CERTIK

Audit Report

Produced by CertiK

for THE SANDBOX

10th Dec, 2019

CertiK Audit Report For The Sandbox

Request Date: 2019-11-08
Revision Date: 2019-12-10
Platform Name: Ethereum

Contents

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and The Sandbox(the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the "Agreement"). This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, has developed a proprietary Formal Verification technology to apply rigorous and complete mathematical reasoning against code. This process ensures algorithms, protocols, and business functionalities are secured and working as intended across all platforms.

CertiK differs from traditional testing approaches by employing Formal Verification to mathematically prove blockchain ecosystem and smart contracts are hacker-resistant and bug-free. CertiK uses this industry-leading technology together with standardized test suites, static analysis, and expert manual review to create a full-stack solution for our partners across the blockchain world to secure 6.2B in assets.

For more information: https://certik.org/

Executive Summary

This report has been prepared for The Sandbox to discover issues and vulnerabilities in the source code of their Asset, AssetSignedAuction, BundleSandSale, CommonMinter, ProxyImplementation and TheSandbox712 smart contract. A comprehensive examination has been performed, utilizing CertiK's Formal Verification Platform, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Vulnerability Classification

CertiK categorizes issues into three buckets based on overall risk levels:

Critical

Code implementation does not match specification, which could result in the loss of funds for contract owner or users.

Medium

Code implementation does not match the specification under certain conditions, which could affect the security standard by loss of access control.

Low

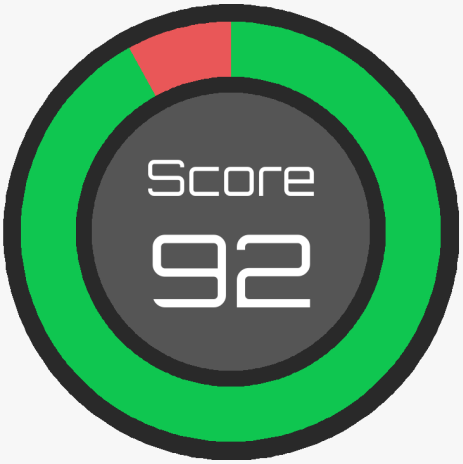
Code implementation does not follow best practices, or uses suboptimal design patterns, which could lead to security vulnerabilities further down the line.

Testing Summary

PASS

CERTIK *believes this smart contract passes security qualifications to be listed on digital asset exchanges.*

Dec 02, 2019



Type of Issues

CertiK's smart label engine applied 100% formal verification coverage on the source code. Our team of engineers has scanned the source code using proprietary static analysis tools and code-review methodologies. The following technical issues were found:

Title	Description	Issues	SWC ID
Integer Overflow and Underflow	An overflow/underflow occurs when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function Incorrectness	Function implementation does not meet specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker can write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by miners to some degree.	0	SWC-116
Insecure Compiler Version	Using a fixed outdated compiler version or floating pragma can be problematic if there are publicly disclosed bugs and issues that affect the current compiler version used.	0	SWC-102 SWC-103

Insecure Randomness	Using block attributes to generate random numbers is unreliable, as they can be influenced by miners to some degree.	0	SWC-120
"tx.origin" for Authorization	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115
Delegatecall to Untrusted Callee	Calling untrusted contracts is very dangerous, so the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default, meaning a malicious user can make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized Variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used.	0	SWC-111
Unused Variables	Unused variables reduce code quality	0	

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

Manual Review Notes

Source Code SHA-256 Checksum[1]

- **Asset.sol**
  `ecfd46cf80f9a57f6cb23c5869a4c53633354791aee73c96610f49c39912e2ec`
- **AssetSignedAuction.sol**
  `29ec420f2ec90c340c646f4117aeb13f40a5421f0ef2db1d6bccbbdc0410ba10`
- **BundleSandSale.sol**
  `0c34845117931230f420901cc02b47ef652d69a2abe64e806af42ab314beb3c8`
- **BytesUtil.sol**
  `2f5a0247cb5b9d905c3f6c64edaa67613d12ebe0e646ffe51ecb00e699144578`
- **CommonMinter.sol**
  `44c5a7e5e60f021344cb99306c7d2fe72d410c527bc444fa8c0f0029c0613808`
- **ProxyImplementation.sol**
  `34112f5c98ab184595655fa4c7b981d43919a2ec900d7eddf0fb435f369e5b09`
- **SafeMathWithRequire.sol**
  `f7b98afacff77193838a9fdfb4f22457b1734951ec5694249186fd42c5730ff1`
- **TheSandbox712.sol**
  `985860bec31c761446c328ff9a03c6bfa5b5acab4838c2f263495dc952188d2b`

Summary

CertiK was chosen by The Sandbox to audit the design and implementation of its soon to be released `LandSale` and related smart contracts. To ensure comprehensive protection,the source code has been analyzed by the proprietary CertiK formal verification engine and manually reviewed by our smart contract experts and engineers. That end-to-end process ensures proof of stability as well as a hands-on, engineering-focused process to close potential loopholes and recommend design changes in accordance with the best practices in the space.

Overall we found the smart contracts to follow good practices. With the final update of source code and delivery of the audit report, we conclude that the contract is structurally sound and not vulnerable to any classically known anti-patterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend to seek multiple opinions, keep improving the codebase, and more test coverage and sandbox deployments before the mainnet release.

Recommendations

Items in this section are labeled CRITICAL, MAJOR, MINOR, INFO, and DISCUSSION in decreasing significance level.

AssetSignedAuction.sol commit $a39eb2caa2c00e7865a4b808aec4c113b804b0c3$, previous

[1] Commit: $752e899abe7d5492227d28470a0bc2a0ae6dfd41$

1. MINOR The directory for `contracts_common/` is `../../../contracts_common/` in `AssetSignedAuction.sol` while it is `../../contracts_common/` in `BundelSandSale.sol`.
   - (The Sandbox - updated): Fixed in commit 76c29863800f85085e15b5070315b8aa85139de3.
2. MINOR `_verifyParameters()` and `_executeDeal()`: Recommend using `SafeMath` for arithmetic operations.
   - (The Sandbox - updated): Fixed in commit 76c29863800f85085e15b5070315b8aa85139de3.
3. INFO `_verifyParameters()`: Recommend marking this function with `view`.
   - (The Sandbox - updated): Fixed in commit 76c29863800f85085e15b5070315b8aa85139de3.
4. INFO `_verifyParameters()`: Recommend adding a `require()` to check `values.length == ids.length`.
   - (The Sandbox - updated): Fixed in commit 76c29863800f85085e15b5070315b8aa85139de3.

BundelSandSale.sol commit a39eb2caa2c00e7865a4b808aec4c113b804b0c3, previous

1. INFO `buyBundleWithEther()` and `buyBundleWithDai()`: Recommend updating `numPacksLeft` in `_transferPack()` instead of before calling `_transferPack()`.
   - (The Sandbox - confirmed): We will leave it there as we use the value in the require and do not want to read it again in the `_transferPack`.
2. INFO `onERC1155BatchReceived()`: Recommend adding a `require()` to check `values.length == ids.length`.
   - (The Sandbox - confirmed): We assume the ERC1155 behave correctly so we do not need to check `ids.length`.
3. INFO Mixed use of self defined checked arithmetic and `SafeMath`. Recommend using `SafeMath` throughout the contracts.

CommonMinter.sol commit a39eb2caa2c00e7865a4b808aec4c113b804b0c3, previous

1. INFO `mintFor()`: Converting `uint32 supply` to a `uint256` in calculations costs extra gas. Thinking of gas saving, should we set `supply` as a `uint256` from the beginning? For `uint40 packId`, if we don't do any calculations on it, the cost of gas doesn't differ a lot from a `uint256` according to our experiments. However, we still recommend using `uint256` for all `uint` type variables, if there's no special reason, according to Solidity Documentation.
   - (The Sandbox - confirmed): We prefer to have our abi as clear as possible and that is why `packId` is `uint40` and supply is `uint32`.
2. MINOR `mintMultipleFor()`: Recommend using `SafeMath` for `totalCopies += supplies[i]`.
   - (The Sandbox - updated): Fixed in commit 76c29863800f85085e15b5070315b8aa85139de3.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File SafeMathWithRequire.sol

```
pragma solidity ^0.5.2;
```

Only these compiler versions are safe to compile your code: 0.5.10

INSECURE_COMPILER_VERSION

Line 1 in File CommonMinter.sol

```
pragma solidity 0.5.9;
```

Version to compile has the following bug:
0.5.9: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement

INSECURE_COMPILER_VERSION

Line 1 in File Asset.sol

```
pragma solidity 0.5.9;
```

Version to compile has the following bug:
0.5.9: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement

INSECURE_COMPILER_VERSION

Line 1 in File BundleSandSale.sol

```
pragma solidity 0.5.9;
```

Version to compile has the following bug:
0.5.9: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement

INSECURE_COMPILER_VERSION

Line 1 in File TheSandbox712.sol

```
pragma solidity 0.5.9;
```

Version to compile has the following bug:
0.5.9: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement

INSECURE_COMPILER_VERSION

Line 1 in File AssetSignedAuction.sol

```
pragma solidity 0.5.9;
```

Version to compile has the following bug:
0.5.9: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement

TIMESTAMP_DEPENDENCY

Line 128 in File AssetSignedAuction.sol

```
        auctionData[AuctionData_StartedAt] <= block.timestamp,
```

"block.timestamp" can be influenced by miners to some degree

TIMESTAMP_DEPENDENCY

Line 132 in File AssetSignedAuction.sol

```
132         auctionData[AuctionData_StartedAt].add(auctionData[AuctionData_Duration]) > block
                .timestamp,
```

! ”block.timestamp” can be influenced by miners to some degree

Formal Verification Results

How to read

Detail for Request 1

transferFrom to same address

Verification date	20, Oct 2018
Verification timespan	395.38 ms

CERTIK *label location*	Line 30-34 in File howtoread.sol

CERTIK *label*

```
30  /*@CTK FAIL "transferFrom to same address"
31      @tag assume_completion
32      @pre from == to
33      @post __post.allowed[from][msg.sender] ==
34  */
```

Raw code location	Line 35-41 in File howtoread.sol

Raw code

```
35  function transferFrom(address from, address to
        ) {
36      balances[from] = balances[from].sub(tokens
37      allowed[from][msg.sender] = allowed[from][
38      balances[to] = balances[to].add(tokens);
39      emit Transfer(from, to, tokens);
40      return true;
41  }
```

Counterexample	This code violates the specification

Initial environment

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          from = 0x0
5          to = 0x0
6          tokens = 0x6c
7      }
8      This = 0
53             balance: 0x0
54         }
55     }
56
```

Post environment

