



# MobileCoin Confidential Tokens

## Security Assessment

August 3, 2022

*Prepared for:*

**Sara Drakeley**

MobileCoin

*Prepared by:* **Samuel Moelius and Jaime Iglesias**

# About Trail of Bits

---

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 80+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at <https://github.com/trailofbits/publications>, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow [@trailofbits](#) on Twitter and explore our public repositories at <https://github.com/trailofbits>. To engage us directly, visit our "Contact" page at <https://www.trailofbits.com/contact>, or email us at [info@trailofbits.com](mailto:info@trailofbits.com).

## **Trail of Bits, Inc.**

228 Park Ave S #80688

New York, NY 10003

<https://www.trailofbits.com>

[info@trailofbits.com](mailto:info@trailofbits.com)

# Notices and Remarks

---

## Copyright and Distribution

© 2022 by Trail of Bits, Inc.

All rights reserved. Trail of Bits hereby asserts its right to be identified as the creator of this report in the United Kingdom.

This report is considered by Trail of Bits to be public information; it is licensed to MobileCoin under the terms of the project statement of work and has been made public at MobileCoin's request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of Trail of Bits.

## Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

# Table of Contents

---

<b>About Trail of Bits</b>	<b>1</b>
<b>Notices and Remarks</b>	<b>2</b>
<b>Table of Contents</b>	<b>3</b>
<b>Executive Summary</b>	<b>4</b>
<b>Project Summary</b>	<b>5</b>
<b>Project Goals</b>	<b>6</b>
<b>Project Targets</b>	<b>7</b>
<b>Project Coverage</b>	<b>8</b>
<b>Codebase Maturity Evaluation</b>	<b>10</b>
<b>Summary of Findings</b>	<b>12</b>
<b>Detailed Findings</b>	<b>13</b>
1. Project contains vulnerable dependencies	13
2. MobileCoin Foundation could infer token IDs in certain scenarios	15
3. Token IDs are protected only by SGX	17
4. Nonces are not stored per token	19
5. Clients have no option for verifying blockchain configuration	21
6. Confidential tokens cannot support frequent price swings	22
7. Overflow handling could allow recovery of transaction token ID	24
<b>Summary of Recommendations</b>	<b>26</b>
<b>A. Vulnerability Categories</b>	<b>27</b>
<b>B. Code Maturity Categories</b>	<b>29</b>
<b>C. Non-Security-Related Findings</b>	<b>31</b>

# Executive Summary

---

## Engagement Overview

MobileCoin engaged Trail of Bits to review the security of its confidential tokens feature. From July 11 to July 15, 2022, a team of two consultants conducted a security review of the client-provided source code, with two person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

## Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with full knowledge of the target system, including access to the source code and documentation. We performed static and manual analysis of the target system.

## Summary of Findings

The audit uncovered only minor flaws that could impact system confidentiality, integrity, or availability. A summary of the findings is provided below.

### EXPOSURE ANALYSIS

<i>Severity</i>	<i>Count</i>
High	0
Medium	1
Low	1
Informational	4
Undetermined	1

### CATEGORY BREAKDOWN

<i>Category</i>	<i>Count</i>
Access Controls	1
Authentication	1
Data Exposure	2
Denial of Service	2
Patching	1

# Project Summary

---

## Contact Information

The following managers were associated with this project:

**Dan Guido**, Account Manager  
[dan@trailofbits.com](mailto:dan@trailofbits.com)

**Mary O'Brien**, Project Manager  
[mary.obrien@trailofbits.com](mailto:mary.obrien@trailofbits.com)

The following engineers were associated with this project:

**Samuel Moelius**, Consultant  
[samuel.moelius@trailofbits.com](mailto:samuel.moelius@trailofbits.com)

**Jaime Iglesias**, Consultant  
[jaime.iglesias@trailofbits.com](mailto:jaime.iglesias@trailofbits.com)

## Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
June 3, 2022	Architecture discussion
July 7, 2022	Pre-project kickoff call
July 18, 2022	Delivery of report draft
July 18, 2022	Report readout meeting
July 19, 2022	Delivery of corrections to errors identified during readout
August 3, 2022	Delivery of final report

