



Security Audit Report

Token Wrap Solana

Delivered: June 11, 2025

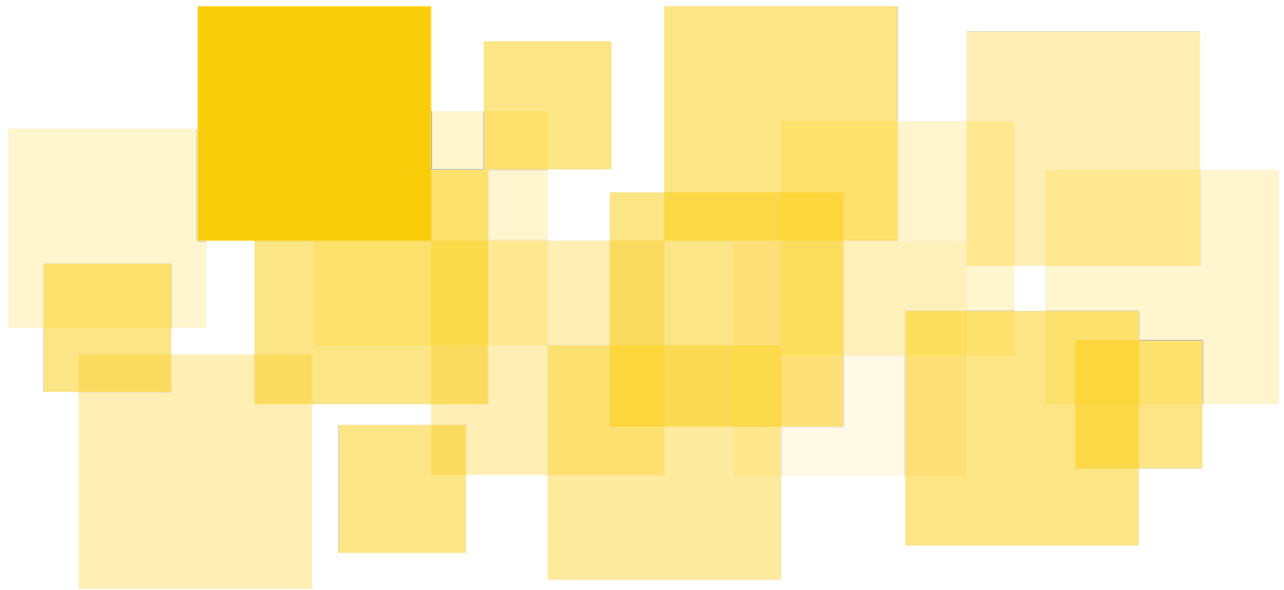




Table of Contents

- [Disclaimer](#)
- [Executive Summary](#)
- [Goal](#)
- [Scope](#)
- [Methodology and Engagement Plan](#)
 - [Audit Plan](#)
- [Platform Features and Logic Description](#)
 - [The Token Wrap Program](#)
- [Token Wrap Formal Specification](#)
 - [Definitions](#)
 - [Token Wrap Formal Specification](#)
- [SPL Token and SPL Token 2022 High-level Specifications](#)
 - [SPL Token Specification](#)
 - [SPL Token 2022](#)
- [Invariants](#)
- [Test Harness Modifications and Results](#)
- [Findings](#)
 - [\[A1\] Unwrapped Tokens Can Be Held Multiple, Distinct Escrows](#)
 - [\[A2\] Token 2022 Extensions May Cause Issues When Wrapped](#)
- [Informative Findings](#)
 - [\[B1\] Best Practices Recommendations](#)
- [Final Considerations](#)



Disclaimer

This report does not constitute legal or investment advice. You understand and agree that this report relates to new and emerging technologies and that there are significant risks inherent in using such technologies that cannot be completely protected against. While this report has been prepared based on data and information that has been provided by you or is otherwise publicly available, there are likely additional unknown risks which otherwise exist. This report is also not comprehensive in scope, excluding a number of components critical to the correct operation of this system. This report is for informational purposes only and is provided on an "as-is" basis and you acknowledge and agree that you are making use of this report and the information contained herein at your own risk. The preparers of this report make no representations or warranties of any kind, either express or implied, regarding the information in or the use of this report and shall not be liable to you or any third parties for any acts or omissions undertaken by you or any third parties based on the information contained herein.

Smart contracts are still a nascent software arena, and their deployment and public offering carries substantial risk.

Finally, the possibility of human error in the manual review process is very real, and we recommend seeking multiple independent opinions on any claims which impact a large quantity of funds.



Executive Summary

Anza engaged [Runtime Verification Inc.](#) to perform a security audit of its smart contract. The audit was conducted between April 30 and May 28, 2025. The objective was to assess the implementation's security and correctness, identify exploitable vulnerabilities, and provide recommendations to enhance the system's reliability.

The Token Wrap program was developed aiming to solve the interoperability issues between different token types in Solana. It solves it by enabling users to wrap tokens into either SPL Token or SPL Token 2022 adherent tokens.

The audit process involved a comprehensive smart contract codebase review, focusing on manual inspection and formal verification techniques. Runtime Verification utilized several techniques related to formal specification generation, invariant analysis, and testing to rigorously test the system's behavior under various conditions. This included the development and analysis of key invariants to ensure the system's integrity across all state transitions.

The audit led to the identification of issues of potential severity for the protocol's health, which have been identified as follows:

- Potential threats to the contract's fund integrity: [\[A1\] Unwrapped Tokens Can Be Held Multiple, Distinct Escrows](#), [\[A2\] Token 2022 Extensions May Cause Issues When Wrapped](#);
- Potential code/logic malfunction: [\[A2\] Token 2022 Extensions May Cause Issues When Wrapped](#).

In addition, several informative findings and general recommendations have also been made, including:

- Best practices and code optimization-related particularities: [\[B1\] Best Practices Recommendations](#).

All findings have been either addressed or partially addressed by the client, along with suggested mitigations to improve the security and maintainability of the Token Wrap



system. If not fully addressed, the pending topics for that finding have been considered not threatening to the health of the project or Solana's ecosystem. Any responses or fixes submitted by the client have been reviewed and noted as part of the report's finalization.



Goal

The goal of the audit is threefold:

- Review the high-level business logic (protocol design) of the Token Wrap program based on the provided documentation and code;
- Review the low-level implementation of the system for the individual Solana smart contract;
- Analyze the integration between abstractions of the modules interacting with the contract in the scope of the engagement and reason about possible exploitative corner cases.

The audit focuses on identifying issues in the system's logic and implementation that could potentially render the system vulnerable to attacks or cause it to malfunction.

Furthermore, the audit highlights informative findings that could be used to improve the safety and efficiency of the implementation.



Scope

The scope of this audit is limited to the code contained in a single public GitHub repository provided by the Anza team. The SLP Token Wrap contract was identified within this repository as the primary artifact under review for this engagement.

- Token Wrap Repository(public)
 - <https://github.com/solana-program/token-wrap>
 - Commit: `dd71fc10c651b07b7d62b151021216e5321b1789`
 - `program/src` : Core files specifying the business logic behind the minting, wrapping, and unwrapping within the Token Wrap contracts.

The codebase under review consists of approximately 600 lines of Rust code written for the Solana ecosystem. In preparing for the audit, Runtime Verification referenced comments provided in the code, publicly available documentation, and supplemental materials shared by the Anza team.

The audit is strictly limited to the artifacts listed above. Off-chain components, frontend logic, deployment infrastructure, and third-party integrations are outside the scope of this engagement.

Commits addressing any findings presented in this report have also been reviewed to verify that identified issues were appropriately addressed prior to report finalization.



Methodology and Engagement Plan

The audit lasted four calendar weeks, and each phase was designed to identify and validate both high-level and low-level security concerns. We followed a structured and thorough approach to maximize the effectiveness of this audit engagement within the agreed-upon timeframe.

To facilitate our understanding of the platform's behavior, higher-level representations of the Rust codebase were created, including:

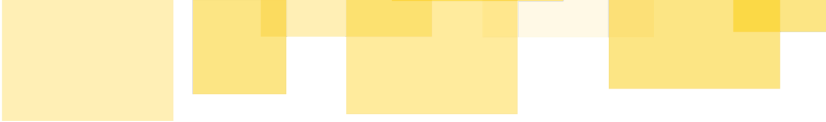
- Modeled sequences of logical operations, considering the limitations enforced by the identified invariants, checking if all desired properties hold for any possible input value;
- Manually built high-level function call maps, aiding the comprehension of the code and organization of the protocol's verification process;
- Created abstractions of the elements outside of the scope of this audit to build a complete picture of the protocol's business logic in action;
- Explored static analyzers and testing tools in a time-boxed effort to identify potential issues identifiable through different knowledge bases.

This approach enabled us to systematically check consistency between the logic and the provided Solana Rust implementation of the system.

On the tooling side, we used [l3x](#), [x-ray](#), [Radar](#), and [cargo audit](#) to look for potential issues, and while we experienced technical difficulties with l3x and x-ray, Radar did not report any found issues, and cargo audit reported three found vulnerabilities, but they were all related to libraries used by the Solana infrastructure needed for your contracts. All potentially useful outputs of these tools have been provided to the client.

Although outside of scope initially, we also explored the project's testing harness and structure, and provided insights on how they can be improved for more flexibility and to become capable of developing more convoluted testing scenarios.

Finally, we conducted rounds of internal discussions with security experts over the code and platform design, aiming to verify possible exploitation vectors and identify



improvements for the analyzed contracts.

Additionally, given the nascent Solana development and auditing community, we reviewed [this list](#) of known Ethereum security vulnerabilities and attack vectors and checked whether they apply to the smart contracts and scripts; if they apply, we checked whether the code is vulnerable to them.

