 10^{28}$ transactions. Assuming an increment of one and that the proof generation takes one minute, it would take a sufficiently long time to reach the maximum value.

- Currently, the off-chain relayer is designed to receive the members' emails and generate a proof per email. Once the threshold of emails is proven, the transaction can be executed. Thus, the `SAMM` contract checks whether the threshold is met and each proof is checked individually. Assuming that on-chain computation costs are higher than those of off-chain computation, this design seems inefficient. Consider having the relayer wait until the threshold number of emails have been received and jointly proving the transaction. This would produce one proof that can then be verified on-chain.
- The current protocol design uses the `s_relayer` storage variable as a public signal for the circuit. In the `SAMM` contract, this value is split into chunks of 1-byte elements of a `bytes32` array and stored in memory within the `getPubSignals` function. This processed data is then passed to the verifier contract. Consequently, this approach incurs costs associated with parsing an email address that can potentially be up to 124 bytes in length, memory expansion, and additional computation within the verifier contract. To optimize, consider using a hash of the `s_relayer` value to reduce the memory consumption to a single 32-byte word instead of up to 124 bytes. The preimage of the `s_relayer` value can then be provided as a private signal to the circuit to ensure the correctness and consistency of the proof.
- In the `checkNProofs` function, the `s_dkimRegistry` storage variable is used in the loop, consider caching the value to save some gas on the storage read.
- The `checkModuleTransaction` function calculates and returns `moduleTxHash` using the `keccak256` hash function, potentially with a large argument size. However, this value is not used in the subsequent logic. Consider returning a constant value instead to reduce execution costs.
- The identification of the `from` and `to` fields and both email addresses (inside those fields) is streamlined by using caller-specified sequence objects pinpointing the exact location. However, the identification of the `subject` field utilizes header search for the relevant string. Consider optimizing the number of constraints by migrating to a sequence object. This revision maintains technical rigor while improving readability and flow.

Consider implementing the above suggestions to increase the efficiency of the codebase.

***Update:** Acknolwedged, not resolved. The Oxorio team stated:*

> We decided not to fix code style issues in this development iteration.

N-03 Code Redundancy

Throughout the codebase, multiple instances of redundant code were identified:

- Several functions in the `SAMM` contract may only be called by the Safe account. This access control pattern is repeated across multiple functions. Consider implementing this logic with a modifier instead.
- Throughout the `SAMM` contract, the `txId` is computed as a packed encoded address and function selector. Consider moving this logic into a dedicated `private` function, given that encoding mismatches are a common source of issues.

Consider applying the above suggestions to improve the clarity and maintainability of the codebase.

Update: *Acknolwedged, not resolved. The Oxorio team stated:*

> We decided not to fix code style issues in this development iteration.

N-04 Unused Contract

The `ModuleGuard` contract appears to be unused. Instead, its functionality has been implemented in the `SAMM` contract.

Consider documenting the reason for having the `ModuleGuard` contract or removing it if it is redundant.

Update: *Acknolwedged, not resolved. The Oxorio team stated:*

> We decided not to fix code style issues in this development iteration.

N-05 Event Parameter for DKIM Registry Is Not Indexed

In the `setup` function of the `SAMM` contract, the `dkimRegistry` parameter is part of the `Setup` event. However, it is not indexed. This makes it difficult to efficiently search for `SAMM` contracts that use a specific `dkimRegistry` when querying blockchain logs.

Consider indexing the `dkimRegistry` parameter in the `Setup` event.

Update: *Acknolwedged, not resolved. The Oxorio team stated:*

> We decided not to fix code style issues in this development iteration.

N-06 Typographical Errors

Throughout the codebase, multiple instances of typographical errors were identified:

- In line 158 and 184 of `SAMM.sol`, "one of the proof" should be "one of the proofs".
- In line 472 of `SAMM.sol`, "uniq" should be "unique".

Consider correcting any typographical errors to improve the readability and clarity of the codebase.

Update: *Acknolwedged, not resolved. The Oxorio team stated:*

> We decided not to fix code style issues in this development iteration.

N-07 Naming Suggestions

Throughout the codebase, multiple opportunities for improved naming were identified:

- The owner account of `DKIMRegistry` is called `_signer` but does not do any signing itself. Consider naming the variable `_owner`.
- The `TxAllowanceChanged` event should be named `TxAllowedChanged` to avoid confusion with the `AllowanceChanged` event. This also applies to the `IModuleGuardEvents` interface.
- The `last_angle_bracket` variable name [1, 2] is confusing. This is because it is not the last angle bracket, which would be the closing bracket after the email address, but the last *opening* angle bracket.
- The allowance naming within the `SAMM` contract gives the wrong idea about the implemented mechanism. The allowances that are defined are never consumed as in the ERC-20 standard. Instead, the allowance defines a maximum value that can be transferred per transaction ID. Thus, consider naming the variable `maximumTxValue` or something similar.

Consider applying the above suggestions to improve the clarity and maintainability of the codebase.

Update: *Acknolwedged, not resolved. The Oxorio team stated:*

> We decided not to fix code style issues in this development iteration.

Conclusion

The reviewed code introduces the Safe Anonymous Mail Module (SAMM) by Oxorio, implemented using Solidity contracts and Noir circuits. During the review, we identified several areas requiring further development to achieve the level of maturity necessary for production deployment. Notably, the implementation would benefit from enhancements to improve security and anonymity. Key concerns include the absence of a check for the DKIM public key corresponding to a participant's domain, partial de-anonymization through domain exposure, unprovable emails, and the potential for DoS attacks on transactions. Additionally, issues related to code quality and readability were highlighted.

We recommend incorporating the improvements outlined in this report, followed by another round of internal reviews and an external audit. These steps will help ensure the system is secure, robust, and fully anonymous before deployment. We greatly appreciated the collaborative interactions with the Oxorio team throughout this engagement. Their responsiveness and solution-oriented approach to technical discussions were commendable. Furthermore, the accompanying technical documentation was invaluable in understanding the high-level architecture of the SAMM.