# Project Goals

---

The engagement was scoped to provide a security assessment of the MobileCoin confidential tokens feature. Specifically, we sought to answer the following non-exhaustive list of questions:

- Does the confidential tokens feature introduce runaway minting bugs?
- Does the confidential tokens feature introduce double spending bugs?
- Are there any ways in which token IDs could be revealed?
- Are there any ways in which funds could be lost?
- Could burned tokens be spent?
- For mint transactions, is the chain of trust from the trusted root checked correctly?
- Are the token-specific generators calculated in constant time?
- Are mint and mint configuration nonces handled correctly?
- Are transaction priorities properly computed?
- Are token fees computed and paid correctly?

# Project Targets

---

The engagement involved a review and testing of the following target.

## MobileCoin

Repository	<a href="https://github.com/mobilecoinfoundation/mobilecoin">https://github.com/mobilecoinfoundation/mobilecoin</a>
Version	<a href="#">196ca2eb14a3c4f377e8e471f80070dafde40f4a</a> (tag v1.2.1) <a href="#">06e09b3e0ffac7f8b6845c9e09bfe992f72a5f59</a> (pull/2215)
Directories	<code>transaction/*</code> <code>consensus/enclave/*</code> <code>consensus/service/*</code> <code>crypto/digestible/*</code> (not reviewed) <code>crypto/multisig/*</code> (not reviewed) <code>account-keys/src/burn_address.rs</code> (not reviewed) <code>consensus/mint-client/*</code> (not reviewed) <code>mint-auditor/*</code> (not reviewed)
Type	Rust
Platform	Intel SGX



# Project Coverage

---

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches and their results include the following:

- **Review of the MobileCoin Improvement Proposals (MCIPs) and white paper**
  - We reviewed the provided documentation both to understand the implemented features and to look for potential shortfalls in the specification.
- **Static analysis with cargo-audit and Clippy**
  - We ran cargo-audit over all of the lockfiles and Clippy with `-W clippy::pedantic` enabled. We reviewed the results of each.
- **Review of test coverage**
  - We ran the tests normally, in **random order**, and with **AddressSanitizer** enabled. We also computed the tests' code coverage using `cargo-llvm-cov` and looked for gaps in the code related to the confidential tokens feature.
- **Manual review**
  - We manually reviewed the contents of the `transaction`, `consensus/enclave`, and `consensus/service` directories, with a focus on changes relevant to the confidential tokens feature.

## Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- Due to time constraints, the following directories, while in scope, were not reviewed:
  - `crypto/digestible/*`
  - `crypto/multisig/*`
  - `account-keys/src/burn_address.rs`
  - `consensus/mint-client/*`
  - `mint-auditor/*`

- Our automated analysis efforts were limited to static analysis via cargo-audit and Clippy and to running the tests (under various configurations). In particular, we performed no fuzzing.
- We used only a manual review to check that operations are performed in constant time. A more thorough approach would involve a trace analysis, similar to what we did for the MobileCoin Fog assessment.

# Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	We identified one problem related to arithmetic that could allow token IDs to be revealed (TOB-MCCT-7).	Moderate
Auditing	We were unable to review code related to auditing of minting and burning operations.	Further Investigation Required
Authentication / Access Controls	We found no significant issues regarding authentication or access controls.	Satisfactory
Complexity Management	The use of MCIPs demonstrates a disciplined development process. However, we found one issue related to the project's dependency management (TOB-MCCT-1).	Moderate
Cryptography and Key Management	We identified no problems related to cryptography or key management. However, we were unable to review the relevant parts of the crypto directory.	Further Investigation Required
Decentralization	Minting and token governance are centrally controlled. All minting configurations must chain back to a centrally controlled MINTING_TRUST_ROOT_PRIVATE_KEY.	Moderate
Documentation	The MCIPs and "Confidential Tokens" white paper exceed industry standards regarding documentation. We recommend that this standard be maintained. However, we recommend adding diagrams to help users better	Strong

	understand the codebase and the architecture.	
Front-Running Resistance	We found no problems related to front-running.	Satisfactory
Low-Level Manipulation	Cryptographic code and code required to run in constant time require further review.	Further Investigation Required
Testing and Verification	Code that changed because of the introduction of the confidential tokens feature appears to be adequately tested.	Satisfactory

## Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Project contains vulnerable dependencies	Patching	Undetermined
2	MobileCoin Foundation could infer token IDs in certain scenarios	Data Exposure	Informational
3	Token IDs are protected only by SGX	Access Controls	Informational
4	Nonces are not stored per token	Denial of Service	Low
5	Clients have no option for verifying blockchain configuration	Authentication	Informational
6	Confidential tokens cannot support frequent price swings	Denial of Service	Informational
7	Overflow handling could allow recovery of transaction token ID	Data Exposure	Medium

# Detailed Findings

## 1. Project contains vulnerable dependencies

Severity: Undetermined	Difficulty: High
Type: Patching	Finding ID: TOB-MCCT-1
Target: Various	

### Description

Running cargo-audit over the codebase revealed that the system under audit uses crates with Rust Security (RustSec) advisories and crates that are no longer maintained.

RustSec ID	Description	Dependency	CVSS Score
2022-0006	Data race in Iter and IterMut	thread_local 1.0.1	N/A
2020-0071	Potential segfault in the time crate	time 0.1.43	Medium
2021-0003	Buffer overflow in SmallVec::insert_many	smallvec 1.2.0	Critical
2022-0013	Regexes with large repetitions on empty sub-expressions take a very long time to parse	regex 1.3.0	High
2020-0036	failure is officially deprecated/unmaintained	failure 0.1.8	Critical
2018-0017	tempdir has been deprecated; use tempfile instead	tempdir 0.3.7	N/A

Figure 1.1: cargo-audit results

### Exploit Scenario

Eve discovers a code path leading to a vulnerable crate and uses the code path to conduct exploits such as crashing nodes and corrupting memory.

### Recommendations

Short term, update the vulnerable dependencies to safe versions and use alternatives to the unmaintained/deprecated ones.

Long term, regularly run `cargo-audit` over the codebase. Doing so can help reveal similar bugs. Additionally, consider integrating `cargo-audit` into the CI/CD pipeline.

## References

- `cargo-audit`

## 2. MobileCoin Foundation could infer token IDs in certain scenarios

Severity: Informational

Difficulty: High

Type: Data Exposure

Finding ID: TOB-MCCT-2

Target: Various

### Description

The MobileCoin Foundation is the recipient of all transaction fees and, in certain scenarios, could infer the token ID used in one of multiple transactions included in a block.

**MCIP-0025** introduced the concept of “confidential token IDs.” The rationale behind the proposal is to allow the MobileCoin network to support tokens other than MOB (MobileCoin’s native token) in the future. Doing so requires not only that these tokens be unequivocally identifiable but also that transactions involving any token, MOB or otherwise, have the same confidentiality properties.

Before the introduction of the confidential tokens feature, all transaction fees were aggregated by the enclave, which created a single transaction fee output per block; however, the same approach applied to a system that supports transfers of tokens other than MOB could introduce information leakage risks. For example, if two users submit two transactions with the same token ID, there would be a single transaction fee output, and therefore, both users would know that they transacted with the same token. To prevent such a leak of information, MCIP-0025 proposes the following:

The number of transaction fee outputs on a block should always equal the minimum value between the number of token IDs and the number of transactions in that block (e.g.,  $\text{num\_tx\_fee\_out} = \min(\text{num\_token\_ids}, \text{num\_transactions})$ ). This essentially means that a block with a single transaction will still have a single transaction fee output, but a block with multiple transactions with the same token ID will have multiple transaction fee outputs, one with the aggregated fee and the others with a zero-value fee.