```
57 After Execution:
58     Input = {
59         from = 0x0
60         to = 0x0
61         tokens = 0x6c
```

Formal Verification Request 1

If method completes, integer overflow would not happen.

10, Dec 2019
30.92 ms

Line 11 in File SafeMathWithRequire.sol

```
11    //@CTK NO_OVERFLOW
```

Line 18-29 in File SafeMathWithRequire.sol

```
18    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
19        // Gas optimization: this is cheaper than asserting 'a' not being zero, but the
20        // benefit is lost if 'b' is also tested.
21        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
22        if (a == 0) {
23            return 0;
24        }
25
26        uint256 c = a * b;
27        require(c / a == b, "overflow");
28        return c;
29    }
```

The code meets the specification.

Formal Verification Request 2

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
0.55 ms

Line 12 in File SafeMathWithRequire.sol

```
12    //@CTK NO_BUF_OVERFLOW
```

Line 18-29 in File SafeMathWithRequire.sol

```
18    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
19        // Gas optimization: this is cheaper than asserting 'a' not being zero, but the
20        // benefit is lost if 'b' is also tested.
21        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
22        if (a == 0) {
23            return 0;
24        }
25
26        uint256 c = a * b;
27        require(c / a == b, "overflow");
28        return c;
29    }
```

The code meets the specification.

Formal Verification Request 3

Method will not encounter an assertion failure.

10, Dec 2019
0.49 ms

Line 13 in File SafeMathWithRequire.sol

```
13    //@CTK NO_ASF
```

Line 18-29 in File SafeMathWithRequire.sol

```
18    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
19        // Gas optimization: this is cheaper than asserting 'a' not being zero, but the
20        // benefit is lost if 'b' is also tested.
21        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
22        if (a == 0) {
23            return 0;
24        }
25
26        uint256 c = a * b;
27        require(c / a == b, "overflow");
28        return c;
29    }
```

The code meets the specification.

Formal Verification Request 4

mul

10, Dec 2019
2.62 ms

Line 14-17 in File SafeMathWithRequire.sol

```
14    /*@CTK mul
15      @tag assume_completion
16      @post __return == a * b
17     */
```

Line 18-29 in File SafeMathWithRequire.sol

```
18    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
19        // Gas optimization: this is cheaper than asserting 'a' not being zero, but the
20        // benefit is lost if 'b' is also tested.
21        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
22        if (a == 0) {
23            return 0;
24        }
25
26        uint256 c = a * b;
27        require(c / a == b, "overflow");
28        return c;
29    }
```

The code meets the specification.

Formal Verification Request 5

If method completes, integer overflow would not happen.

10, Dec 2019
5.37 ms

Line 34 in File SafeMathWithRequire.sol

```
34    //@CTK NO_OVERFLOW
```

Line 41-46 in File SafeMathWithRequire.sol

```
41    function div(uint256 a, uint256 b) internal pure returns (uint256) {
42        // assert(b > 0); // Solidity automatically throws when dividing by 0
43        // uint256 c = a / b;
44        // assert(a == b * c + a % b); // There is no case in which this doesn't hold
45        return a / b;
46    }
```

The code meets the specification.

Formal Verification Request 6

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
0.27 ms

Line 35 in File SafeMathWithRequire.sol

```
35    //@CTK NO_BUF_OVERFLOW
```

Line 41-46 in File SafeMathWithRequire.sol

```
41    function div(uint256 a, uint256 b) internal pure returns (uint256) {
42        // assert(b > 0); // Solidity automatically throws when dividing by 0
43        // uint256 c = a / b;
44        // assert(a == b * c + a % b); // There is no case in which this doesn't hold
45        return a / b;
46    }
```

The code meets the specification.

Formal Verification Request 7

Method will not encounter an assertion failure.

10, Dec 2019
0.32 ms

Line 36 in File SafeMathWithRequire.sol

```
36    //@CTK FAIL NO_ASF
```

Line 41-46 in File SafeMathWithRequire.sol

```
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // assert(b > 0); // Solidity automatically throws when dividing by 0
        // uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold
        return a / b;
    }
```

This code violates the specification.

```
Counter Example:
Before Execution:
    Input = {
        a = 0
        b = 0
    }
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 0
        }
    }
    Other = {
        __return = 0
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": "ALL_OTHERS",
        "value": "EmptyAddress"
      }
    ]

Function invocation is reverted.
```

Formal Verification Request 8

div

10, Dec 2019
0.28 ms

Line 37-40 in File SafeMathWithRequire.sol

```
    /*@CTK div
      @tag assume_completion
      @post __return == a / b
     */
```

Line 41-46 in File SafeMathWithRequire.sol

```
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // assert(b > 0); // Solidity automatically throws when dividing by 0
        // uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold
        return a / b;
    }
```

The code meets the specification.

Formal Verification Request 9

If method completes, integer overflow would not happen.

10, Dec 2019
10.72 ms

Line 51 in File SafeMathWithRequire.sol

```
    //@CTK NO_OVERFLOW
```

Line 58-61 in File SafeMathWithRequire.sol

```
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "undeflow");
        return a - b;
    }
```

The code meets the specification.

Formal Verification Request 10

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
0.44 ms

Line 52 in File SafeMathWithRequire.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 58-61 in File SafeMathWithRequire.sol

```
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "undeflow");
        return a - b;
    }
```

The code meets the specification.

Formal Verification Request 11

Method will not encounter an assertion failure.

10, Dec 2019
0.52 ms

Line 53 in File SafeMathWithRequire.sol

```
53    //@CTK NO_ASF
```

Line 58-61 in File SafeMathWithRequire.sol

```
58    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
59        require(b <= a, "undeflow");
60        return a - b;
61    }
```

✅ The code meets the specification.

Formal Verification Request 12

sub

📅 10, Dec 2019
⏱ 0.71 ms

Line 54-57 in File SafeMathWithRequire.sol

```
54    /*@CTK sub
55      @tag assume_completion
56      @post __return == a - b
57     */
```

Line 58-61 in File SafeMathWithRequire.sol

```
58    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
59        require(b <= a, "undeflow");
60        return a - b;
61    }
```

✅ The code meets the specification.

Formal Verification Request 13

If method completes, integer overflow would not happen.

📅 10, Dec 2019
⏱ 11.32 ms

Line 66 in File SafeMathWithRequire.sol

```
66    //@CTK NO_OVERFLOW
```

Line 73-77 in File SafeMathWithRequire.sol

```
73    function add(uint256 a, uint256 b) internal pure returns (uint256) {
74        uint256 c = a + b;
75        require(c >= a, "overflow");
76        return c;
77    }
```

✅ The code meets the specification.

Formal Verification Request 14

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
0.35 ms

Line 67 in File SafeMathWithRequire.sol

```
67    //@CTK NO_BUF_OVERFLOW
```

Line 73-77 in File SafeMathWithRequire.sol

```
73    function add(uint256 a, uint256 b) internal pure returns (uint256) {
74        uint256 c = a + b;
75        require(c >= a, "overflow");
76        return c;
77    }
```

The code meets the specification.

Formal Verification Request 15

Method will not encounter an assertion failure.

10, Dec 2019
0.34 ms

Line 68 in File SafeMathWithRequire.sol

```
68    //@CTK NO_ASF
```

Line 73-77 in File SafeMathWithRequire.sol

```
73    function add(uint256 a, uint256 b) internal pure returns (uint256) {
74        uint256 c = a + b;
75        require(c >= a, "overflow");
76        return c;
77    }
```

The code meets the specification.

Formal Verification Request 16

add

10, Dec 2019
0.71 ms

Line 69-72 in File SafeMathWithRequire.sol

```
69    /*@CTK add
70      @tag assume_completion
71      @post __return == a + b
72     */
```

Line 73-77 in File SafeMathWithRequire.sol

```
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "overflow");
        return c;
    }
```

The code meets the specification.

Formal Verification Request 17

If method completes, integer overflow would not happen.

10, Dec 2019
34.84 ms

Line 18 in File CommonMinter.sol

```
18  //@CTK NO_OVERFLOW
```

Line 30-39 in File CommonMinter.sol

```
30  constructor(ERC1155ERC721 asset, ERC20 sand, uint256 feePerCopy, address admin, address
        feeReceiver)
31      public
32  {
33      _sand = sand;
34      _asset = asset;
35      _feePerCopy = feePerCopy;
36      _admin = admin;
37      _feeReceiver = feeReceiver;
38      _setMetaTransactionProcessor(address(sand), true);
39  }
```

The code meets the specification.

Formal Verification Request 18

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
0.49 ms

Line 19 in File CommonMinter.sol

```
19  //@CTK NO_BUF_OVERFLOW
```

Line 30-39 in File CommonMinter.sol

```
30  constructor(ERC1155ERC721 asset, ERC20 sand, uint256 feePerCopy, address admin, address
        feeReceiver)
31      public
32  {
33      _sand = sand;
34      _asset = asset;
35      _feePerCopy = feePerCopy;
36      _admin = admin;
37      _feeReceiver = feeReceiver;
```

```
        _setMetaTransactionProcessor(address(sand), true);
    }
```

The code meets the specification.

Formal Verification Request 19

Method will not encounter an assertion failure.

10, Dec 2019
0.47 ms

Line 20 in File CommonMinter.sol

```
    //@CTK NO_ASF
```

Line 30-39 in File CommonMinter.sol

```
    constructor(ERC1155ERC721 asset, ERC20 sand, uint256 feePerCopy, address admin, address
        feeReceiver)
        public
    {
        _sand = sand;
        _asset = asset;
        _feePerCopy = feePerCopy;
        _admin = admin;
        _feeReceiver = feeReceiver;
        _setMetaTransactionProcessor(address(sand), true);
    }
```

The code meets the specification.

Formal Verification Request 20

CommonMinter

10, Dec 2019
2.13 ms

Line 21-29 in File CommonMinter.sol

```
    /*@CTK CommonMinter
      @tag assume_completion
      @post __post._sand == sand
      @post __post._asset == asset
      @post __post._feePerCopy == feePerCopy
      @post __post._admin == admin
      @post __post._feeReceiver == feeReceiver
      @post __post._metaTransactionContracts[address(sand)] == true
     */
```

Line 30-39 in File CommonMinter.sol

```
    constructor(ERC1155ERC721 asset, ERC20 sand, uint256 feePerCopy, address admin, address
        feeReceiver)
        public
    {
```

```
        _sand = sand;
        _asset = asset;
        _feePerCopy = feePerCopy;
        _admin = admin;
        _feeReceiver = feeReceiver;
        _setMetaTransactionProcessor(address(sand), true);
    }
```

The code meets the specification.

Formal Verification Request 21

If method completes, integer overflow would not happen.

10, Dec 2019
12.78 ms

Line 43 in File CommonMinter.sol