Audit Plan

Week 1

- Design review and business logic comprehension;
- Investigation of static analysis tools;
- High-level code review;
- Review the Token Wrap Contract and adjacent specifications (SPL Token and Token 2022).

Week 2

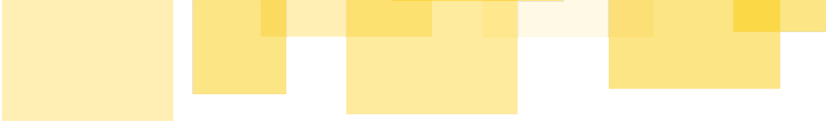
- Creation of the formal specification of the Token Wrap program;
- Formulate abstractions of the programs interacting with the Token Wrap program;
- Use the formal specs to derive the contract invariants;
- Report potential issues related to incomplete testing suites.

Week 3

- Low-level code review;
- Validation of the invariants identified against the actual implementation of the contract;
- Identifying potential exploits based on low-level implementation or blockchain-related particularities.

Week 4

- Finalization of pending analysis;
- Validate the coverage of the testing suite;

- 
- If possible, add tests to validate and demonstrate the invariants;
 - Reasoning on the implementation of potential fixes, as well as reviewing any provided code addressing findings;
 - Curation of the report.

It was possible to follow the audit plan, with an additional time requested by the auditors to validate suspicions related to the abstractions of the SPL Token and Token 2022 standards.



Platform Features and Logic Description

The Token Wrap program is a Solana smart contract and associated tools designed to create wrapped versions of existing SPL tokens. Its main goal is to improve interoperability between the SPL Token and the SPL Token 2022 (also referred to as Token 2022, or simply Token22) standards.

The SPL Token standard, being the first introduced in the Solana ecosystem, provided a robust foundation for creating fungible and non-fungible tokens. However, as the blockchain space evolved and in contrast with other blockchains that implemented more feature-rich token standards, it became increasingly clear a need for more sophisticated features that weren't easily implementable within the original standard without breaking backward compatibility or requiring complex workarounds. This led to the creation of SPL Token 2022, which essentially mirrors the core functionalities of the original but introduces a powerful system of "extensions". These extensions allow for new features like transfer fees, confidential transfers (privacy-preserving transactions), interest-bearing tokens, and more.

The necessity for token wrapping arises because applications, wallets, and protocols often need to interact seamlessly with both older SPL tokens and newer Token 2022 tokens, some of which might possess extensions. Wrapping allows a token conforming to one standard to be represented as a token of another standard, maintaining a one-to-one backing. This ensures that a token with enhanced features can still be used in older applications, or conversely, an older token can be given a "wrapper" that allows it to interact with newer applications designed for Token 2022's capabilities. Without wrapping, the ecosystem faces significant fragmentation, hindering liquidity and forcing developers to build separate integrations for each token type, limiting the composability and overall utility of assets on Solana.

This interoperability concept is summarized in Figure 1.

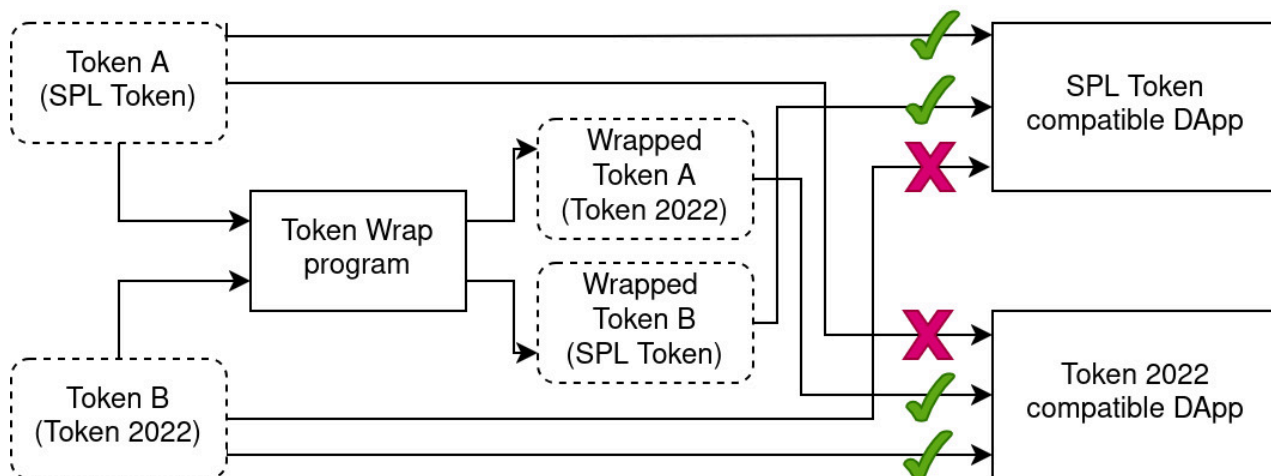


Figure 1: Solana token standards interoperability issues diagram, and the capability of the Token Wrap program to solve these issues.

With Anza's Token Wrap program, users/programs can wrap tokens implemented following the SPL Token or Token 2022 standard into wrapped versions of these tokens, which, in turn, are tokens following either the SPL Token or Token 2022 (without extensions) standard.

The Token Wrap Program

The Token Wrap program itself supports three distinct instructions:

- `createMint` : Initializes a token mint representing the wrapped token and a backpointer that stores information about the unwrapped token. This wrapped token can follow either the SPL Token or Token 2022 standards;
- `wrap` : processes a transfer of unwrapped tokens to an escrow and mints wrapped tokens to a recipient;
- `unwrap` : processes a transfer of unwrapped tokens from an escrow to a recipient, and burns the wrapped tokens from a source.

For the `wrap` and `unwrap` operations, the number of unwrapped tokens transferred and the number of minted/burnt wrapped tokens according to the operation are equal, guaranteeing a one-to-one backing mechanism of unwrapped to wrapped tokens.

In Figure 2, we display the instruction interface of the Token Wrap program, together with the forwarded instruction information (accounts and parameters) and account state changes consequent to the successful execution of each instruction.

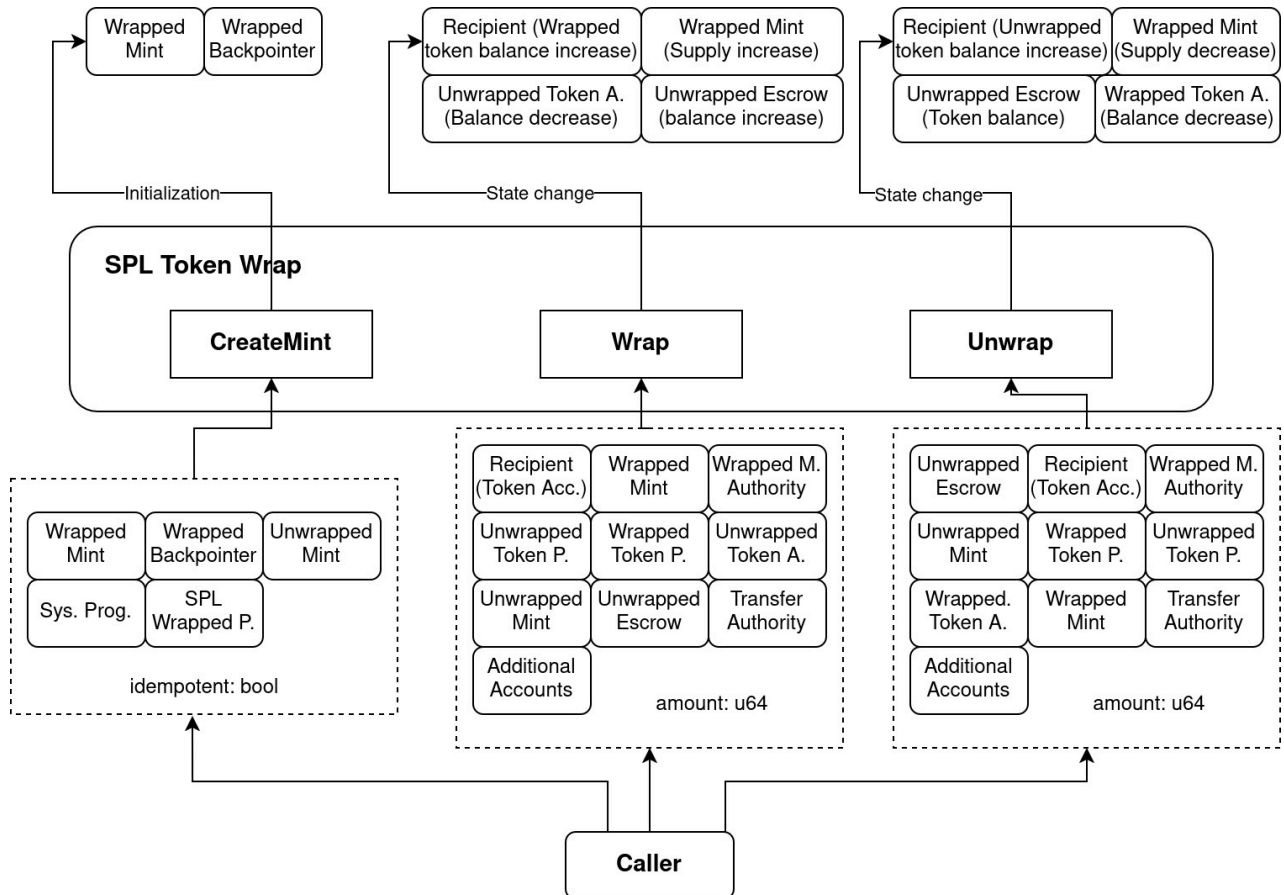


Figure 2: Token Wrap's interface and state change diagram.

The backpointer, which is initialized in the `createMint` instruction, stores the unwrapped token mint, creating a bidirectional mapping between the unwrapped and wrapped tokens.