Finally, it is worth mentioning that transaction fees are not paid in MOB but in the token that is being transacted; this creates a better user experience, as users do not need to own MOB to send tokens to other people.

While this proposal does indeed preserve the confidentiality requirement, it falls short in one respect: the receiver of all transaction fees in the MobileCoin network is the MobileCoin Foundation, meaning that it will always know the token ID corresponding to a transaction fee output. Therefore, if only a single token is used in a block, the foundation will know the token ID used by all of the transactions in that block.



## Exploit Scenario

Alice and Bob use the MobileCoin network to make payments between them. They send each other multiple payments, using the same token, and their transactions are included in a single block.

Eve, who has access to the MobileCoin Foundation's viewing key, is able to decrypt the transaction fee outputs corresponding to that block and, because no other token was used inside the block, is able to infer the token that Alice and Bob used to make the payments.

## Recommendations

Short term, document the fact that transaction token IDs are visible to the MobileCoin Foundation. Transparency on this issue will help users understand the information that is visible by some parties.

Additionally, consider implementing the following alternative designs:

- Require that all transaction fees be paid in MOB.

This solution would result in a degraded user experience compared to the current design; however, it would address the issue at hand.

- Aggregate fee outputs across multiple blocks.

This solution would achieve only “probabilistic confidentiality” of information because if all those blocks transact in the same token, the foundation would still be able to infer the ID.

Long term, document the trade-offs between allowing users to pay fees in the tokens they transact with and restricting fee payments to only MOB, and document how these trade-offs could affect the confidentiality of the system.

### 3. Token IDs are protected only by SGX

Severity: Informational

Difficulty: High

Type: Access Controls

Finding ID: TOB-MCCT-3

Target: consensus/enclave/impl/src/lib.rs

#### Description

Token IDs are intended to be confidential. However, they are operated on within an SGX enclave. This is an apparent departure from MobileCoin's previous approach of using SGX as an additional security mechanism, not a primary one.

Previously, most confidential information in MobileCoin was protected by SGX *and another security mechanism*. Examples include the following:

- A transactions' senders, recipients, and amounts are protected by SGX and ring signatures.
- The transactions a user interacts with through Fog are protected by both SGX and oblivious RAM.

However, token IDs are protected by SGX alone. (An example in which a token ID is operated on within an enclave appears in figure 3.1.) Thus, the incorporation of confidential tokens seems to represent a shift in MobileCoin's security posture.

```
let token_id = TokenId::from(tx.prefix.fee_token_id);

let minimum_fee = ct_min_fees
    .get(&token_id)
    .ok_or(TransactionValidationError::TokenNotYetConfigured)?;
```

Figure 3.1: [consensus/enclave/impl/src/lib.rs#L239-L243](#)

#### Exploit Scenario

Mallory learns of a vulnerability that allows her to see inside of an SGX enclave. Mallory uses the vulnerability to observe the token IDs used in transactions in a MobileCoin enclave that she runs.

#### Recommendations

Short term, document the fact that token IDs are not offered the same level of security as other aspects of MobileCoin. This will help set users' expectations regarding the confidentiality of their information (i.e., whether it could be revealed to an attacker).

Long term, continue to investigate solutions to the security problems surrounding the confidential tokens feature. A solution that does not reveal token IDs to the enclave could exist.

#### 4. Nonces are not stored per token

Severity: Low

Difficulty: High

Type: Denial of Service

Finding ID: TOB-MCCT-4

Target: Various

#### Description

Mint and mint configuration transaction nonces are not distinguished by the tokens with which they are associated. Malicious minters or governors could use this fact to conduct denial-of-service attacks against other minters and governors.