```
43    //@CTK NO_OVERFLOW
```

Line 55-58 in File CommonMinter.sol

```
55    function setFeeReceiver(address newFeeReceiver) external {
56        require(msg.sender == _admin, "only admin can change the receiver");
57        _feeReceiver = newFeeReceiver;
58    }
```

The code meets the specification.

Formal Verification Request 22

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
0.34 ms

Line 44 in File CommonMinter.sol

```
44    //@CTK NO_BUF_OVERFLOW
```

Line 55-58 in File CommonMinter.sol

```
55    function setFeeReceiver(address newFeeReceiver) external {
56        require(msg.sender == _admin, "only admin can change the receiver");
57        _feeReceiver = newFeeReceiver;
58    }
```

The code meets the specification.

Formal Verification Request 23

Method will not encounter an assertion failure.

10, Dec 2019
0.32 ms

Line 45 in File CommonMinter.sol

```
45    //@CTK NO_ASF
```

Line 55-58 in File CommonMinter.sol

```
55    function setFeeReceiver(address newFeeReceiver) external {
56        require(msg.sender == _admin, "only admin can change the receiver");
57        _feeReceiver = newFeeReceiver;
58    }
```

✅ The code meets the specification.

Formal Verification Request 24

setFeeReceiver_change

📅 10, Dec 2019
⏱ 0.78 ms

Line 46-49 in File CommonMinter.sol

```
46    /*@CTK setFeeReceiver_change
47      @tag assume_completion
48      @post msg.sender == _admin
49     */
```

Line 55-58 in File CommonMinter.sol

```
55    function setFeeReceiver(address newFeeReceiver) external {
56        require(msg.sender == _admin, "only admin can change the receiver");
57        _feeReceiver = newFeeReceiver;
58    }
```

✅ The code meets the specification.

Formal Verification Request 25

setFeeReceiver_change

📅 10, Dec 2019
⏱ 0.78 ms

Line 50-54 in File CommonMinter.sol

```
50    /*@CTK setFeeReceiver_change
51      @tag assume_completion
52      @pre msg.sender == _admin
53      @post __post._feeReceiver == newFeeReceiver
54     */
```

Line 55-58 in File CommonMinter.sol

```
55    function setFeeReceiver(address newFeeReceiver) external {
56        require(msg.sender == _admin, "only admin can change the receiver");
57        _feeReceiver = newFeeReceiver;
58    }
```

✅ The code meets the specification.

Formal Verification Request 26

If method completes, integer overflow would not happen.

10, Dec 2019
12.74 ms

Line 62 in File CommonMinter.sol

```
62    //@CTK NO_OVERFLOW
```

Line 74-77 in File CommonMinter.sol

```
74    function setFeePerCopy(uint256 newFee) external {
75        require(msg.sender == _admin, "only admin allowed to set fee");
76        _feePerCopy = newFee;
77    }
```

The code meets the specification.

Formal Verification Request 27

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
0.32 ms

Line 63 in File CommonMinter.sol

```
63    //@CTK NO_BUF_OVERFLOW
```

Line 74-77 in File CommonMinter.sol

```
74    function setFeePerCopy(uint256 newFee) external {
75        require(msg.sender == _admin, "only admin allowed to set fee");
76        _feePerCopy = newFee;
77    }
```

The code meets the specification.

Formal Verification Request 28

Method will not encounter an assertion failure.

10, Dec 2019
0.32 ms

Line 64 in File CommonMinter.sol

```
64    //@CTK NO_ASF
```

Line 74-77 in File CommonMinter.sol

```
74    function setFeePerCopy(uint256 newFee) external {
75        require(msg.sender == _admin, "only admin allowed to set fee");
76        _feePerCopy = newFee;
77    }
```

The code meets the specification.

Formal Verification Request 29

setFeePerCopy_change

10, Dec 2019
0.74 ms

Line 65-68 in File CommonMinter.sol

```
65    /*@CTK setFeePerCopy_change
66      @tag assume_completion
67      @post msg.sender == _admin
68     */
```

Line 74-77 in File CommonMinter.sol

```
74    function setFeePerCopy(uint256 newFee) external {
75        require(msg.sender == _admin, "only admin allowed to set fee");
76        _feePerCopy = newFee;
77    }
```

The code meets the specification.

Formal Verification Request 30

setFeePerCopy_change

10, Dec 2019
0.74 ms

Line 69-73 in File CommonMinter.sol

```
69    /*@CTK setFeePerCopy_change
70      @tag assume_completion
71      @pre msg.sender == _admin
72      @post __post._feePerCopy == newFee
73     */
```

Line 74-77 in File CommonMinter.sol

```
74    function setFeePerCopy(uint256 newFee) external {
75        require(msg.sender == _admin, "only admin allowed to set fee");
76        _feePerCopy = newFee;
77    }
```

The code meets the specification.

Formal Verification Request 31

If method completes, integer overflow would not happen.

10, Dec 2019
62.85 ms

Line 7 in File Asset.sol

```
7    //@CTK NO_OVERFLOW
```

Line 16-20 in File Asset.sol

```
16    constructor(
17        address metaTransactionContract,
18        address assetAdmin,
19        address bouncerAdmin
20    ) public ERC1155ERC721(metaTransactionContract, assetAdmin, bouncerAdmin) {}
```

✅ The code meets the specification.

Formal Verification Request 32

Buffer overflow / array index out of bound would never happen.

📅 10, Dec 2019
⏱ 0.48 ms

Line 8 in File Asset.sol

```
8    //@CTK NO_BUF_OVERFLOW
```

Line 16-20 in File Asset.sol

```
16    constructor(
17        address metaTransactionContract,
18        address assetAdmin,
19        address bouncerAdmin
20    ) public ERC1155ERC721(metaTransactionContract, assetAdmin, bouncerAdmin) {}
```

✅ The code meets the specification.

Formal Verification Request 33

Method will not encounter an assertion failure.

📅 10, Dec 2019
⏱ 0.48 ms

Line 9 in File Asset.sol

```
9    //@CTK NO_ASF
```

Line 16-20 in File Asset.sol

```
16    constructor(
17        address metaTransactionContract,
18        address assetAdmin,
19        address bouncerAdmin
20    ) public ERC1155ERC721(metaTransactionContract, assetAdmin, bouncerAdmin) {}
```

✅ The code meets the specification.

Formal Verification Request 34

Asset

10, Dec 2019
2.89 ms

Line 10-15 in File Asset.sol

```
10  /*@CTK Asset
11    @tag assume_completion
12    @post __post._metaTransactionContracts[metaTransactionContract] == true
13    @post __post._admin == assetAdmin
14    @post __post._bouncerAdmin == bouncerAdmin
15   */
```

Line 16-20 in File Asset.sol

```
16  constructor(
17      address metaTransactionContract,
18      address assetAdmin,
19      address bouncerAdmin
20  ) public ERC1155ERC721(metaTransactionContract, assetAdmin, bouncerAdmin) {}
```

The code meets the specification.

Formal Verification Request 35

If method completes, integer overflow would not happen.

10, Dec 2019
26.3 ms

Line 51 in File BundleSandSale.sol

```
51  //@CTK NO_OVERFLOW
```

Line 67-82 in File BundleSandSale.sol

```
67  constructor(
68      address sandTokenContractAddress,
69      address assetTokenContractAddress,
70      address medianizerContractAddress,
71      address daiTokenContractAddress,
72      address admin,
73      address payable receivingWallet
74  ) public {
75      require(receivingWallet != address(0), "need a wallet to receive funds");
76      _medianizer = Medianizer(medianizerContractAddress);
77      _sand = ERC20(sandTokenContractAddress);
78      _asset = ERC1155ERC721(assetTokenContractAddress);
79      _dai = ERC20(daiTokenContractAddress);
80      _admin = admin;
81      _receivingWallet = receivingWallet;
82  }
```

The code meets the specification.

Formal Verification Request 36

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
0.46 ms

Line 52 in File BundleSandSale.sol

```
52    //@CTK NO_BUF_OVERFLOW
```

Line 67-82 in File BundleSandSale.sol

```
67    constructor(
68        address sandTokenContractAddress,
69        address assetTokenContractAddress,
70        address medianizerContractAddress,
71        address daiTokenContractAddress,
72        address admin,
73        address payable receivingWallet
74    ) public {
75        require(receivingWallet != address(0), "need a wallet to receive funds");
76        _medianizer = Medianizer(medianizerContractAddress);
77        _sand = ERC20(sandTokenContractAddress);
78        _asset = ERC1155ERC721(assetTokenContractAddress);
79        _dai = ERC20(daiTokenContractAddress);
80        _admin = admin;
81        _receivingWallet = receivingWallet;
82    }
```

The code meets the specification.

Formal Verification Request 37

Method will not encounter an assertion failure.

10, Dec 2019
0.44 ms

Line 53 in File BundleSandSale.sol

```
53    //@CTK NO_ASF
```

Line 67-82 in File BundleSandSale.sol

```
67    constructor(
68        address sandTokenContractAddress,
69        address assetTokenContractAddress,
70        address medianizerContractAddress,
71        address daiTokenContractAddress,
72        address admin,
73        address payable receivingWallet
74    ) public {
75        require(receivingWallet != address(0), "need a wallet to receive funds");
76        _medianizer = Medianizer(medianizerContractAddress);
77        _sand = ERC20(sandTokenContractAddress);
78        _asset = ERC1155ERC721(assetTokenContractAddress);
79        _dai = ERC20(daiTokenContractAddress);
80        _admin = admin;
```

```
        _receivingWallet = receivingWallet;
    }
```

The code meets the specification.

Formal Verification Request 38

BundleSandSale_require

10, Dec 2019
0.39 ms

Line 54-57 in File BundleSandSale.sol

```
54    /*@CTK BundleSandSale_require
55      @tag assume_completion
56      @post receivingWallet != address(0)
57     */
```

Line 67-82 in File BundleSandSale.sol

```
67    constructor(
68        address sandTokenContractAddress,
69        address assetTokenContractAddress,
70        address medianizerContractAddress,
71        address daiTokenContractAddress,
72        address admin,
73        address payable receivingWallet
74    ) public {
75        require(receivingWallet != address(0), "need a wallet to receive funds");
76        _medianizer = Medianizer(medianizerContractAddress);
77        _sand = ERC20(sandTokenContractAddress);
78        _asset = ERC1155ERC721(assetTokenContractAddress);
79        _dai = ERC20(daiTokenContractAddress);
80        _admin = admin;
81        _receivingWallet = receivingWallet;
82    }
```

The code meets the specification.

Formal Verification Request 39

BundleSandSale_change

10, Dec 2019
3.5 ms

Line 58-66 in File BundleSandSale.sol