The relationship between the accounts plays an integral part in the security of the Token Wrap program. Notice in Figure 2 that all token accounts and token program IDs are provided by the caller, including the addresses and IDs of the tokens that are being deposited as well as the tokens being minted/burnt. While it is necessary to specify to the

program which accounts it must interact with, it burdens the Token Wrap program with the responsibility to validate that the relationship between an unwrapped token and its wrapped counterpart is respected. This is necessary to prevent a malicious user from, for instance, requesting to wrap token A and attempting to receive a wrapped version of token B.

Figure 3, a diagram of ownership and address derivation, explores how the Token Wrap program validates the account relationship. The validations of these relationships, specifically 1, 2, and 3 (highlighted in the image), link wrapped tokens to unwrapped tokens within the protocol's logic. Validations 1, 2, and 3 are made explicit in the `wrap` procedure, and 1 and 2 also are in `unwrap`, while validation 3 is implicit to the transfer function, as the Wrapped Mint Authority is the transfer authority for the escrow.

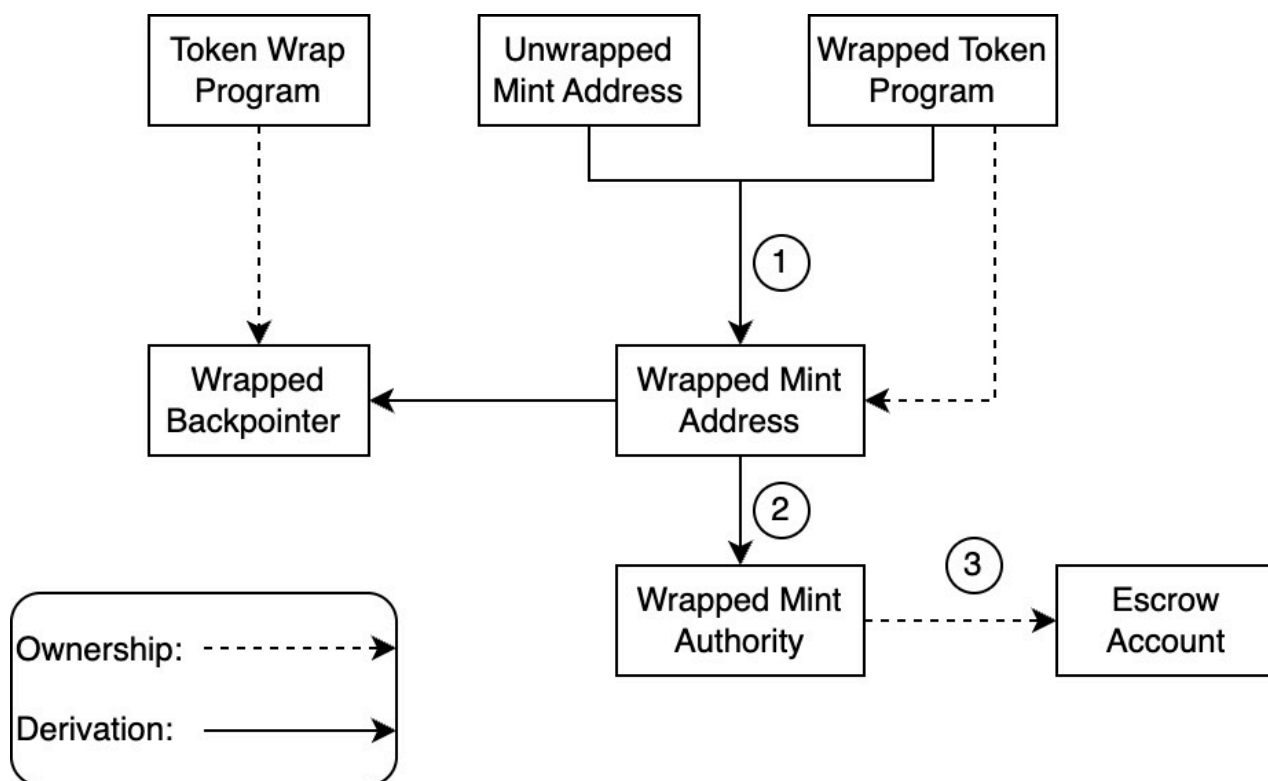
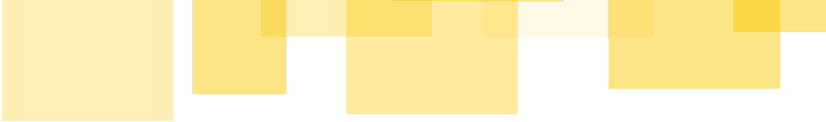


Figure 3: Token Wrap account ownership and derivation diagram.

The Token Wrap is a permissionless contract, meaning that any user(s)/program can create a mint for a wrapped token, wrap and unwrap tokens.



For an in-depth explanation of the instructions executed by the Token Wrap program, please refer to the [Token Wrap Formal Specification](#), as well as the abstractions for the programs interacting with the Token Wrap contract (namely the SPL Token and Token 2022 programs), created as a form of a [SPL Token and SPL Token 2022 High-level Specifications](#).

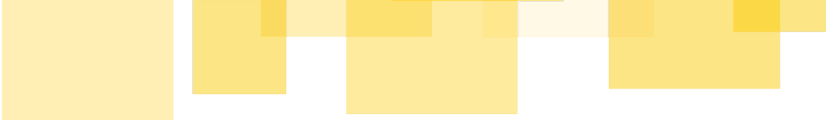


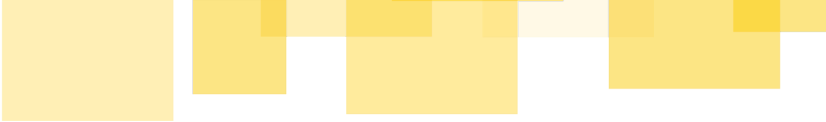
Token Wrap Formal Specification

This section defines a formal specification in plain English for the Token Wrap program. It is used to build an abstraction for both understanding and validating properties of the program, as well as deriving new properties to be analyzed and validated. Below are some terms used throughout the document:

Definitions

- **MUST** *behaviour* - the *behaviour* is required;
- **MUST NOT** *behaviour* - the *behaviour* is prohibited;
- **IF** *cond* **THEN** *effect* - The condition *cond* being true means the effect *effect* **MUST** hold. The condition *cond* being false does not necessarily mean that the effect *effect* does not hold;
- **Address** - the unique key to an **Account** ([Solana docs: Terminology](#)).
 - Exactly one of the following:
 - **IF** **Account** is a **Non-executable Account** (wallet) **THEN** 32-byte valid ed25519 public key;
 - **IF** **Account** is an Executable Account **THEN** 32-byte value that **MUST NOT** be a valid ed25519 public key (see **PDA** below for details);
- **Program Derived Address (PDA)** - A unique key to an **Account** belonging to an **Executable Account** that is derived from the **Address** of that **Executable Account**, optionally some seed data as Bytes, and a **Bump** number as a u8. When we mention that a PDA is primarily derived from a value, we indicate that PDA uses a unique, usually user-provided value that differentiates the data seed used for its derivation. The remaining data provided for the PDA derivation is pre-deterministic or can be obtained/calculated before a function call, therefore not mentioned unless it is of interest;
- **Account** - a Solana network account unique to an **Address** ([Solana docs: Account](#), [Solana docs.rs: Account](#)).
 - Exactly one of:

- 
- **Executable Account**, with SBF bytecode that would be executed in the data field;
 - **Non-executable Account**, with state data or no data in the data field;
 - **Program Derived Address Account (PDA Account)** - an account with a unique address mathematically derived from a specific program's ID and a set of seeds, guaranteed to have no corresponding private key, allowing the deriving program to have secure, deterministic on-chain control. 1 ([Solana docs: PDA](#))
 - **MUST**: be derived from Program Account (**BPF Loader** is the owner), some other data
 - **Token Program** - An **Account** owned by a **BPF Loader** program that is either a SPL Token (Original) or Token 2022. Programs are identified by a unique identifier (program ID).
 - **Token Account** - An **Account** used to track individual ownership of each token unit. A Token Account stores the necessary data for token management for that user (owner, mint address, and balance) ([Solana docs: Token Account](#))
 - **Mint Account** - **Account** that stores core information of a token (Supply, Decimals, Mint Authority, Freeze Authority) ([Solana docs: Mint account](#));
 - **Authority Accounts** - Accounts that have been granted specific, pre-defined permissions or control over a token's lifecycle or behavior. These permissions may be related to transfer, freezing (prevention of transfers), closing (ending the lifecycle of an account), or any specific privilege related to an operation.
 - **Token Wrap contract/program** - a program that wraps a token implemented following the SPL Token or Token 2022 standard in a wrapper token following either SPL Token or Token 2022 (without extensions) standard;
 - **Unwrapped token** - A token implemented in either the SPL Token or Token 2022 (with or without extensions) standard that is deposited or withdrawn from a contract to mint or burn wrapped tokens, following the logic defined in the Token Wrap contract;
 - **Wrapped token** - A token implemented in the SPL Token or Token 2022 standard (with no extensions), minted or burned according to business logic defined within the Token Wrap contract;

- 
- **Associated Token Account (ATA)** - A form of PDA primarily derived from an account address, the Token Program identifier, and the Token Mint address. It is a programmatically, on-chain instantiated **Token Account**.

Token Wrap Formal Specification

Simple Summary

The Token Wrap program enables the creation of "wrapped" versions of existing SPL tokens, facilitating interoperability between different token standards, specifically between SPL Token and Token 2022. This specification defines the contract implemented by the program, detailing the assumptions and guarantees for each public interaction.