The relevant code appears in figures 4.1 and 4.2. For each type of transaction, nonces are inserted into a seen\_nonces set without regard to the token indicated in the transaction.

```
let mut seen_nonces = BTreeSet::default();
let mut validated_txs = Vec::with_capacity(mint_config_txs.len());

for tx in mint_config_txs {
    // Ensure all nonces are unique.
    if !seen_nonces.insert(tx.prefix.nonce.clone()) {
        return Err(Error::FormBlock(format!(
            "Duplicate MintConfigTx nonce: {:?}",
            tx.prefix.nonce
        )));
    }
}
```

Figure 4.1: *consensus/enclave/impl/src/lib.rs#L342-L352*

```
let mut mint_txs = Vec::with_capacity(mint_txs_with_config.len());
let mut seen_nonces = BTreeSet::default();
for (mint_tx, mint_config_tx, mint_config) in mint_txs_with_config {
    // The nonce should be unique.
    if !seen_nonces.insert(mint_tx.prefix.nonce.clone()) {
        return Err(Error::FormBlock(format!(
            "Duplicate MintTx nonce: {:?}",
            mint_tx.prefix.nonce
        )));
    }
}
```

Figure 4.2: *consensus/enclave/impl/src/lib.rs#L384-L393*

Note that the described attack could be made worse by how nonces are intended to be used. The following passage from the white paper suggests that nonces are generated

deterministically from public data. Generating nonces in this way could make them easy for an attacker to predict.

*When submitting a MintTx, we include a nonce to protect against replay attacks, and a tombstone block to prevent the transaction from being nominated indefinitely, and these are committed to the chain. (For example, in a bridge application, this nonce may be derived from records on the source chain, to ensure that each deposit on the source chain leads to at most one mint.)*

### **Exploit Scenario**

Mallory (a minter) learns that Alice (another minter) intends to submit a mint transaction with a particular nonce. Mallory submits a mint transaction with that nonce first, making Alice's invalid.

### **Recommendations**

Short term, store nonces per token, instead of all together. Doing so will prevent the denial-of-service attack described above.

Long term, when adding new data to blocks or to the blockchain configuration, carefully consider whether it should be stored per token. Doing so could help to prevent denial-of-service attacks.

## 5. Clients have no option for verifying blockchain configuration

Severity: Informational

Difficulty: High

Type: Authentication

Finding ID: TOB-MCCT-5

Target: Various

### Description

Clients have no way to verify whether the MobileCoin node they connect to is using the correct blockchain configuration. This exposes users to attacks, as detailed in the white paper:

*Similarly to how the nodes ensure that they are similarly configured during attestation, (by mixing a hash of their configuration into the responder id used during attestation), the peer- to-node attestation channels could also do this, so that users can fail to attest immediately if malicious manipulation of configuration has occurred. The problem with this approach is that the users have no particularly good source of truth around the correct runtime configuration of the services.*

The problem that “users have no particularly good source of truth” could be solved by publishing the latest blockchain configuration via a separate channel (e.g., a publicly accessible server). Furthermore, allowing users to opt in to such additional checks would provide additional security to users who desire it.

### Exploit Scenario

Alice falls victim to the attack described in the white paper. The attack would have been thwarted had Alice known that the node she connected to was not using the correct blockchain configuration.

### Recommendations

Short term, make the current blockchain configuration publicly available, and allow nodes to attest to clients using their configuration. Doing so will help security-conscious users to better protect themselves.

Long term, avoid withholding data from clients during attestation. Adopt a general principle that if data should be included in node-to-node attestation, then it should be included in node-to-client attestation as well. Doing so will help to ensure the security of users.

## 6. Confidential tokens cannot support frequent price swings

Severity: Informational

Difficulty: High

Type: Denial of Service

Finding ID: TOB-MCCT-6

Target: Various

### Description

The method for determining tokens' minimum fees has limited applicability. In particular, it cannot support tokens whose prices change frequently.

In principle, a token's minimum fee should be comparable in value to the MOB minimum fee. Thus, if a token's price increases relative to the price of MOB, the token's minimum fee should decrease. Similarly, if a token's price decreases relative to the price of MOB, the token's minimum fee should increase.