```
58    /*@CTK BundleSandSale_change
59      @tag assume_completion
60      @post __post._medianizer == medianizerContractAddress
61      @post __post._sand == sandTokenContractAddress
62      @post __post._asset == assetTokenContractAddress
63      @post __post._dai == daiTokenContractAddress
64      @post __post._admin == admin
```

```
65      @post __post._receivingWallet == receivingWallet
66     */
```

Line 67-82 in File BundleSandSale.sol

```
67    constructor(
68        address sandTokenContractAddress,
69        address assetTokenContractAddress,
70        address medianizerContractAddress,
71        address daiTokenContractAddress,
72        address admin,
73        address payable receivingWallet
74    ) public {
75        require(receivingWallet != address(0), "need a wallet to receive funds");
76        _medianizer = Medianizer(medianizerContractAddress);
77        _sand = ERC20(sandTokenContractAddress);
78        _asset = ERC1155ERC721(assetTokenContractAddress);
79        _dai = ERC20(daiTokenContractAddress);
80        _admin = admin;
81        _receivingWallet = receivingWallet;
82    }
```

✅ The code meets the specification.

Formal Verification Request 40

If method completes, integer overflow would not happen.

📅 10, Dec 2019
⏱ 20.73 ms

Line 86 in File BundleSandSale.sol

```
86    //@CTK NO_OVERFLOW
```

Line 100-104 in File BundleSandSale.sol

```
100   function setReceivingWallet(address payable newWallet) external {
101       require(newWallet != address(0), "receiving wallet cannot be zero address");
102       require(msg.sender == _admin, "only admin can change the receiving wallet");
103       _receivingWallet = newWallet;
104   }
```

✅ The code meets the specification.

Formal Verification Request 41

Buffer overflow / array index out of bound would never happen.

📅 10, Dec 2019
⏱ 0.57 ms

Line 87 in File BundleSandSale.sol

```
87    //@CTK NO_BUF_OVERFLOW
```

Line 100-104 in File BundleSandSale.sol

```
100    function setReceivingWallet(address payable newWallet) external {
101        require(newWallet != address(0), "receiving wallet cannot be zero address");
102        require(msg.sender == _admin, "only admin can change the receiving wallet");
103        _receivingWallet = newWallet;
104    }
```

The code meets the specification.

Formal Verification Request 42

Method will not encounter an assertion failure.

10, Dec 2019
0.42 ms

Line 88 in File BundleSandSale.sol

```
88    //@CTK NO_ASF
```

Line 100-104 in File BundleSandSale.sol

```
100    function setReceivingWallet(address payable newWallet) external {
101        require(newWallet != address(0), "receiving wallet cannot be zero address");
102        require(msg.sender == _admin, "only admin can change the receiving wallet");
103        _receivingWallet = newWallet;
104    }
```

The code meets the specification.

Formal Verification Request 43

setReceivingWallet_require

10, Dec 2019
2.35 ms

Line 89-93 in File BundleSandSale.sol

```
89    /*@CTK setReceivingWallet_require
90      @tag assume_completion
91      @post newWallet != address(0)
92      @post msg.sender == _admin
93     */
```

Line 100-104 in File BundleSandSale.sol

```
100    function setReceivingWallet(address payable newWallet) external {
101        require(newWallet != address(0), "receiving wallet cannot be zero address");
102        require(msg.sender == _admin, "only admin can change the receiving wallet");
103        _receivingWallet = newWallet;
104    }
```

The code meets the specification.

Formal Verification Request 44

setReceivingWallet_change

10, Dec 2019
1.72 ms

Line 94-99 in File BundleSandSale.sol

```
/*@CTK setReceivingWallet_change
  @tag assume_completion
  @pre newWallet != address(0)
  @pre msg.sender == _admin
  @post __post._receivingWallet == newWallet
 */
```

Line 100-104 in File BundleSandSale.sol

```
function setReceivingWallet(address payable newWallet) external {
    require(newWallet != address(0), "receiving wallet cannot be zero address");
    require(msg.sender == _admin, "only admin can change the receiving wallet");
    _receivingWallet = newWallet;
}
```

The code meets the specification.

Formal Verification Request 45

If method completes, integer overflow would not happen.

10, Dec 2019
7.85 ms

Line 106 in File BundleSandSale.sol

```
//@CTK NO_OVERFLOW
```

Line 112-134 in File BundleSandSale.sol

```
function _transferPack(uint256 saleIndex, uint256 numPacks, address to) internal {
    uint256 sandAmountPerPack = sales[saleIndex].sandAmount;
    require(
        _sand.transferFrom(address(this), to, sandAmountPerPack.mul(numPacks)),
        "Sand Transfer failed"
    );
    uint256[] memory ids = sales[saleIndex].ids;
    uint256[] memory amounts = sales[saleIndex].amounts;
    uint256 numIds = ids.length;
    /*@FIXME FAIL "_transferPack_loop"
      @inv i <= sales[saleIndex].ids.length
      @post forall j: uint256. (j >= 0 /\ j < sales[saleIndex].ids.length) -> (__post.
          sales[saleIndex].amounts[j] == sales[saleIndex].amounts[j] * numPacks)
      @post i == sales[saleIndex].ids.length
     */
    for (uint256 i = 0; i < numIds; i++) {
        amounts[i] = amounts[i].mul(numPacks);
    }
    _asset.safeBatchTransferFrom(address(this), to, ids, amounts, "");
}
```

The code meets the specification.

Formal Verification Request 46

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
3.95 ms

Line 107 in File BundleSandSale.sol

```
107    //@CTK FAIL NO_BUF_OVERFLOW
```

Line 112-134 in File BundleSandSale.sol

```
112    function _transferPack(uint256 saleIndex, uint256 numPacks, address to) internal {
113        uint256 sandAmountPerPack = sales[saleIndex].sandAmount;
114        require(
115            _sand.transferFrom(address(this), to, sandAmountPerPack.mul(numPacks)),
116            "Sand Transfer failed"
117        );
118        uint256[] memory ids = sales[saleIndex].ids;
119        uint256[] memory amounts = sales[saleIndex].amounts;
120        uint256 numIds = ids.length;
121        /*@FIXME FAIL "_transferPack_loop"
122          @inv i <= sales[saleIndex].ids.length
123          @post forall j: uint256. (j >= 0 /\ j < sales[saleIndex].ids.length) -> (__post.
               sales[saleIndex].amounts[j] == sales[saleIndex].amounts[j] * numPacks)
124          @post i == sales[saleIndex].ids.length
125         */
126        for (uint256 i = 0; i < numIds; i++) {
127            amounts[i] = amounts[i].mul(numPacks);
128        }
129        _asset.safeBatchTransferFrom(address(this), to, ids, amounts, "");
130    }
```

This code violates the specification.

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          numPacks = 0
5          saleIndex = 128
6          to = 0
7      }
8      This = 0
9      Internal = {
10         __has_assertion_failure = false
11         __has_buf_overflow = false
12         __has_overflow = false
13         __has_returned = false
14         __reverted = false
15         msg = {
16           "gas": 0,
17           "sender": 0,
18           "value": 0
19         }
20     }
```

```
    Other = {
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": "ALL_OTHERS",
        "value": {
          "contract_name": "BundleSandSale",
          "balance": 0,
          "contract": {
            "ERC1155_RECEIVED": "AAAA",
            "ERC1155_BATCH_RECEIVED": "IIII",
            "_medianizer": 0,
            "_dai": 0,
            "_sand": 0,
            "_asset": 0,
            "_receivingWallet": 0,
            "sales": [
              {
                "ids": [
                  {
                    "key": "ALL_OTHERS",
                    "value": 8
                  }
                ],
                "amounts": [
                  {
                    "key": "ALL_OTHERS",
                    "value": 8
                  }
                ],
                "sandAmount": 128,
                "priceUSD": 128,
                "numPacksLeft": 128
              }
            ],
            "_admin": 0
          }
        }
      }
    ]

After Execution:
    Input = {
        numPacks = 0
        saleIndex = 128
        to = 0
    }
    This = 0
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = true
        __has_overflow = false
        __has_returned = false
        __reverted = false
```

```
      msg = {
        "gas": 0,
        "sender": 0,
        "value": 0
      }
    }
    Other = {
      block = {
        "number": 0,
        "timestamp": 0
      }
    }
    Address_Map = [
      {
        "key": "ALL_OTHERS",
        "value": {
          "contract_name": "BundleSandSale",
          "balance": 0,
          "contract": {
            "ERC1155_RECEIVED": "AAAA",
            "ERC1155_BATCH_RECEIVED": "IIII",
            "_medianizer": 0,
            "_dai": 0,
            "_sand": 0,
            "_asset": 0,
            "_receivingWallet": 0,
            "sales": [
              {
                "ids": [
                  {
                    "key": "ALL_OTHERS",
                    "value": 8
                  }
                ],
                "amounts": [
                  {
                    "key": "ALL_OTHERS",
                    "value": 8
                  }
                ],
                "sandAmount": 128,
                "priceUSD": 128,
                "numPacksLeft": 128
              }
            ],
            "_admin": 0
          }
        }
      }
    ]
```

Formal Verification Request 47

Method will not encounter an assertion failure.

📅 10, Dec 2019
⏱ 0.33 ms

Line 108 in File BundleSandSale.sol

```
108    //@CTK NO_ASF
```

Line 112-134 in File BundleSandSale.sol

```
112    function _transferPack(uint256 saleIndex, uint256 numPacks, address to) internal {
113        uint256 sandAmountPerPack = sales[saleIndex].sandAmount;
114        require(
115            _sand.transferFrom(address(this), to, sandAmountPerPack.mul(numPacks)),
116            "Sand Transfer failed"
117        );
118        uint256[] memory ids = sales[saleIndex].ids;
119        uint256[] memory amounts = sales[saleIndex].amounts;
120        uint256 numIds = ids.length;
121        /*@FIXME FAIL "_transferPack_loop"
122          @inv i <= sales[saleIndex].ids.length
123          @post forall j: uint256. (j >= 0 /\ j < sales[saleIndex].ids.length) -> (__post.
               sales[saleIndex].amounts[j] == sales[saleIndex].amounts[j] * numPacks)
124          @post i == sales[saleIndex].ids.length
125         */
126        for (uint256 i = 0; i < numIds; i++) {
127            amounts[i] = amounts[i].mul(numPacks);
128        }
129        _asset.safeBatchTransferFrom(address(this), to, ids, amounts, "");
130    }
```

✅ The code meets the specification.