Program Derived Accounts

PDAs initialized and manipulated during the execution of the Token Wrap contract:

- **Wrapped Mint:** A **PDA** primarily derived from the unwrapped mint address and wrapped token program ID. It is used to manage wrapped tokens, whose minting operation is triggered when executing the wrap instruction, and the burning operation is triggered when executing the unwrap instruction.
- **Wrapped Mint Authority:** A **PDA** primarily derived from the wrapped mint address. It is used to manage wrapped tokens and the Wrapped Mint, where their minting operation is the wrap instruction.
- **Backpointer Account:** A **PDA** primarily derived from the wrapped mint address. Being derived from the wrapped mint address, it stores data related to the unwrapped token and creates a helpful link for mapping wrapped tokens back to their unwrapped counterparts.

Other addresses of importance

- **Unwrapped Token Program:** The source account for tokens to be wrapped
 - **MUST** be exactly one of:
 - A **SPL Token (Original) Program**;
 - A **Token 2022 Program**, with or without extensions.

- 
- IF the **Unwrapped Token Program** follows the **Token 2022** standard, **THEN** the resulting **Wrapped Token MUST NOT** support any **Extensions**;
 - **Wrapped Token Account**: The source or destination account for wrapped tokens to be minted (wrapping) to or burned from (unwrapping), depending on the operation being executed by the Token Wrap contract. May or may not be an ATA;
 - **Unwrapped Token Account**: The source or destination account for unwrapped tokens to be transferred to (unwrapping) and from (wrapping), depending on the operation being executed by the Token Wrap contract. May or may not be an ATA;
 - **Escrow Account**: Holds the unwrapped tokens while they are wrapped. Must be owned by the wrapped mint authority PDA;
 - **Transfer Authority**: An authority to transfer unwrapped tokens (for wrapping) or burn wrapped tokens (for unwrapping);
 - **Multisig Signers**: Optional additional signers for multisig-controlled token accounts;
 - **Wrapped Token Program**: The token program for the wrapped tokens (**SPL Token** or **Token 2022** without extensions).

Specification (Endpoints/Instructions)

1. CreateMint

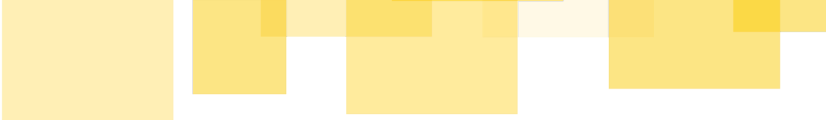
Creates a wrapped token mint and its associated backpointer account, enabling the wrapping and unwrapping of the tokens through the use of the newly created mint.

Processed by a function with the following signature:

```
pub fn process_create_mint( program_id: &Pubkey, accounts: &[AccountInfo], idempotent: bool,) -> ProgramResult
```

Where `program_id` reflects the unique program number of the deployed Token Wrap program, `idempotent` reflects whether the operation will or will not fail, depending on whether the mint has already been initialized or not. The second parameter, `accounts`, represents the set of accounts with which the Token Wrap contract interacts in this operation. These accounts are, in order:

1. Unallocated wrapped mint account, which will be initialized as the wrapped token mint;

- 
2. Unallocated wrapped backpointer, which will store data on the unwrapped token mint;
 3. Unwrapped mint account, containing information about the token that will be wrapped;
 4. System program account;
 5. Account of the token program to which the wrapped token will adhere.

- **Validated Assumptions:**

- The wrapped mint and backpointer accounts **MUST** be prefunded with sufficient lamports;
- The wrapped mint address **MUST** be a valid PDA primarily derived from the unwrapped mint address and wrapped token program ID;
- The backpointer account address **MUST** be a valid PDA primarily derived from the wrapped mint address.

- **Guarantees:**

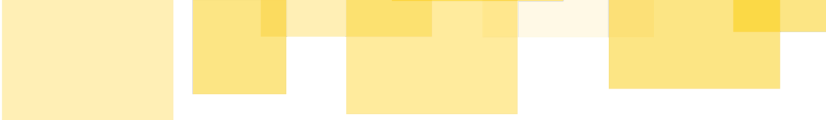
- A new wrapped mint **MUST** be initialized with the wrapped mint authority as its mint authority;
- A backpointer account **MUST** be initialized, storing the address of the unwrapped mint;
- **IF** `idempotent` is true and the mint already exists, **THEN** the operation is a no-op;
- **IF** `idempotent` is false and the mint already exists, **THEN** the operation fails.

2. Wrap

Transfers unwrapped tokens to an escrow account and mints an equivalent amount of wrapped tokens. Processed by a function with the following signature:

```
pub fn process_wrap(accounts: &[AccountInfo], amount: u64) -> ProgramResult
```

Where `amount` reflects the number of tokens that will be sent to a token account, enabling the minting of an equal number of wrapped tokens. The second parameter,



accounts , represents the set of accounts with which the Token Wrap contract interacts in this operation. These accounts are, in order:

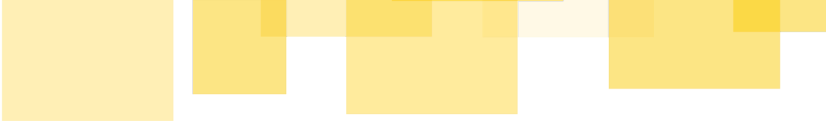
1. Token account that will receive the newly minted wrapped tokens, also referred to as the “recipient” account;
2. Account of the wrapped token mint;
3. Account of the mint authority of the wrapped token;
4. Account of the unwrapped token program;
5. Account of the wrapped token program;
6. Token account from which the unwrapped tokens will be transferred, also referred to as “source” account;
7. Account of the unwrapped mint;
8. Token account, which will hold the unwrapped tokens, also referred to as “escrow” account;
9. Account of the authority for the transfer of the unwrapped token;
10. If using a multisig as the authority for the transfers, the required accounts that configure this multisig should be listed here.

- **Validated assumptions:**

- The wrapped mint **MUST** have been initialized via the CreateMint instruction in this instance of the Token Wrap contract;
- The wrapped mint address **MUST** be a valid PDA primarily derived from the unwrapped mint address and wrapped token program ID;
- The wrapped mint authority **MUST** be a valid PDA primarily derived from the wrapped mint address;
- The source account **MUST** have sufficient tokens to wrap;
- The transfer authority **MUST** have permission to transfer tokens from the unwrapped token account;
- The wrapped mint authority **MUST** own the escrow account.

- **Guarantees:**

- The specified amount of unwrapped tokens **MUST** be transferred from the source account to the escrow account;

- 
- An equivalent amount of wrapped tokens **MUST** be minted to the recipient's wrapped token account.

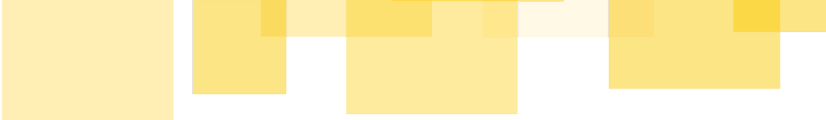
3. Unwrap

Burns wrapped tokens and transfers an equivalent amount of unwrapped tokens from escrow. Processed by a function with the following signature:

```
process_unwrap(accounts: &[amp]AccountInfo, amount: u64) -> ProgramResult
```

Where `amount` reflects the number of unwrapped tokens that will be sent to a recipient token account, enabling the burning of an equal number of wrapped tokens. The second parameter, `accounts`, represents the set of accounts with which the Token Wrap contract interacts in this operation. These accounts are, in order:

1. Token account, which will hold the unwrapped tokens, also referred to as “escrow” account;
 2. Token account to which the unwrapped tokens will be sent, also referred to as “recipient” account;
 3. Account of the mint authority of the wrapped token;
 4. Account of the unwrapped mint;
 5. Account of the wrapped token program;
 6. Account of the unwrapped token program;
 7. Token account from which the wrapped tokens will be transferred, also referred to as “source” account;
 8. Account of the wrapped token mint;
 9. Account of the authority for the wrapped token burning operation;
 10. If using a multisig as the authority for the transfers, the required accounts that configure this multisig should be listed here.
- **Validated assumptions:**
 - The source account **MUST** have sufficient tokens to be unwrapped, i.e., to be burnt;

- 
- The wrapped mint address **MUST** be a valid PDA primarily derived from the unwrapped mint address and wrapped token program ID;
 - The wrapped mint authority **MUST** be a valid PDA derived from the wrapped mint address;
 - The escrow account **MUST** have sufficient unwrapped tokens;
 - The wrapped mint authority **MUST** own the escrow account;
 - The transfer authority **MUST** have permission to burn tokens from the wrapped token account.
 - **Guarantees:**
 - The specified amount of wrapped tokens **MUST** be burned from the user's source account;
 - An equivalent amount of unwrapped tokens **MUST** be transferred from the escrow account to the recipient account;



SPL Token and SPL Token 2022 High-level Specifications

SPL Token Specification

Simple Summary

The Solana Program Library (SPL) Token project implements a fungible token standard on the Solana blockchain. It provides functionality for creating and managing token mints, token accounts, and multisignature authorities. It works similarly to the ERC-20 standard on Ethereum, being adapted to Solana's particularities involving its storage and program models.

Program Derived Accounts

PDAs that can be used during the execution of the SPL Token contract:

- **Associated Token Account:** A PDA primarily derived from the wrapped mint address, the caller wallet address, and the token program address. It is used to manage the token information in relation to a specific user. This account is not a part of the SPL Token program and is handled by a separate program.

Other addresses of importance

- **Mint Account:** This account stores general information about the token (authorities, supply, decimals) and is owned by the token program.
- **Mint Authority:** The authority that can mint new tokens for a specific mint. The authority can or cannot be a multisig account;
- **Freeze Authority:** The authority that can freeze token accounts for a specific mint;
- **Token Account:** Serves the same purpose as an ATA;
- **Delegate:** An address authorized by the owner to transfer tokens on their behalf up to a specified amount;
- **Close Authority:** The authority that can close a token account.