However, an enclave sets its fee map from the blockchain configuration during initialization (figure 6.1) and does not change the fee map thereafter. Thus, the enclave would seem to have to be restarted if its blockchain configuration and fee map were to change. This fact implies that the current setup cannot support tokens whose prices shift frequently.

```
fn enclave_init(
    &self,
    peer_self_id: &ResponderId,
    client_self_id: &ResponderId,
    sealed_key: &Option<SealedBlockSigningKey>,
    blockchain_config: BlockchainConfig,
) -> Result<(SealedBlockSigningKey, Vec<String>)> {
    // Check that fee map is actually well formed

    FeeMap::is_valid_map(blockchain_config.fee_map.as_ref()).map_err(Error::FeeMap)?;

    // Validate governors signature.
    if !blockchain_config.governors_map.is_empty() {
        let signature = blockchain_config
            .governors_signature
            .ok_or(Error::MissingGovernorsSignature)?;

        let minting_trust_root_public_key =
            Ed25519Public::try_from(&MINTING_TRUST_ROOT_PUBLIC_KEY[..])
                .map_err(Error::ParseMintingTrustRootPublicKey)?;

        minting_trust_root_public_key
            .verify_governors_map(&blockchain_config.governors_map, &signature)
```

```
        .map_err(|_| Error::InvalidGovernorsSignature)?;
    }

    self.ct_min_fee_map
        .set(Box::new(
            blockchain_config.fee_map.as_ref().iter().collect(),
        ))
        .expect("enclave was already initialized");
```

Figure 6.1: *consensus/enclave/impl/src/lib.rs#L454-L483*

## Exploit Scenario

MobileCoin integrates token T. The value of T decreases, but the minimum fee remains the same. Users pay the minimum fee, resulting in lost income to the MobileCoin Foundation.

## Recommendations

Short term, accept only tokens with a history of price stability. Doing so will ensure that the new features are used only with tokens that can be supported.

Long term, consider including two inputs in each transaction, one for the token transferred and one to pay the fee in MOB, as suggested in [TOB-MCCT-2](#).



## 7. Overflow handling could allow recovery of transaction token ID

Severity: **Medium**

Difficulty: **High**

Type: Data Exposure

Finding ID: TOB-MCCT-7

Target: consensus/enclave/impl/src/lib.rs

### Description

The system's fee calculation could overflow a u64 value. When this occurs, the fee is divided up into multiple smaller fees, each fitting into a u64 value. Under certain conditions, this behavior could be abused to reveal whether a token ID is used in a block.

The relevant code appears in figure 7.1. The hypothetical attack is described in the exploit scenario below.

```
loop {
    let output_fee = min(total_fee, u64::MAX as u128) as u64;
    outputs.push(mint_output(
        config.block_version,
        &fee_recipient,
        FEES_OUTPUT_PRIVATE_KEY_DOMAIN_TAG.as_bytes(),
        parent_block,
        &transactions,
        Amount {
            value: output_fee,
            token_id,
        },
        outputs.len(),
    ));
    total_fee -= output_fee as u128;
    if total_fee == 0 {
        break;
    }
}
```

Figure 7.1: [consensus/enclave/impl/src/lib.rs#L855-L873](#)

### Exploit Scenario

Mallory is a (malicious) minter of token T. Suppose B is a recently minted block whose total number of fee outputs is equal to the number of tokens, which is less than the number of transactions in B. Further suppose that Mallory wishes to determine whether B contains a transaction involving T. Mallory does the following:

1. She puts her node into its state just prior to the minting of B.

2. She mints to herself a quantity of T worth  $\lfloor u64 : : \text{MAX} / \text{min\_fee} \rfloor * \text{min\_fee}$ . Call this quantity F.
3. She submits to her node a transaction with a fee of F.
4. She allows the block to be minted.
5. She observes the number of fee outputs in the modified block, B':
  - a. If B does not contain a transaction involving T, then B contains a fee output for T equal to zero, and B' contains a fee output for T equal to F.
  - b. If B does contain a transaction involving T, then B contains a fee output for T equal to at least min\_fee, and B' contains two fee outputs for T, one of which is equal to  $u64 : : \text{MAX}$ .