Formal Verification Request 48

__transferPack

📅 10, Dec 2019
⏱ 0.26 ms

Line 109-111 in File BundleSandSale.sol

```
109    /*@CTK _transferPack
110      @tag assume_completion
111     */
```

Line 112-134 in File BundleSandSale.sol

```
112    function _transferPack(uint256 saleIndex, uint256 numPacks, address to) internal {
113        uint256 sandAmountPerPack = sales[saleIndex].sandAmount;
114        require(
115            _sand.transferFrom(address(this), to, sandAmountPerPack.mul(numPacks)),
116            "Sand Transfer failed"
117        );
118        uint256[] memory ids = sales[saleIndex].ids;
119        uint256[] memory amounts = sales[saleIndex].amounts;
120        uint256 numIds = ids.length;
121        /*@FIXME FAIL "_transferPack_loop"
122          @inv i <= sales[saleIndex].ids.length
123          @post forall j: uint256. (j >= 0 /\ j < sales[saleIndex].ids.length) -> (__post.
               sales[saleIndex].amounts[j] == sales[saleIndex].amounts[j] * numPacks)
124          @post i == sales[saleIndex].ids.length
```

```
        */
        for (uint256 i = 0; i < numIds; i++) {
            amounts[i] = amounts[i].mul(numPacks);
        }
        _asset.safeBatchTransferFrom(address(this), to, ids, amounts, "");
    }
```

The code meets the specification.

Formal Verification Request 49

If method completes, integer overflow would not happen.

10, Dec 2019
239.87 ms

Line 142 in File BundleSandSale.sol

```
142     //@CTK NO_OVERFLOW
```

Line 150-169 in File BundleSandSale.sol

```
150     function buyBundleWithEther(uint256 saleId, uint256 numPacks, address to) external
            payable {
151         require(saleId > 0, "invalid saleId");
152         uint256 saleIndex = saleId - 1;
153         uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
154         require(numPacksLeft >= numPacks, "not enough packs on sale");
155         sales[saleIndex].numPacksLeft = numPacksLeft - numPacks;
156
157         uint256 USDRequired = numPacks.mul(sales[saleIndex].priceUSD);
158         uint256 ETHRequired = getEtherAmountWithUSD(USDRequired);
159         require(msg.value >= ETHRequired, "not enough ether sent");
160         uint256 leftOver = msg.value - ETHRequired;
161         if(leftOver > 0) {
162             msg.sender.transfer(leftOver); // refund extra
163         }
164         address(_receivingWallet).transfer(ETHRequired);
165         _transferPack(saleIndex, numPacks, to);
166         emit BundleSold(saleId, msg.sender, numPacks, address(0), ETHRequired);
167     }
```

The code meets the specification.

Formal Verification Request 50

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
441.93 ms

Line 143 in File BundleSandSale.sol

```
143     //@CTK FAIL NO_BUF_OVERFLOW
```

Line 150-169 in File BundleSandSale.sol

```
    function buyBundleWithEther(uint256 saleId, uint256 numPacks, address to) external
        payable {
        require(saleId > 0, "invalid saleId");
        uint256 saleIndex = saleId - 1;
        uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
        require(numPacksLeft >= numPacks, "not enough packs on sale");
        sales[saleIndex].numPacksLeft = numPacksLeft - numPacks;

        uint256 USDRequired = numPacks.mul(sales[saleIndex].priceUSD);
        uint256 ETHRequired = getEtherAmountWithUSD(USDRequired);
        require(msg.value >= ETHRequired, "not enough ether sent");
        uint256 leftOver = msg.value - ETHRequired;
        if(leftOver > 0) {
            msg.sender.transfer(leftOver); // refund extra
        }
        address(_receivingWallet).transfer(ETHRequired);
        _transferPack(saleIndex, numPacks, to);
        emit BundleSold(saleId, msg.sender, numPacks, address(0), ETHRequired);
    }
```

This code violates the specification.

```
Counter Example:
Before Execution:
    Input = {
        numPacks = 17
        saleId = 8
        to = 0
    }
    This = 16
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 17
        }
    }
    Other = {
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": 16,
        "value": {
          "contract_name": "BundleSandSale",
          "balance": 0,
          "contract": {
            "ERC1155_RECEIVED": "BBBB",
            "ERC1155_BATCH_RECEIVED": "BBBB",
            "_medianizer": 0,
            "_dai": 64,
```

```
          "_sand": 0,
          "_asset": 0,
          "_receivingWallet": 0,
          "sales": [],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]

Function invocation is reverted.
```

Formal Verification Request 51

Method will not encounter an assertion failure.

10, Dec 2019
56.77 ms

Line 144 in File BundleSandSale.sol

```
144    //@CTK FAIL NO_ASF
```

Line 150-169 in File BundleSandSale.sol

```
150    function buyBundleWithEther(uint256 saleId, uint256 numPacks, address to) external
           payable {
151        require(saleId > 0, "invalid saleId");
152        uint256 saleIndex = saleId - 1;
153        uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
154        require(numPacksLeft >= numPacks, "not enough packs on sale");
155        sales[saleIndex].numPacksLeft = numPacksLeft - numPacks;
156
157        uint256 USDRequired = numPacks.mul(sales[saleIndex].priceUSD);
158        uint256 ETHRequired = getEtherAmountWithUSD(USDRequired);
159        require(msg.value >= ETHRequired, "not enough ether sent");
160        uint256 leftOver = msg.value - ETHRequired;
161        if(leftOver > 0) {
162            msg.sender.transfer(leftOver); // refund extra
163        }
164        address(_receivingWallet).transfer(ETHRequired);
165        _transferPack(saleIndex, numPacks, to);
166        emit BundleSold(saleId, msg.sender, numPacks, address(0), ETHRequired);
167    }
```

This code violates the specification.

```
Counter Example:
Before Execution:
    Input = {
        numPacks = 128
        saleId = 32
        to = 0
    }
    This = 1
```

```
Internal = {
    __has_assertion_failure = false
    __has_buf_overflow = false
    __has_overflow = false
    __has_returned = false
    __reverted = false
    msg = {
      "gas": 0,
      "sender": 0,
      "value": 59
    }
}
Other = {
    block = {
      "number": 0,
      "timestamp": 0
    }
}
Address_Map = [
  {
    "key": 1,
    "value": {
      "contract_name": "BundleSandSale",
      "balance": 0,
      "contract": {
        "ERC1155_RECEIVED": "||||",
        "ERC1155_BATCH_RECEIVED": "||||",
        "_medianizer": 0,
        "_dai": 0,
        "_sand": 0,
        "_asset": 0,
        "_receivingWallet": 0,
        "sales": [
          {
            "key": 31,
            "value": {
              "ids": [],
              "amounts": [],
              "sandAmount": 0,
              "priceUSD": 0,
              "numPacksLeft": 128
            }
          },
          {
            "key": "ALL_OTHERS",
            "value": {
              "ids": [
                {
                  "key": "ALL_OTHERS",
                  "value": 128
                }
              ],
              "amounts": [
                {
                  "key": "ALL_OTHERS",
                  "value": 128
                }
              ],
```

```
                "sandAmount": 128,
                "priceUSD": 128,
                "numPacksLeft": 128
              }
            }
          ],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]

Function invocation is reverted.
```

Formal Verification Request 52

buyBundleWithEther_require

10, Dec 2019
35.92 ms

Line 145-149 in File BundleSandSale.sol

```
145   /*@CTK buyBundleWithEther_require
146     @tag assume_completion
147     @post saleId > 0
148     @post sales[saleId - 1].numPacksLeft >= numPacks
149    */
```

Line 150-169 in File BundleSandSale.sol

```
150   function buyBundleWithEther(uint256 saleId, uint256 numPacks, address to) external
        payable {
151       require(saleId > 0, "invalid saleId");
152       uint256 saleIndex = saleId - 1;
153       uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
154       require(numPacksLeft >= numPacks, "not enough packs on sale");
155       sales[saleIndex].numPacksLeft = numPacksLeft - numPacks;
156
157       uint256 USDRequired = numPacks.mul(sales[saleIndex].priceUSD);
158       uint256 ETHRequired = getEtherAmountWithUSD(USDRequired);
159       require(msg.value >= ETHRequired, "not enough ether sent");
160       uint256 leftOver = msg.value - ETHRequired;
161       if(leftOver > 0) {
162           msg.sender.transfer(leftOver); // refund extra
163       }
164       address(_receivingWallet).transfer(ETHRequired);
165       _transferPack(saleIndex, numPacks, to);
166       emit BundleSold(saleId, msg.sender, numPacks, address(0), ETHRequired);
167   }
```

The code meets the specification.

Formal Verification Request 53

If method completes, integer overflow would not happen.

10, Dec 2019
65.2 ms

Line 177 in File BundleSandSale.sol

```
177    //@CTK NO_OVERFLOW
```

Line 185-199 in File BundleSandSale.sol

```
185    function buyBundleWithDai(uint256 saleId, uint256 numPacks, address to) external {
186        require(saleId > 0, "invalid saleId");
187        uint256 saleIndex = saleId - 1;
188        uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
189        require(numPacksLeft >= numPacks, "not enough packs on sale");
190        sales[saleIndex].numPacksLeft = numPacksLeft - numPacks;
191
192        uint256 USDRequired = numPacks.mul(sales[saleIndex].priceUSD);
193        require(_dai.transferFrom(msg.sender, _receivingWallet, USDRequired), "failed to
               transfer dai");
194        _transferPack(saleIndex, numPacks, to);
195
196        emit BundleSold(saleId, msg.sender, numPacks, address(_dai), USDRequired);
197    }
```

The code meets the specification.