High-level specification (Endpoint/Instructions)

1. InitializeMint

Initializes a new token mint with specified decimals, mint authority, and optional freeze authority.

- Assumptions:
 - The mint account has been created but not initialized;
 - The mint account has sufficient lamports for rent exemption.
- Guarantees:
 - The mint is initialized with the specified parameters;
 - Only the freeze authority can freeze accounts (if specified);
 - The initial supply is zero.

2. InitializeAccount

Initializes a new account to hold tokens.

- Assumptions:
 - The account has been created but not initialized;
 - The account has sufficient lamports for rent exemption;
 - The mint exists and is initialized.
- Guarantees:
 - The account is initialized and linked to the specified mint, and its fields are set to default values according to their types (0, None, and Initialized);
 - The initial token balance is zero (or equal to lamports if native SOL).

3. InitializeMultisig

Initializes a multisignature account with N signers and M required signatures.

- Assumptions:
 - The account has been created but not initialized;
 - The account has sufficient lamports for rent exemption;

- $1 \leq m \leq n \leq 11$ (where n is the number of signers).
- Guarantees:
 - The multisignature account is initialized with the specified parameters.

4. **Transfer**

Transfers tokens from one account to another.

- Assumptions:
 - Both source and destination accounts are initialized;
 - Both source and destination accounts belong to the same mint;
 - The source account has sufficient tokens;
 - Neither the source nor the destination accounts are frozen;
 - The authority is either the owner or a delegate of the source account.
- Guarantees:
 - The specified amount is deducted from the source account;
 - The specified amount is added to the destination account;
 - If the source account has a delegate, the delegated amount is reduced accordingly. The delegate is cleared if the delegated amount is reduced to 0;
 - If the source account is for native SOL, the corresponding lamports are transferred.

5. **Approve**

Approves a delegate to transfer up to a specified amount of tokens.

- Assumptions:
 - The source account is initialized;
 - The source account is not frozen;
 - The authority is the owner of the source account.
- Guarantees:
 - The delegate is authorized to transfer up to the specified amount from the source account;

- 
- Any previous delegation is overwritten.

6. **Revoke**

Revokes a delegate's authority.

- Assumptions:
 - The source account is initialized;
 - The authority is the owner of the source account;
- Guarantees:
 - The delegate's authority is revoked;
 - The delegated amount is set to zero.

7. **SetAuthority**

Changes the authority of a mint or account.

- Assumptions:
 - The provided parameter account (mint or token account) is initialized;
 - The signer is the current authority for the specified authority type.
- Guarantees:
 - The authority is changed to the new authority;
 - If the new authority is None, the authority is permanently removed.

8. **MintTo**

Mints new tokens to an account.

- Assumptions:
 - The mint is initialized and has a mint authority;
 - The destination account is initialized and associated with the mint;
 - The destination account is not frozen;
 - The signer is the mint authority;
- Guarantees:

- The specified amount of tokens is added to the account;
- The specified amount increases the mint's supply;
- The total supply cannot overflow a u64.

9. **Burn**

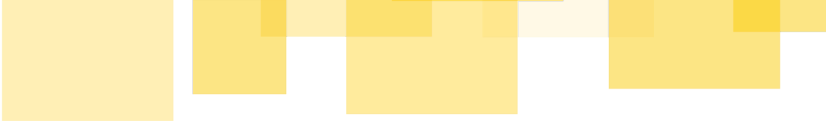
Burns tokens by removing them from an account.

- Assumptions:
 - The informed account, in which the tokens are going to be burned, is initialized and not frozen;
 - The informed account, in which the tokens are going to be burned, has sufficient tokens to burn;
 - The informed account, in which the tokens are going to be burned, is not associated with the native mint
 - The informed account, in which the tokens are going to be burned, is associated with the specified mint;
 - The signer is either the owner or a delegate of the informed account;
- Guarantees:
 - The specified amount of tokens is removed from the informed account;
 - The mint's supply is decreased by the specified amount;
 - If the account has a delegate, the delegated amount is reduced accordingly.

10. **CloseAccount**

Closes a token account, defined as the source, and recovers all its SOL to an informed destination account.

- Assumptions:
 - The source and destination accounts are different accounts;
 - The account is initialized;
 - The account has zero token balance (if not native);
 - If a close authority is set, the signer must be the close authority of the account;
 - If a close authority is not set, the signer must be the owner of the account.

- 
- Guarantees:
 - The account's lamports are transferred to the destination account;
 - The account's data is zeroed out.

11. **FreezeAccount**

Freezes an account, preventing token manipulation.

- Assumptions:
 - The informed account is initialized and not already frozen;
 - The mint has a freeze authority;
 - The signer is the freeze authority of the mint;
 - The account is not associated with the native mint;
 - The informed account is associated with the specified mint.
- Guarantees:
 - The account's state is changed to Frozen.

12. **ThawAccount**

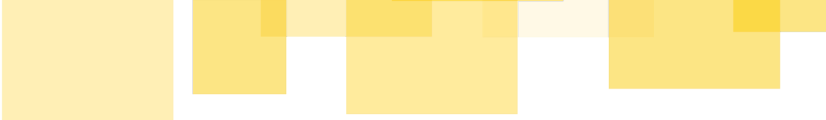
Thaws (unfreezes) a frozen account, allowing it to manipulate its tokens once again.

- Assumptions:
 - The account must be in a frozen state;
 - The mint has a freeze authority;
 - The signer is the freeze authority of the mint;
 - The informed account is associated with the specified mint.
- Guarantees:
 - The account's state is changed to Initialized;
 - Tokens can be transferred to and from the account.

13. **TransferChecked**

Transfer with additional mint and decimals validation.

- Assumptions:

- 
- Same as Transfer;
 - The mint's decimals match the expected decimals.
 - Guarantees:
 - Same as Transfer;
 - Failure if there's a divergence between the token decimals and informed decimals.

14. **ApproveChecked**

Approve with additional mint and decimals validation.

- Assumptions:
 - Same as Approve;
 - The mint's decimals match the expected decimals.
- Guarantees:
 - Same as Approve;
 - Failure if there's a divergence between the token decimals and informed decimals.

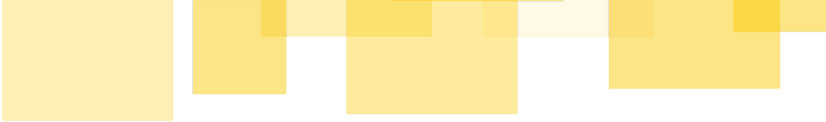
15. **MintToChecked**

MintTo with additional decimals validation.

- Assumptions:
 - Same as MintTo;
 - The mint's decimals match the expected decimals.
- Guarantees:
 - Same as MintTo;
 - Failure if there's a divergence between the token decimals and informed decimals.

16. **BurnChecked**

Burn with additional decimals validation.

- 
- Assumptions:
 - Same as Burn;
 - The mint's decimals match the expected decimals.
 - Guarantees:
 - Same as Burn;
 - Failure if there's a divergence between the token decimals and the informed decimals;

17. **InitializeAccount2**

Initialize the account with the owner provided in the instruction data instead of the account list.

- Assumptions
 - Same as InitializeAccount.
- Guarantees
 - Same as InitializeAccount.

18. **InitializeAccount3**

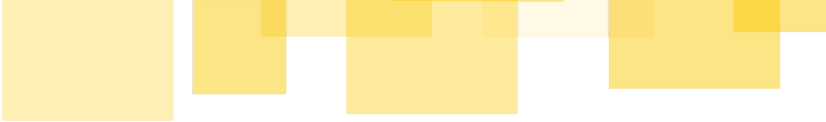
Same as InitializeAccount2, but doesn't require the Rent sysvar, instead using the rent information from a provided account.

- Assumptions
 - Same as InitializeAccount2.
- Guarantees
 - Same as InitializeAccount2.

19. **InitializeMultisig2**

Initialize multisig without requiring the Rent sysvar, instead using the rent information from a provided account.

- Assumptions

- 
- Same as InitializeMultisig.
 - Guarantees
 - Same as InitializeMultisig.

20. **InitializeMint2**

Initialize mint without requiring the Rent sysvar, instead using the rent information from a provided account.

- Assumptions
 - Same as InitializeMint.
- Guarantees
 - Same as InitializeMint.

21. **SyncNative**

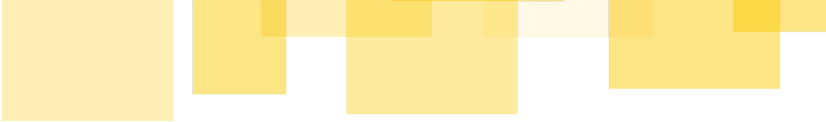
Syncs a native SOL-wrapped token account with its SOL balance.

- Assumptions:
 - The account is initialized and associated with the native mint.
- Guarantees:
 - The token balance is updated to match the lamport balance minus the rent-exempt reserve.

22. **GetAccountDataSize**

Returns the required account data size for a token account.

- Assumptions
 - The mint account must be an initialized mint owned by this contract.
- Guarantees
 - Returns the required size for a token account associated with the given mint as a little-endian u64. Given the SPL Token fixed size, the return value is always 165.

- 
- Return data can be fetched using `sol_get_return_data` and deserializing the return data as a little-endian u64.

23. **InitializeImmutableOwner**

Initializes an account with an immutable owner.

- Assumptions
 - The first account provided in the account list is initialized.
- Guarantees
 - None.

24. **AmountToUiAmount**

Converts raw token amount to human-readable format.

- Assumptions
 - The mint account must be an initialized mint owned by this contract.
- Guarantees
 - Returns a string representation of the token amount, formatted according to the mint's decimal places.