Thus, by observing the number of outputs in B', Mallory can determine whether B contains a transaction involving T.

### Recommendations

Short term, require the total supply of all incorporated tokens not to exceed a u64 value. Doing so will eliminate the possibility of overflow and prevent the attack described above.

Long term, consider incorporating randomness into the number of fee outputs generated. This could provide an alternative means of preventing the attack in a way that still allows for overflow. Alternatively, consider including two inputs in each transaction, one for the token transferred and one to pay the fee in MOB, as suggested in [TOB-MCCT-2](#).

## Summary of Recommendations

---

Trail of Bits recommends that MobileCoin address the findings detailed in this report and take the following additional steps prior to deployment:

- Maintain the current industry-exceeding standard of documentation.
- Conduct an audit of the directories that were not reviewed during this engagement. Include the `ledger` directory within the scope of that audit.
- Regularly run `cargo-audit` as part of the project's CI process (TOB-MCCT-1).
- Consider including two inputs in each transaction: one for the token transferred and one to pay the fee in MOB (TOB-MCCT-2, TOB-MCCT-6, TOB-MCCT-7).
- Store nonces per token ID to remediate TOB-MCCT-4.
- Require the total supply of all incorporated tokens not to exceed a u64 value (TOB-MCCT-7).

## A. Vulnerability Categories

---

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	Description
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	Description
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	Description
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

## B. Code Maturity Categories

---

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	Description
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Front-Running Resistance	The system's resistance to front-running attacks
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	Description
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

## C. Non-Security-Related Findings

---

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

- We noticed the following typos:

- In the second of the following **two lines**, `//` should be `///`:

```
/// Seed to use when generating blocks (e.g.  
// 1234567812345678123456781234567812345678123456781234567812345678).
```

- In the following **lines**, “origianl” should be “original”:

```
return Err(Error::InvalidMintConfig(  
    "MintConfigs do not match origianl transaction".to_string(),  
));
```

- In the following **line**, there is an extra space before ``C``:

```
/// `view_public_key` - The user's public subaddress view key `C`,
```

- In the following **lines**, “maximal” could be “maximum.” The latter sounds more natural and is correct because the value is unique:

```
/// The maximal amount this configuration can mint from the moment it has  
/// been applied.
```

- There is considerable code duplication between the definitions of **add\_output** and **add\_change\_output** in `transaction/std/src/transaction_builder.rs`. We recommend that the functions’ definitions be consolidated:

```
pub fn add_output<RNG: CryptoRng + RngCore>(  
    &mut self,  
    value: u64,  
    recipient: &PublicAddress,  
    rng: &mut RNG,  
) -> Result<TxOutContext, TxBuilderError> {  
    // Taking self.memo_builder here means that we can call functions on &mut  
    self,  
    // and pass them something that has captured the memo builder.  
    // Calling take() on Option<Box> is just moving a pointer.  
    let mut mb = self  
        .memo_builder  
        .take()  
        .expect("memo builder is missing, this is a logic error");  
    ...  
pub fn add_change_output<RNG: CryptoRng + RngCore>(  
    &mut self,  
    value: u64,  
    recipient: &PublicAddress,  
    rng: &mut RNG,  
) -> Result<TxOutContext, TxBuilderError> {  
    // Taking self.memo_builder here means that we can call functions on &mut  
    self,  
    // and pass them something that has captured the memo builder.  
    // Calling take() on Option<Box> is just moving a pointer.  
    let mut mb = self  
        .memo_builder  
        .take()  
        .expect("memo builder is missing, this is a logic error");  
    ...
```



```

    &mut self,
    value: u64,
    change_destination: &ChangeDestination,
    rng: &mut RNG,
) -> Result<TxOutContext, TxBuilderError> {
    // Taking self.memo_builder here means that we can call functions on &mut
    self,
    // and pass them something that has captured the memo builder.
    // Calling take() on Option<Box> is just moving a pointer.
    let mut mb = self
        .memo_builder
        .take()
        .expect("memo builder is missing, this is a logic error");
    ...

```