Formal Verification Request 54

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
63.84 ms

Line 178 in File BundleSandSale.sol

```
178    //@CTK FAIL NO_BUF_OVERFLOW
```

Line 185-199 in File BundleSandSale.sol

```
185    function buyBundleWithDai(uint256 saleId, uint256 numPacks, address to) external {
186        require(saleId > 0, "invalid saleId");
187        uint256 saleIndex = saleId - 1;
188        uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
189        require(numPacksLeft >= numPacks, "not enough packs on sale");
190        sales[saleIndex].numPacksLeft = numPacksLeft - numPacks;
191
192        uint256 USDRequired = numPacks.mul(sales[saleIndex].priceUSD);
193        require(_dai.transferFrom(msg.sender, _receivingWallet, USDRequired), "failed to
               transfer dai");
194        _transferPack(saleIndex, numPacks, to);
195
196        emit BundleSold(saleId, msg.sender, numPacks, address(_dai), USDRequired);
197    }
```

This code violates the specification.

```
Counter Example:
Before Execution:
    Input = {
        numPacks = 70
        saleId = 17
        to = 0
    }
    This = 0
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 0
        }
    }
    Other = {
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": 0,
        "value": {
          "contract_name": "BundleSandSale",
          "balance": 0,
          "contract": {
            "ERC1155_RECEIVED": "\u00b1\u00b1\u00b1\u00b1",
            "ERC1155_BATCH_RECEIVED": "\u00b1\u00b1\u00b1\u00b1",
            "_medianizer": 0,
            "_dai": 0,
            "_sand": 0,
            "_asset": 0,
            "_receivingWallet": 0,
            "sales": [
              {
                "ids": [
                  {
                    "key": "ALL_OTHERS",
                    "value": 112
                  }
                ],
                "amounts": [
                  {
                    "key": "ALL_OTHERS",
                    "value": 112
                  }
                ],
                "sandAmount": 112,
                "priceUSD": 112,
                "numPacksLeft": 112
              }
```

```
          ],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]

Function invocation is reverted.
```

Formal Verification Request 55

Method will not encounter an assertion failure.

10, Dec 2019
7.4 ms

Line 179 in File BundleSandSale.sol

```
179    //@CTK NO_ASF
```

Line 185-199 in File BundleSandSale.sol

```
185    function buyBundleWithDai(uint256 saleId, uint256 numPacks, address to) external {
186       require(saleId > 0, "invalid saleId");
187       uint256 saleIndex = saleId - 1;
188       uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
189       require(numPacksLeft >= numPacks, "not enough packs on sale");
190       sales[saleIndex].numPacksLeft = numPacksLeft - numPacks;
191
192       uint256 USDRequired = numPacks.mul(sales[saleIndex].priceUSD);
193       require(_dai.transferFrom(msg.sender, _receivingWallet, USDRequired), "failed to
             transfer dai");
194       _transferPack(saleIndex, numPacks, to);
195
196       emit BundleSold(saleId, msg.sender, numPacks, address(_dai), USDRequired);
197    }
```

The code meets the specification.

Formal Verification Request 56

buyBundleWithDai_require

10, Dec 2019
15.42 ms

Line 180-184 in File BundleSandSale.sol

```
180    /*@CTK buyBundleWithDai_require
181      @tag assume_completion
182      @post saleId > 0
183      @post sales[saleId - 1].numPacksLeft >= numPacks
184     */
```

Line 185-199 in File BundleSandSale.sol

```
185    function buyBundleWithDai(uint256 saleId, uint256 numPacks, address to) external {
186        require(saleId > 0, "invalid saleId");
187        uint256 saleIndex = saleId - 1;
188        uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
189        require(numPacksLeft >= numPacks, "not enough packs on sale");
190        sales[saleIndex].numPacksLeft = numPacksLeft - numPacks;
191
192        uint256 USDRequired = numPacks.mul(sales[saleIndex].priceUSD);
193        require(_dai.transferFrom(msg.sender, _receivingWallet, USDRequired), "failed to
               transfer dai");
194        _transferPack(saleIndex, numPacks, to);
195
196        emit BundleSold(saleId, msg.sender, numPacks, address(_dai), USDRequired);
197    }
```

✅ The code meets the specification.

Formal Verification Request 57

If method completes, integer overflow would not happen.

📅 10, Dec 2019
⏱ 14.92 ms

Line 201 in File BundleSandSale.sol

```
201    //@CTK NO_OVERFLOW
```

Line 214-219 in File BundleSandSale.sol

```
214    function getSaleInfo(uint256 saleId) external view returns(uint256 priceUSD, uint256
           numPacksLeft) {
215        require(saleId > 0, "invalid saleId");
216        uint256 saleIndex = saleId - 1;
217        priceUSD = sales[saleIndex].priceUSD;
218        numPacksLeft = sales[saleIndex].numPacksLeft;
219    }
```

✅ The code meets the specification.

Formal Verification Request 58

Buffer overflow / array index out of bound would never happen.

📅 10, Dec 2019
⏱ 7.81 ms

Line 202 in File BundleSandSale.sol

```
202    //@CTK FAIL NO_BUF_OVERFLOW
```

Line 214-219 in File BundleSandSale.sol

```
    function getSaleInfo(uint256 saleId) external view returns(uint256 priceUSD, uint256
        numPacksLeft) {
        require(saleId > 0, "invalid saleId");
        uint256 saleIndex = saleId - 1;
        priceUSD = sales[saleIndex].priceUSD;
        numPacksLeft = sales[saleIndex].numPacksLeft;
    }
```

This code violates the specification.

```
Counter Example:
Before Execution:
    Input = {
        saleId = 129
    }
    This = 0
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 0
        }
    }
    Other = {
        block = {
          "number": 0,
          "timestamp": 0
        }
        numPacksLeft = 0
        priceUSD = 0
    }
    Address_Map = [
      {
        "key": "ALL_OTHERS",
        "value": {
          "contract_name": "BundleSandSale",
          "balance": 0,
          "contract": {
            "ERC1155_RECEIVED": "AAAA",
            "ERC1155_BATCH_RECEIVED": "AAAA",
            "_medianizer": 0,
            "_dai": 0,
            "_sand": 0,
            "_asset": 0,
            "_receivingWallet": 0,
            "sales": [
              {
                "ids": [],
                "amounts": [],
                "sandAmount": 0,
                "priceUSD": 0,
                "numPacksLeft": 0
              }
            ],
```

```
          "_admin": 0
        }
      }
    }
  ]

After Execution:
  Input = {
      saleId = 129
  }
  This = 0
  Internal = {
      __has_assertion_failure = false
      __has_buf_overflow = true
      __has_overflow = false
      __has_returned = false
      __reverted = false
      msg = {
        "gas": 0,
        "sender": 0,
        "value": 0
      }
  }
  Other = {
      block = {
        "number": 0,
        "timestamp": 0
      }
      numPacksLeft = 0
      priceUSD = 0
  }
  Address_Map = [
    {
      "key": "ALL_OTHERS",
      "value": {
        "contract_name": "BundleSandSale",
        "balance": 0,
        "contract": {
          "ERC1155_RECEIVED": "AAAA",
          "ERC1155_BATCH_RECEIVED": "AAAA",
          "_medianizer": 0,
          "_dai": 0,
          "_sand": 0,
          "_asset": 0,
          "_receivingWallet": 0,
          "sales": [
            {
              "ids": [],
              "amounts": [],
              "sandAmount": 0,
              "priceUSD": 0,
              "numPacksLeft": 0
            }
          ],
          "_admin": 0
        }
      }
    }
```

```
]
```

Formal Verification Request 59

Method will not encounter an assertion failure.

10, Dec 2019
0.48 ms

Line 203 in File BundleSandSale.sol

```
//@CTK NO_ASF
```

Line 214-219 in File BundleSandSale.sol

```
function getSaleInfo(uint256 saleId) external view returns(uint256 priceUSD, uint256
    numPacksLeft) {
    require(saleId > 0, "invalid saleId");
    uint256 saleIndex = saleId - 1;
    priceUSD = sales[saleIndex].priceUSD;
    numPacksLeft = sales[saleIndex].numPacksLeft;
}
```

The code meets the specification.

Formal Verification Request 60

getSaleInfo_require

10, Dec 2019
0.53 ms

Line 204-207 in File BundleSandSale.sol

```
/*@CTK getSaleInfo_require
  @tag assume_completion
  @post saleId > 0
 */
```

Line 214-219 in File BundleSandSale.sol

```
function getSaleInfo(uint256 saleId) external view returns(uint256 priceUSD, uint256
    numPacksLeft) {
    require(saleId > 0, "invalid saleId");
    uint256 saleIndex = saleId - 1;
    priceUSD = sales[saleIndex].priceUSD;
    numPacksLeft = sales[saleIndex].numPacksLeft;
}
```

The code meets the specification.

Formal Verification Request 61

getSaleInfo_change

10, Dec 2019
1.79 ms

Line 208-213 in File BundleSandSale.sol

```
/*@CTK getSaleInfo_change
  @tag assume_completion
  @pre saleId > 0
  @post priceUSD == sales[saleId - 1].priceUSD
  @post numPacksLeft == sales[saleId - 1].numPacksLeft
 */
```

Line 214-219 in File BundleSandSale.sol

```
function getSaleInfo(uint256 saleId) external view returns(uint256 priceUSD, uint256
    numPacksLeft) {
    require(saleId > 0, "invalid saleId");
    uint256 saleIndex = saleId - 1;
    priceUSD = sales[saleIndex].priceUSD;
    numPacksLeft = sales[saleIndex].numPacksLeft;
}
```

✅ The code meets the specification.

Formal Verification Request 62

If method completes, integer overflow would not happen.

📅 10, Dec 2019
⏱ 38.08 ms

Line 221 in File BundleSandSale.sol

```
//@CTK NO_OVERFLOW
```

Line 235-256 in File BundleSandSale.sol

```
function withdrawSale(uint256 saleId, address to) external onlyAdmin() {
    require(saleId > 0, "invalid saleId");
    uint256 saleIndex = saleId - 1;
    uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
    sales[saleIndex].numPacksLeft = 0;

    uint256[] memory ids = sales[saleIndex].ids;
    uint256[] memory amounts = sales[saleIndex].amounts;
    uint256 numIds = ids.length;
    /*@FIXME FAIL "withdrawSale_loop"
      @inv i <= sales[saleId - 1].ids.length
      @post forall j: uint256. (j >= 0 /\ j < sales[saleId - 1].ids.length) -> (__post.
          sales[saleId - 1].amounts[j] == sales[saleId - 1].amounts[j] * sales[saleId -
          1].numPacksLeft)
      @post i == sales[saleId - 1].ids.length
     */
    for (uint256 i = 0; i < numIds; i++) {
        amounts[i] = amounts[i].mul(numPacksLeft);
    }
    require(_sand.transferFrom(address(this), to, numPacksLeft.mul(sales[saleIndex].
        sandAmount)), "transfer fo Sand failed");
    _asset.safeBatchTransferFrom(address(this), to, ids, amounts, "");
}
```

The code meets the specification.