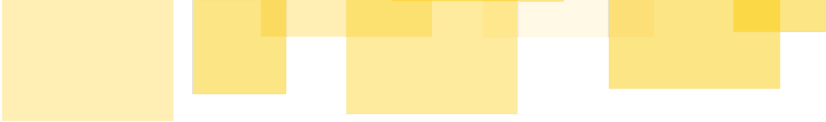
25. **UiAmountToAmount**

Converts a human-readable amount to a raw token amount.

- Assumptions
 - The mint account must be an initialized mint owned by this contract;
 - The `ui_amount` string must be a valid decimal number.
- Guarantees
 - Returns the raw token amount as a little-endian u64.

SPL Token 2022

Simple Summary



The SPL Token-2022 standard, also referred to in this document simply as Token 2022, is an evolution and extension of the original SPL Token Program on the Solana blockchain. Its primary purpose is to provide a more flexible, feature-rich, and future-proof token standard by allowing for the addition of extensions to tokens. While sharing many features of the SPL Token Program, aside from the fact that these are two different contracts with different program IDs, the added functionalities imply different implementations of methods of the same purpose, as well as added instructions/endpoints of interactions for the Token 2022 program.

Addresses of importance

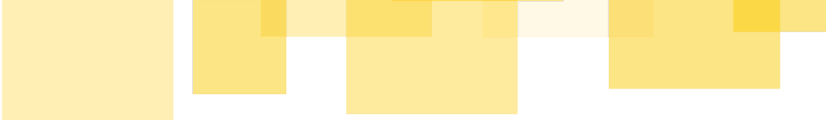
Both PDA and Other Addresses Of Importance mentioned in the SPL Token definition, with addition of Mint Close Authority, which is the account capable of closing the a token's mint account and retrieving its associated SOL.

High-level specification (Endpoint/Instructions)

For the sake of the abstraction, being outside of scope, the SPL Token 2022 standard, we assume that the following instructions share the same set of Assumptions and Guarantees as their counterparts in the SPL Token standard: `InitializeMint` , `InitializeAccount` , `InitializeMultisig` , `CloseAccount` , `Transfer` , `Approve` , `Revoke` , `MintTo` , `Burn` , `FreezeAccount` , `ThawAccount` , `TransferChecked` , `ApproveChecked` , `MintToChecked` , `BurnChecked` , `SetAuthority` , `SyncNative` , `InitializeAccount2` , `InitializeAccount3` , `InitializeMultisig2` , `InitializeMint2` , `InitializeImmutableOwner` , `AmountToUiAmount` , `UiAmountToAmount` .

`GetAccountDataSize` , although present in both standards, has diverging implementation. In the SPL Token standard, it returns a 64 bit unsigned integer of an account for the given mint. For the Token 2022 standard, it receives a parametrized vector which indicates the extension types to be included into the account which the size is being calculated.

Furthermore, a few of the instructions available in the Token 2022 are instruction prefixes, where the extension-specific instructions provide the detailed functionality. Since in this abstraction we do not consider the extensions themselves, we do not delve into the



following instructions: `TransferFeeExtension` , `ConfidentialTransferExtension` , `DefaultAccountStateExtension` , `MemoTransferExtension` , `InterestBearingMintExtension` , `CpiGuardExtension` , `TransferHookExtension` , `ConfidentialTransferFeeExtension` , `MetadataPointerExtension` , `GroupPointerExtension` , `GroupMemberPointerExtension` , `ConfidentialMintBurnExtension` , `ScaledUiAmountExtension` , `PausableExtension` .

Even though these have not been thoroughly analyzed, a suitable research was made in order to ensure the compatibility of the Token 2022 standard extensions with the Token Wrap program, culminating in [\[A2\] Token 2022 Extensions May Cause Issues When Wrapped](#).

Here are the remaining instructions available by default in the Token 2022 standard:

1. **InitializeMintCloseAuthority**

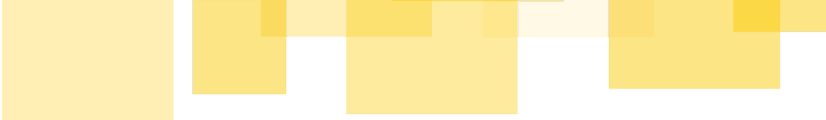
Initializes the close authority on a new mint, which can later close the mint account.

- Assumptions:
 - The mint account has not been initialized yet (must be called before `InitializeMint`);
 - The mint account has sufficient space allocated for the base mint (82 bytes), padding (83 bytes), account type (1 byte), and the extension data.
- Guarantees:
 - The mint will have a designated close authority (or none if null is provided);
 - Only the designated close authority will be able to close the mint account.

2. **Reallocate**

Reallocates an existing token account or mint to add extension support based on its data size.

- Assumptions:
 - The informed account exists and is initialized;
 - The owner of the informed account has signed the transaction;
 - The account has sufficient lamports to cover rent for the new size.

- 
- Guarantees:
 - The account will be resized to accommodate the requested extensions;
 - The account's data will be preserved;
 - Additional lamports will be transferred if needed for rent.

3. **CreateNativeMint**

Creates the native mint (Wrapped SOL).

- Assumptions:
 - This instruction only needs to be invoked once after deployment;
 - The funding account is a system account and has signed the transaction.
- Guarantees:
 - The native mint will be created at the predefined address.

4. **InitializeNonTransferableMint**

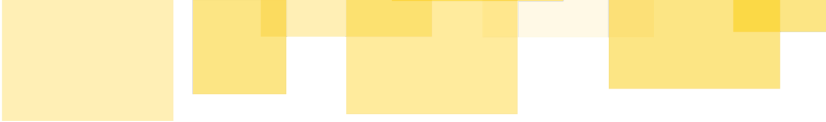
Initializes the non-transferable extension for a mint account.

- Assumptions:
 - The mint account has not been initialized yet (must be called before `InitializeMint`);
 - The mint account has sufficient space allocated for the extension.
- Guarantees:
 - Tokens from this mint cannot be transferred between accounts;
 - Accounts holding these tokens will automatically have the `ImmutableOwner` extension.

5. **InitializePermanentDelegate**

Initializes the permanent delegate on a new mint.

- Assumptions:
 - The mint account has not been initialized yet (must be called before `InitializeMint`);

- 
- The mint account has sufficient space allocated for the base mint (82 bytes), padding (83 bytes), account type (1 byte), and the extension data.
 - Guarantees:
 - The mint will have a designated permanent delegate;
 - The permanent delegate can sign for Transfer and Burn operations on any account holding tokens from this mint.

6. **WithdrawExcessLamports**

Withdraws excess lamports from a token account, multisig, or mint.

- Assumptions:
 - The account has more lamports than required for rent exemption;
 - The owner of the account has signed the transaction.
- Guarantees:
 - Excess lamports will be transferred to the destination account;
 - The account will remain rent-exempt.



Invariants

During the audit, invariants were defined and used to guide part of our search for possible issues with Anza's Token Wrap program. Using the previously explored specifications, the client's documentation, the intended business logic, and references collected during the audit, we identified the following invariants:

- Wrapped Mints must have their addresses primarily derived from an Unwrapped Mint Address and a corresponding Wrapped Token Program;
- The Wrapped Mint Authority is primarily derived from a Wrapped Mint Address;
- Any account that holds unwrapped tokens for the Token Wrap contract (referred to as escrow accounts) must be under the Wrapped Mint Authority's ownership;
- Wrapped tokens can only be minted if unwrapped tokens are simultaneously transferred to an escrow account;
- Wrapped tokens can only be burnt if unwrapped tokens are simultaneously transferred from an escrow account;
- Outside of an unwrapping operation, the balance of an escrow cannot be reduced;
- There must be a one-to-one relationship between the supply of wrapped tokens and wrapped tokens held in custody of the escrows owned by the Wrapped Mint Authority;

We highlight that these invariants were raised assuming that Solana's native features used by the protocol are sound regarding safety and security (e.g., no PDA derivation collision).

Furthermore, although these invariants guide our analysis and research, any other behavior diverging from the expected protocol business logic was noted and brought to the client's attention.



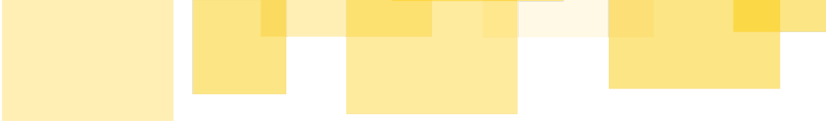
Test Harness Modifications and Results

Although outside the scope of this engagement, we went through the testing harnesses and structure available in the Token Wrap project repository in an attempt to create proof of concepts and explore possible corner cases involving the wrapping and unwrapping procedures.

This project currently relies on [Mollusk](#) as its primary testing harness. While Mollusk is a great choice for its lightweight nature and speed, enabling quick tests without requiring a full Solana node, it presents significant limitations for comprehensive testing, the major one being the lack of statefulness between test runs, which complicates complex, multi-instruction scenarios. Although it is possible to construct, the testing structure of the Token Wrap project did not contribute to our attempts to build such tests.

A major architectural challenge in the current testing setup is the deeply embedded configuration and hard-coded execution logic. For instance, [Token 2022 extension settings are fixed within common functions](#), preventing easy modification for diverse test cases, and that would be significantly better to the developers if able to set such configurations at [the top level when calling a test with a Token2022](#). This design restricts test methods from being reusable, as they can't simply accept different configurations as parameters. Similarly, the [hard-coded execution flow](#), often limited to [a single instruction](#), makes testing the same token logic with varying sets of instructions or expected outcomes difficult. The existing framework also lacks capabilities for testing with a local or dev-net client, performing stateful testing with multiple instructions, or conducting parameterized testing, and it offers minimal coverage for Token 2022 extensions beyond a basic implementation for TransferHook.