Formal Verification Request 63

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
29.81 ms

Line 222 in File BundleSandSale.sol

```
222    //@CTK FAIL NO_BUF_OVERFLOW
```

Line 235-256 in File BundleSandSale.sol

```
235    function withdrawSale(uint256 saleId, address to) external onlyAdmin() {
236        require(saleId > 0, "invalid saleId");
237        uint256 saleIndex = saleId - 1;
238        uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
239        sales[saleIndex].numPacksLeft = 0;
240
241        uint256[] memory ids = sales[saleIndex].ids;
242        uint256[] memory amounts = sales[saleIndex].amounts;
243        uint256 numIds = ids.length;
244        /*@FIXME FAIL "withdrawSale_loop"
245          @inv i <= sales[saleId - 1].ids.length
246          @post forall j: uint256. (j >= 0 /\ j < sales[saleId - 1].ids.length) -> (__post.
               sales[saleId - 1].amounts[j] == sales[saleId - 1].amounts[j] * sales[saleId -
               1].numPacksLeft)
247          @post i == sales[saleId - 1].ids.length
248         */
249        for (uint256 i = 0; i < numIds; i++) {
250            amounts[i] = amounts[i].mul(numPacksLeft);
251        }
252        require(_sand.transferFrom(address(this), to, numPacksLeft.mul(sales[saleIndex].
               sandAmount)), "transfer fo Sand failed");
253        _asset.safeBatchTransferFrom(address(this), to, ids, amounts, "");
254    }
```

This code violates the specification.

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          saleId = 129
5          to = 0
6      }
7      This = 0
8      Internal = {
9          __has_assertion_failure = false
10         __has_buf_overflow = false
11         __has_overflow = false
12         __has_returned = false
13         __reverted = false
14         msg = {
15           "gas": 0,
16           "sender": 0,
17           "value": 0
```

```
      }
    }
    Other = {
      block = {
        "number": 0,
        "timestamp": 0
      }
    }
    Address_Map = [
      {
        "key": 0,
        "value": {
          "contract_name": "BundleSandSale",
          "balance": 0,
          "contract": {
            "ERC1155_RECEIVED": "\u00c1\u00c1\u00c1\u00c1",
            "ERC1155_BATCH_RECEIVED": "\u00c1\u00c1\u00c1\u00c1",
            "_medianizer": 0,
            "_dai": 0,
            "_sand": 0,
            "_asset": 0,
            "_receivingWallet": 0,
            "sales": [
              {
                "ids": [
                  {
                    "key": "ALL_OTHERS",
                    "value": 128
                  }
                ],
                "amounts": [
                  {
                    "key": "ALL_OTHERS",
                    "value": 128
                  }
                ],
                "sandAmount": 128,
                "priceUSD": 128,
                "numPacksLeft": 128
              },
              {
                "ids": [
                  {
                    "key": "ALL_OTHERS",
                    "value": 128
                  }
                ],
                "amounts": [
                  {
                    "key": "ALL_OTHERS",
                    "value": 128
                  }
                ],
                "sandAmount": 128,
                "priceUSD": 128,
                "numPacksLeft": 128
              },
              {
```

```
              "ids": [
                {
                  "key": "ALL_OTHERS",
                  "value": 128
                }
              ],
              "amounts": [
                {
                  "key": "ALL_OTHERS",
                  "value": 128
                }
              ],
              "sandAmount": 128,
              "priceUSD": 128,
              "numPacksLeft": 128
            },
            {
              "ids": [
                {
                  "key": "ALL_OTHERS",
                  "value": 128
                }
              ],
              "amounts": [
                {
                  "key": "ALL_OTHERS",
                  "value": 128
                }
              ],
              "sandAmount": 128,
              "priceUSD": 128,
              "numPacksLeft": 128
            }
          ],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]

After Execution:
  Input = {
    saleId = 129
    to = 0
  }
  This = 0
  Internal = {
    __has_assertion_failure = false
    __has_buf_overflow = true
    __has_overflow = false
    __has_returned = false
    __reverted = false
    msg = {
      "gas": 0,
```

```
        "sender": 0,
        "value": 0
      }
  }
  Other = {
      block = {
        "number": 0,
        "timestamp": 0
      }
  }
  Address_Map = [
    {
      "key": 0,
      "value": {
        "contract_name": "BundleSandSale",
        "balance": 0,
        "contract": {
          "ERC1155_RECEIVED": "\u00c1\u00c1\u00c1\u00c1",
          "ERC1155_BATCH_RECEIVED": "\u00c1\u00c1\u00c1\u00c1",
          "_medianizer": 0,
          "_dai": 0,
          "_sand": 0,
          "_asset": 0,
          "_receivingWallet": 0,
          "sales": [
            {
              "ids": [
                {
                  "key": "ALL_OTHERS",
                  "value": 128
                }
              ],
              "amounts": [
                {
                  "key": "ALL_OTHERS",
                  "value": 128
                }
              ],
              "sandAmount": 128,
              "priceUSD": 128,
              "numPacksLeft": 128
            },
            {
              "ids": [
                {
                  "key": "ALL_OTHERS",
                  "value": 128
                }
              ],
              "amounts": [
                {
                  "key": "ALL_OTHERS",
                  "value": 128
                }
              ],
              "sandAmount": 128,
              "priceUSD": 128,
              "numPacksLeft": 128
```

```
            },
            {
              "ids": [
                {
                  "key": "ALL_OTHERS",
                  "value": 128
                }
              ],
              "amounts": [
                {
                  "key": "ALL_OTHERS",
                  "value": 128
                }
              ],
              "sandAmount": 128,
              "priceUSD": 128,
              "numPacksLeft": 128
            },
            {
              "ids": [
                {
                  "key": "ALL_OTHERS",
                  "value": 128
                }
              ],
              "amounts": [
                {
                  "key": "ALL_OTHERS",
                  "value": 128
                }
              ],
              "sandAmount": 128,
              "priceUSD": 128,
              "numPacksLeft": 128
            }
          ],
          "_admin": 0
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]
```


Formal Verification Request 64

Method will not encounter an assertion failure.

10, Dec 2019
0.59 ms

Line 223 in File BundleSandSale.sol

```
223    //@CTK NO_ASF
```

Line 235-256 in File BundleSandSale.sol

```
235    function withdrawSale(uint256 saleId, address to) external onlyAdmin() {
236        require(saleId > 0, "invalid saleId");
237        uint256 saleIndex = saleId - 1;
238        uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
239        sales[saleIndex].numPacksLeft = 0;
240
241        uint256[] memory ids = sales[saleIndex].ids;
242        uint256[] memory amounts = sales[saleIndex].amounts;
243        uint256 numIds = ids.length;
244        /*@FIXME FAIL "withdrawSale_loop"
245          @inv i <= sales[saleId - 1].ids.length
246          @post forall j: uint256. (j >= 0 /\ j < sales[saleId - 1].ids.length) -> (__post.
               sales[saleId - 1].amounts[j] == sales[saleId - 1].amounts[j] * sales[saleId -
               1].numPacksLeft)
247          @post i == sales[saleId - 1].ids.length
248         */
249        for (uint256 i = 0; i < numIds; i++) {
250            amounts[i] = amounts[i].mul(numPacksLeft);
251        }
252        require(_sand.transferFrom(address(this), to, numPacksLeft.mul(sales[saleIndex].
               sandAmount)), "transfer fo Sand failed");
253        _asset.safeBatchTransferFrom(address(this), to, ids, amounts, "");
254    }
```

✅ The code meets the specification.

Formal Verification Request 65

withdrawSale_require

📅 10, Dec 2019
⏱ 3.05 ms

Line 224-228 in File BundleSandSale.sol

```
224    /*@CTK withdrawSale_require
225      @tag assume_completion
226      @post msg.sender == _admin
227      @post saleId > 0
228     */
```

Line 235-256 in File BundleSandSale.sol

```
235    function withdrawSale(uint256 saleId, address to) external onlyAdmin() {
236        require(saleId > 0, "invalid saleId");
237        uint256 saleIndex = saleId - 1;
238        uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
239        sales[saleIndex].numPacksLeft = 0;
240
241        uint256[] memory ids = sales[saleIndex].ids;
242        uint256[] memory amounts = sales[saleIndex].amounts;
243        uint256 numIds = ids.length;
244        /*@FIXME FAIL "withdrawSale_loop"
245          @inv i <= sales[saleId - 1].ids.length
246          @post forall j: uint256. (j >= 0 /\ j < sales[saleId - 1].ids.length) -> (__post.
               sales[saleId - 1].amounts[j] == sales[saleId - 1].amounts[j] * sales[saleId -
               1].numPacksLeft)
```

```
        @post i == sales[saleId - 1].ids.length
       */
      for (uint256 i = 0; i < numIds; i++) {
          amounts[i] = amounts[i].mul(numPacksLeft);
      }
      require(_sand.transferFrom(address(this), to, numPacksLeft.mul(sales[saleIndex].
          sandAmount)), "transfer fo Sand failed");
      _asset.safeBatchTransferFrom(address(this), to, ids, amounts, "");
  }
```

The code meets the specification.

Formal Verification Request 66

withdrawSale_change

10, Dec 2019
2.98 ms

Line 229-234 in File BundleSandSale.sol

```
  /*@CTK withdrawSale_change
    @tag assume_completion
    @pre msg.sender == _admin
    @pre saleId > 0
    @post __post.sales[saleId-1].numPacksLeft == 0
   */
```

Line 235-256 in File BundleSandSale.sol

```
  function withdrawSale(uint256 saleId, address to) external onlyAdmin() {
      require(saleId > 0, "invalid saleId");
      uint256 saleIndex = saleId - 1;
      uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
      sales[saleIndex].numPacksLeft = 0;

      uint256[] memory ids = sales[saleIndex].ids;
      uint256[] memory amounts = sales[saleIndex].amounts;
      uint256 numIds = ids.length;
      /*@FIXME FAIL "withdrawSale_loop"
        @inv i <= sales[saleId - 1].ids.length
        @post forall j: uint256. (j >= 0 /\ j < sales[saleId - 1].ids.length) -> (__post.
            sales[saleId - 1].amounts[j] == sales[saleId - 1].amounts[j] * sales[saleId -
            1].numPacksLeft)
        @post i == sales[saleId - 1].ids.length
       */
      for (uint256 i = 0; i < numIds; i++) {
          amounts[i] = amounts[i].mul(numPacksLeft);
      }
      require(_sand.transferFrom(address(this), to, numPacksLeft.mul(sales[saleIndex].
          sandAmount)), "transfer fo Sand failed");
      _asset.safeBatchTransferFrom(address(this), to, ids, amounts, "");
  }
```

The code meets the specification.

Formal Verification Request 67

If method completes, integer overflow would not happen.

10, Dec 2019
0.65 ms

Line 263 in File BundleSandSale.sol

```
263    //@CTK NO_OVERFLOW
```

Line 266-269 in File BundleSandSale.sol

```
266    function getEtherAmountWithUSD(uint256 usdAmount) public view returns (uint256) {
267        uint256 ethUsdPair = getEthUsdPair();
268        return usdAmount.mul(1000000000000000000).div(ethUsdPair);
269    }
```

The code meets the specification.

Formal Verification Request 68

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
0.68 ms

Line 264 in File BundleSandSale.sol

```
264    //@CTK NO_BUF_OVERFLOW
```

Line 266-269 in File BundleSandSale.sol

```
266    function getEtherAmountWithUSD(uint256 usdAmount) public view returns (uint256) {
267        uint256 ethUsdPair = getEthUsdPair();
268        return usdAmount.mul(1000000000000000000).div(ethUsdPair);
269    }
```

The code meets the specification.

Formal Verification Request 69

Method will not encounter an assertion failure.

10, Dec 2019
8.18 ms

Line 265 in File BundleSandSale.sol

```
265    //@CTK FAIL NO_ASF
```

Line 266-269 in File BundleSandSale.sol

```
266    function getEtherAmountWithUSD(uint256 usdAmount) public view returns (uint256) {
267        uint256 ethUsdPair = getEthUsdPair();
268        return usdAmount.mul(1000000000000000000).div(ethUsdPair);
269    }
```

This code violates the specification.