To address these limitations, the team provided [a PR to the client repo](#) with improvement suggestions on the project's tests. The goal is to evolve the testing framework to support Mollusk-based testing with a more flexible interface. This new interface will allow for setting parameters, defining instruction sequences, and specifying expected results from a high level, enabling parameterized testing. Initial changes include enhanced helper functions for initializing Mint Accounts with all Token 2022 extensions (with stubs for



individual extension helpers), and a modification to the `TokenProgram::SplToken2022` enum variant to couple tokens directly with their enabled extensions. These changes aim to allow tests to declare Token 2022 extensions at a high level, streamlining test setup and promoting reusability.



Findings

Findings presented in this section are issues that can cause the system to fail, malfunction, and/or be exploited, and should be properly addressed.

All findings have a severity level and an execution difficulty level, ranging from low to high, as well as categories in which it fits. For more information about the classifications of the findings, refer to our [Smart Contract Analysis](#) page (adaptations performed where applicable).



[A1] Unwrapped Tokens Can Be Held Multiple, Distinct Escrows

Severity: Low

Difficulty: Medium

Recommended Action: Fix Design

Addressed by client

Description

For the wrapping and unwrapping operations, an escrow address is provided to aid in manipulating unwrapped tokens according to the operation being performed. Unwrapped tokens are transferred to these escrows when wrapping, and moved from them when unwrapping.

There are no assertions in the logic of the smart contract preventing the usage of multiple escrows to hold the unwrapped tokens. This causes a dilution of the unwrapped tokens, while there are no means to prevent the wrapped tokens from being concentrated into a single entity, potentially causing liquidity issues for unwrapping operations.

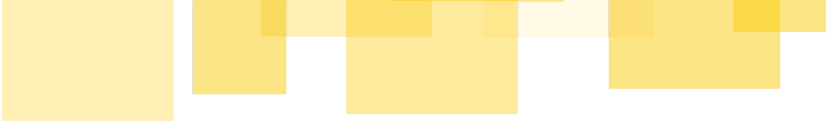
For example, see the scenario below:

Scenario

This scenario exemplifies a complication raised from the distribution of unwrapped assets into different escrows.

1. Alice wraps 100 token **T**, providing escrow **E1** as the holder for her unwrapped tokens. Alice receives 100 **wT**;
2. Bob wraps 100 token **T**, providing escrow **E2** as the holder for his unwrapped tokens. Bob receives 100 **wT**;
3. Alice transfers 100 **wT** to Bob in exchange for something of equal value;

Scenario issue: Bob, who is not tracking transactions involving the Token Wrap program, now has 200 **wT**, but only knowledge of the existence of one escrow that holds 100 **T**. Bob now has to have technical knowledge and the means to find **E2**, or must have a tool



that will perform these tasks for him. Until then, Bob has tokens of theoretical value, but they cannot be redeemed. And even obtaining the necessary information to unwrap all **wTs** , multiple transactions will be needed to unwrap all of Bob's tokens.

This distribution of assets complication can scale into more convoluted problems when including other tools and protocols in the scenario. For instance, a user could attempt to unwrap tokens and withdraw as many assets as possible from a publicly known escrow while his unwrapped tokens stay stored in another, newly created escrow in hopes of creating issues for keepers (arbitrageurs or liquidators, for instance) that may rely on the liquidity of wrapped tokens, and/or to bring themselves a unfair advantage in such protocols.

Recommendation

Hold the unwrapped tokens deposited in wrapping operations in a single token account.

Status

This finding has been addressed by the client by enforcing that a single escrow must be used for handling individual tokens. This escrow is a PDA primarily derived from the unwrapped mint address, the wrapped mint authority, and the unwrapped token program, creating relationship number 4 seen in Figure 4.

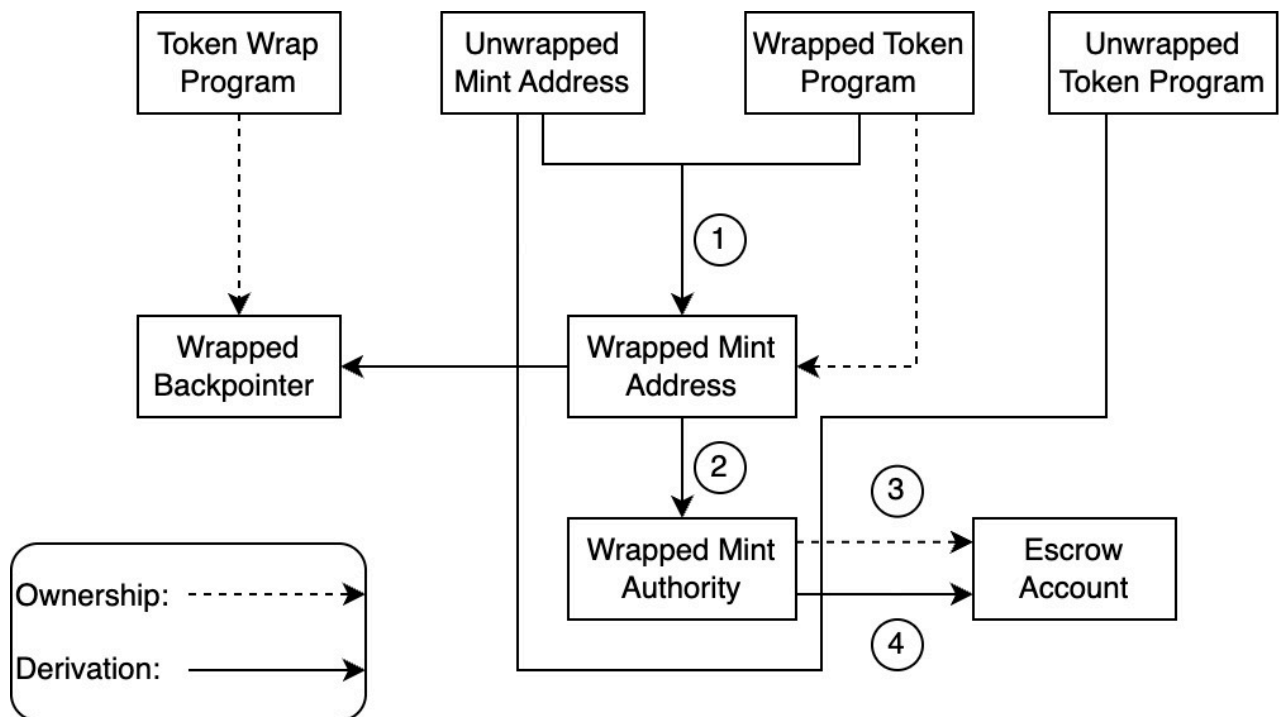


Figure 4: Updated Token Wrap account ownership and derivation diagram.

This has been implemented in [PR #108 in the client's repository](#).



[A2] Token 2022 Extensions May Cause Issues When Wrapped

Severity: Medium

Difficulty: Low

Recommended Action: Fix Design

Partially addressed by client

Description

As stated in the [Platform Features and Logic Description](#) and [Token Wrap Formal Specification](#) sections, tokens of the SPL Token and Token 2022 standards can be wrapped into either of these two standards.

A caveat here is that the wrapped token, if following the Token 2022 standard, will have no extensions by default. This means that if a token following the Token 2022 standard, with extensions, is wrapped using the Token Wrap program in its current, audited state, the resulting wrapped token will have none of the extensions present on its counterpart.

An abstraction of the extensions, built based on documentation and code analysis, was constructed to identify extensions that can be used together with the Token Wrap program without issues. The extensions were classified into three types:

- **Functional:** extensions that their removal will not cause overall issues to the Token Wrap program, its security invariants, or the business logic of the unwrapped tokens themselves. While they might have their functionalities stripped on the wrapped tokens, no loss is incurred from the wrapping and unwrapping process;
- **Context-Dependent:** consider extensions that may cause localized execution failures on the Token Program due to account configurations. Furthermore, they might be stripped of their core features, but the transfer behavior remains the same, with no loss incurred from wrapping and unwrapping the tokens if not prevented by the involved account states. Ultimately, it is up to the client's decision to allow or not these extensions;
- **Incompatible:** these extensions, when used, will likely cause failures on the Token Wrap program, disrespect the security invariants, or their essential purpose for usage



can be bypassed, potentially creating security/financial exploits.

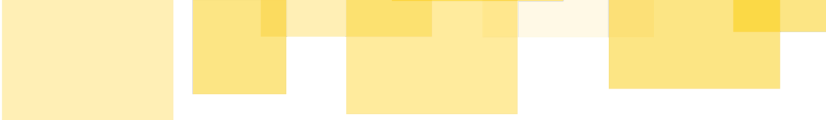
Below is the evaluation of the extensions grouped by their classification:

Incompatible

- **Transfer Fee:** Allows the token's issuer to configure a fee (either a fixed amount or a percentage of the transfer) that is automatically applied and withheld at the time of every token transfer. These fees are held on the recipient's account and can only be redeemed from the specific authority.
 - **Evaluation:** Considering the transfer fee extension, values would be withheld in the account that holds the unwrapped tokens, leading to a mismatched number of minted wrapped tokens to unwrapped tokens available for withdrawal, thus breaking the one-to-one minted-to-backing invariant. Furthermore, individuals could wrap tokens that have the transfer fee extension to bypass the fees when managing the tokens, invalidating their core logic.
- **Pausable:** Grants a designated authority the ability to temporarily pause all transfers, minting, or burning for a specific token.
 - **Evaluation:** While this extension does not directly affect any of the elaborated invariants for the security of the Token Wrap program, the wrapped tokens will not inherit the pausable property of their unwrapped counterparts. This may be used to circumvent the original purpose of pausing the unwrapped token operations, and may be used to exploit protocols and/or users that handle both wrapped and unwrapped tokens equally.

Context-Dependent

- **Confidential Transfers:** Enables transfers of tokens where the transaction amount is encrypted and remains hidden from the public ledger, using zero-knowledge proofs for verification.
 - **Evaluation:** When wrapping a token that uses the confidential transfer extension, the resulting wrapped token will not be capable of using this feature. While this limits the purpose of the wrapped token in contrast to its unwrapped



counterpart, it enables its interoperability within protocols that rely on the Token Wrap program. Furthermore, no invariant is damaged by the involvement of this extension in the wrapping and unwrapping procedure. The caveat becomes the configuration of the Confidential Transfer Token Account, which has a field defining if normal transfers, referred to as non-confidential transfers, can be performed (`allow_non_confidential_credits`). If this field is set to false, then transfers, as used in the TokenWrap contract, will fail.

- **Default Account State:** Allows the token's mint authority to define a default state (e.g., "uninitialized", "frozen", or "initialized") for all new Token Accounts created for that specific token.
 - **Evaluation:** The only way in which this exception influences or affects the Token Wrap program is where one of the specific token accounts involved in the wrapping or unwrapping processes is frozen in its default state. While one could argue that the owner of the frozen token account could still use wrapped tokens to bypass this "block", the user would not be able to mint wrapped tokens or burn the wrapped tokens to receive unwrapped tokens, given their frozen account state. Even so, the configuration of the accounts may lead to attempts to operate tokens in accounts that are not in a proper state to do so, causing the failure of execution of instructions.
- **Memo Transfer:** Requires a `spl-memo` instruction to be included in the same transaction as a token transfer.
 - **Evaluation:** The memo transfer extension enables user and/or protocols to decide whether their transactions should be accompanied by a memo instruction or not. Depending on the configuration of the token account issuing the transaction, attempts to transfer unwrapped tokens could lead to failure. Given that this is a behavior attributed to an account-level configuration, we neither recommend nor discourage enabling this extension in the tokens wrapped by the Token Wrap program.
- **Transfer Hook:** Allows the token's issuer to define a custom program as a "transfer hook." Every time a token with this extension is transferred, the Solana runtime

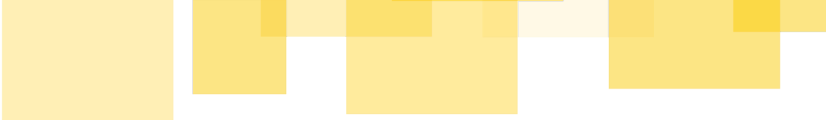


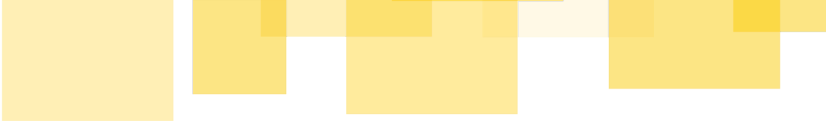
invokes this designated hook program.

- **Evaluation:** When the Token Extensions program CPIs to a Transfer Hook program, all accounts from the initial transfer are converted to read-only accounts. This means the signer privileges of the sender do not extend to the Transfer Hook program. During a transfer involving Token 2022 extensions and a Transfer Hook program, the CPI call makes all accounts from the initiating transfer read-only. Consequently, the sender's signing authority does not carry over to the transfer hook program. While this means that no malicious actions can be performed on the accounts involved in the wrap and unwrap operations, the transfer hook program may have logic that leads to transaction failure under conditions outside the Token Wrap program's control.

Functional

- **Interest-Bearing Mint:** Allows the token's value to accrue interest over time directly at the mint level, without requiring users to stake their tokens. The symbolic balance in a user's token account grows automatically.
 - **Evaluation:** When transferring tokens that use the Interest-Bearing feature, the nominal balance contained in the token accounts is modified. It is essential to highlight that the interest accrued isn't reflected in an increased supply of the interest-bearing token and/or a change of the nominal balances of token accounts, but rather in the calculation of the real token balance using data in the mint account. Therefore, having interest-bearing tokens wrapped and unwrapped does not cause harm to the owners of that token, and all invariants of the token wrap program are respected, considering the usage of this extension in its business logic.
- **CPI Guard:** CPI Guard is an extension that prohibits certain actions inside cross-program invocations, to protect users from implicitly signing for actions they can't see, hidden in programs that aren't the System or Token programs.
 - **Evaluation:** The protective measures imposed by the CPI Guard extension take no effect on the operations performed by the Token Wrap program concerning the wrapped and unwrapped tokens.

- 
- **Confidential Transfer Fee:** A specific extension related to Confidential Transfers that enables the application of transfer fees when the underlying transfer amount is confidential.
 - **Evaluation:** Confidential transfer fees are designed to be applied over confidential transfers, differentiating them from the Transfer Fee extension and preventing tokens from being withheld in the escrow when transferred using the transferChecked instruction. The Confidential Transfer Fee extension is used in conjunction with the Confidential Transfer extension, so even though our analysis does not identify a problem with this extension, the evaluation of the latter still must be carried out in order to include this extension in a whitelist.
 - **Metadata Pointer:** Stores a pointer (an account address) within the Mint Account's data that directs to an external account holding the token's rich metadata (e.g., name, symbol, URI, image).
 - **Evaluation:** This extension does not affect the operations performed by the Token Wrap program in relation to the wrapped and unwrapped tokens.
 - **Group Pointer:** Allows a specific token to point to a "Group" account (another token or a custom account); **Group Member Pointer:** (Complementary to Group Pointer) Allows a "Group" account to point to a "Group Member" account.
 - **Evaluation:** The Group Pointer and Group Member Pointer extensions enable token developers to group tokens together for a specific purpose. The functionalities derived from these extensions do not affect the operations performed by the Token Wrap program in relation to the wrapped and unwrapped tokens.
 - **Confidential Mint Burn:** Extends the confidential transfers framework to allow for the minting and burning of tokens in a confidential manner, where the amount minted or burned is not publicly revealed.
 - **Evaluation:** This extension does not influence the behavior of the wrapped token, and also does not influence normal transfers of the unwrapped token. All operations performed by the Token Wrap contract are executed successfully



while utilizing this extension, and while analyzed in isolation, this extension is not capable of affecting the invariants raised for the Token Wrap program analysis.

- **Scaled Ui Amount:** The scaled UI amount extension allows issuers to apply an updatable multiplier to the UI amount of a token. This is useful for scenarios where the token is used to represent a real-world asset, such as a stock or dividend.
 - **Evaluation:** Similar to the Interest-Bearing Mint extension, this extension has no effect on the nominal token balance of a token account. While the wrapped tokens cannot have their value calculated, as their mint has no information about the unwrapped token's scaling factor, the relationship of unwrapped to wrapped tokens is still one-to-one. This means that the wrapped token will still hold an equal symbolic value as the unwrapped one.

Recommendation

Filter the extension of the tokens and prevent tokens with incompatible extensions from being wrapped, or modify the program's logic to address the missing features on wrapped tokens.

Status

This finding has been partially addressed by considering fees during the wrapping process, enabling the compatibility of the Token Wrap contract with the Transfer Fee extension. This has been implemented in [PR #119 in the client's repository](#).

The considerations and evaluations of the remaining incompatible and context-dependent token extensions have been acknowledged by the client. Enabling these extensions has not been deemed threatening or damaging to the Token Wrap program or applications interacting with it.



Informative Findings

The findings presented in this section do not necessarily represent any flaw in the code itself. However, they indicate areas where the code may need external support or deviate from best practices. We have also included information on potential code size reductions and remarks on the operational perspective of the contract.



[B1] Best Practices Recommendations

Severity: Informative

Recommended Action: Fix Code

Partially addressed by client

Description

Here are some notes on the protocol's particularities, comments, and suggestions to improve the code or the business logic of the protocol in a best-practice sense. They do not present issues with the audited protocol themselves. Still, they are advised to either be aware of or to follow when possible, and they may explain minor unexpected behaviors in the deployed project.

1. In the `program/src/processor.rs` file of the Token Wrap project, within the `process_wrap` function in lines 198 to 204, a scope is explicitly added. This is done to prevent issues related to data borrowing. While no practical issues come from this, we advise maintaining a code standard, especially considering other Anza repositories analyzed through this engagement, and use `drop` to release the ownership of the mutable borrow instead;
2. In the `process_wrap` and `process_unwrap` functions, the last operations performed prior to the functions' return are invocations to mint operations or transfers, propagating their return after their calls. It is not necessary to have the `Ok(())` return afterwards, given that these invocations already return a `ProgramResult`.

Recommendation

For each of the topics elaborated above, we recommend implementing the following approaches into the protocol's contracts:

1. Maintain consistency regarding the code writing style. Use `drop` to prevent issues related to ownership and borrow instead of an explicit scope;
2. Remove the last lines of the `process_wrap` and `process_unwrap` functions (the returned `Ok(())`). These are the lines 243 and 316 of `program/src/processor.rs` in the frozen commit analyzed during this engagement.



Status

This finding has been partially addressed, with topic number one ceasing to exist after the modifications performed while addressing [\[A1\] Unwrapped Tokens Can Be Held Multiple, Distinct Escrows](#).



Final Considerations

It has been a privilege to conduct the code security review for Anza on the Token Wrap project. Our engagement involved a comprehensive analysis of the project's codebase with a focus on the core smart contract, and we are confident that this review has significantly strengthened the project's foundation. A critical component of our work included rigorously verifying the Token Wrap program's security against a defined set of invariants, with support of the developed formal and semi-formal specifications of the actors and programs involved, and we are pleased to confirm that no exploits were identified based on the project's established interaction interface.

Beyond the core security verification, this engagement has yielded several enhancements for the Token Wrap codebase. The test scaffolding has been refined to be substantially more flexible, which will facilitate future improvements and modifications to the project. Additionally, the interoperability of the underlying programs upon which the Token Wrap program is built has been notably enhanced. We anticipate these improvements will contribute significantly to the continued success and resilience of Solana's token interoperability systems.