```
Counter Example:
Before Execution:
    Input = {
        usdAmount = 96
    }
    This = 0
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 0
        }
    }
    Other = {
        __return = 0
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": "ALL_OTHERS",
        "value": {
          "contract_name": "BundleSandSale",
          "balance": 0,
          "contract": {
            "ERC1155_RECEIVED": "AAAA",
            "ERC1155_BATCH_RECEIVED": "CCCC",
            "_medianizer": 0,
            "_dai": 0,
            "_sand": 0,
            "_asset": 0,
            "_receivingWallet": 0,
            "sales": [],
            "_admin": 0
          }
        }
      }
    ]

Function invocation is reverted.
```

Formal Verification Request 70

If method completes, integer overflow would not happen.

📅 10, Dec 2019
⏱ 11.64 ms

Line 11 in File TheSandbox712.sol

```
11    //@CTK NO_OVERFLOW
```

Line 18-27 in File TheSandbox712.sol

```
18    function init712() public phase("712") {
19        DOMAIN_SEPARATOR = keccak256(
20            abi.encode(
21                EIP712DOMAIN_TYPEHASH,
22                keccak256("The Sandbox 3D"),
23                keccak256("1"),
24                address(this)
25            )
26        );
27    }
```

✅ The code meets the specification.

Formal Verification Request 71

Buffer overflow / array index out of bound would never happen.

📅 10, Dec 2019
⏱ 0.37 ms

Line 12 in File TheSandbox712.sol

```
12    //@CTK NO_BUF_OVERFLOW
```

Line 18-27 in File TheSandbox712.sol

```
18    function init712() public phase("712") {
19        DOMAIN_SEPARATOR = keccak256(
20            abi.encode(
21                EIP712DOMAIN_TYPEHASH,
22                keccak256("The Sandbox 3D"),
23                keccak256("1"),
24                address(this)
25            )
26        );
27    }
```

✅ The code meets the specification.

Formal Verification Request 72

Method will not encounter an assertion failure.

📅 10, Dec 2019
⏱ 0.29 ms

Line 13 in File TheSandbox712.sol

```
13    //@CTK NO_ASF
```

Line 18-27 in File TheSandbox712.sol

```
18    function init712() public phase("712") {
19        DOMAIN_SEPARATOR = keccak256(
20            abi.encode(
21                EIP712DOMAIN_TYPEHASH,
22                keccak256("The Sandbox 3D"),
23                keccak256("1"),
24                address(this)
25            )
26        );
27    }
```

The code meets the specification.

Formal Verification Request 73

init712

10, Dec 2019
2.43 ms

Line 14-17 in File TheSandbox712.sol

```
14    /*@CTK init712
15      @tag assume_completion
16      @post __post._initialised["712"] == true
17     */
```

Line 18-27 in File TheSandbox712.sol

```
18    function init712() public phase("712") {
19        DOMAIN_SEPARATOR = keccak256(
20            abi.encode(
21                EIP712DOMAIN_TYPEHASH,
22                keccak256("The Sandbox 3D"),
23                keccak256("1"),
24                address(this)
25            )
26        );
27    }
```

The code meets the specification.

Formal Verification Request 74

If method completes, integer overflow would not happen.

10, Dec 2019
3.94 ms

Line 29 in File TheSandbox712.sol

```
29    //@CTK NO_OVERFLOW
```

Line 32-34 in File TheSandbox712.sol

```
32    function domainSeparator() internal view returns (bytes32) {
33        return DOMAIN_SEPARATOR;
34    }
```

The code meets the specification.

Formal Verification Request 75

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
0.33 ms

Line 30 in File TheSandbox712.sol

```
30    //@CTK NO_BUF_OVERFLOW
```

Line 32-34 in File TheSandbox712.sol

```
32    function domainSeparator() internal view returns (bytes32) {
33        return DOMAIN_SEPARATOR;
34    }
```

The code meets the specification.

Formal Verification Request 76

Method will not encounter an assertion failure.

10, Dec 2019
0.26 ms

Line 31 in File TheSandbox712.sol

```
31    //@CTK NO_ASF
```

Line 32-34 in File TheSandbox712.sol

```
32    function domainSeparator() internal view returns (bytes32) {
33        return DOMAIN_SEPARATOR;
34    }
```

The code meets the specification.

Formal Verification Request 77

If method completes, integer overflow would not happen.

10, Dec 2019
81.59 ms

Line 54 in File AssetSignedAuction.sol

```
54    //@CTK NO_OVERFLOW
```

Line 64-72 in File AssetSignedAuction.sol

```
64    constructor(Asset asset, address admin, address initialMetaTx, address payable
          feeCollector, uint256 fee10000th) public {
65        _asset = asset;
66        _feeCollector = feeCollector;
67        _fee10000th = fee10000th;
68        emit FeeSetup(feeCollector, fee10000th);
69        _admin = admin;
70        _setMetaTransactionProcessor(initialMetaTx, true);
71        init712();
72    }
```

The code meets the specification.

Formal Verification Request 78

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
0.75 ms

Line 55 in File AssetSignedAuction.sol

```
55    //@CTK NO_BUF_OVERFLOW
```

Line 64-72 in File AssetSignedAuction.sol

```
64    constructor(Asset asset, address admin, address initialMetaTx, address payable
          feeCollector, uint256 fee10000th) public {
65        _asset = asset;
66        _feeCollector = feeCollector;
67        _fee10000th = fee10000th;
68        emit FeeSetup(feeCollector, fee10000th);
69        _admin = admin;
70        _setMetaTransactionProcessor(initialMetaTx, true);
71        init712();
72    }
```

The code meets the specification.

Formal Verification Request 79

Method will not encounter an assertion failure.

10, Dec 2019
0.74 ms

Line 56 in File AssetSignedAuction.sol

```
56    //@CTK NO_ASF
```

Line 64-72 in File AssetSignedAuction.sol

```
64    constructor(Asset asset, address admin, address initialMetaTx, address payable
          feeCollector, uint256 fee10000th) public {
65        _asset = asset;
66        _feeCollector = feeCollector;
67        _fee10000th = fee10000th;
```

```
        emit FeeSetup(feeCollector, fee10000th);
        _admin = admin;
        _setMetaTransactionProcessor(initialMetaTx, true);
        init712();
    }
```

The code meets the specification.

Formal Verification Request 80

AssetSignedAuction

10, Dec 2019
20.15 ms

Line 57-63 in File AssetSignedAuction.sol

```
    /*@CTK AssetSignedAuction
      @post __post._asset == asset
      @post __post._feeCollector == feeCollector
      @post __post._fee10000th == fee10000th
      @post __post._admin == admin
      @post __post._metaTransactionContracts[initialMetaTx] == true
     */
```

Line 64-72 in File AssetSignedAuction.sol

```
    constructor(Asset asset, address admin, address initialMetaTx, address payable
        feeCollector, uint256 fee10000th) public {
        _asset = asset;
        _feeCollector = feeCollector;
        _fee10000th = fee10000th;
        emit FeeSetup(feeCollector, fee10000th);
        _admin = admin;
        _setMetaTransactionProcessor(initialMetaTx, true);
        init712();
    }
```

The code meets the specification.

Formal Verification Request 81

If method completes, integer overflow would not happen.

10, Dec 2019
18.7 ms

Line 77 in File AssetSignedAuction.sol

```
    //@CTK NO_OVERFLOW
```

Line 90-95 in File AssetSignedAuction.sol

```
    function setFee(address payable feeCollector, uint256 fee10000th) external {
        require(msg.sender == _admin, "only admin can change fee");
        _feeCollector = feeCollector;
        _fee10000th = fee10000th;
```

```
        emit FeeSetup(feeCollector, fee10000th);
    }
```

The code meets the specification.

Formal Verification Request 82

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
0.67 ms

Line 78 in File AssetSignedAuction.sol

```
78    //@CTK NO_BUF_OVERFLOW
```

Line 90-95 in File AssetSignedAuction.sol

```
90    function setFee(address payable feeCollector, uint256 fee10000th) external {
91        require(msg.sender == _admin, "only admin can change fee");
92        _feeCollector = feeCollector;
93        _fee10000th = fee10000th;
94        emit FeeSetup(feeCollector, fee10000th);
95    }
```

The code meets the specification.

Formal Verification Request 83

Method will not encounter an assertion failure.

10, Dec 2019
0.42 ms

Line 79 in File AssetSignedAuction.sol

```
79    //@CTK NO_ASF
```

Line 90-95 in File AssetSignedAuction.sol

```
90    function setFee(address payable feeCollector, uint256 fee10000th) external {
91        require(msg.sender == _admin, "only admin can change fee");
92        _feeCollector = feeCollector;
93        _fee10000th = fee10000th;
94        emit FeeSetup(feeCollector, fee10000th);
95    }
```

The code meets the specification.

Formal Verification Request 84

setFee_require

10, Dec 2019
1.08 ms

Line 80-83 in File AssetSignedAuction.sol

```
80    /*@CTK setFee_require
81      @tag assume_completion
82      @post msg.sender == _admin
83     */
```

Line 90-95 in File AssetSignedAuction.sol

```
90    function setFee(address payable feeCollector, uint256 fee10000th) external {
91        require(msg.sender == _admin, "only admin can change fee");
92        _feeCollector = feeCollector;
93        _fee10000th = fee10000th;
94        emit FeeSetup(feeCollector, fee10000th);
95    }
```

✅ The code meets the specification.

Formal Verification Request 85

setFee_change

📅 10, Dec 2019
⏱ 2.98 ms

Line 84-89 in File AssetSignedAuction.sol

```
84    /*@CTK setFee_change
85      @tag assume_completion
86      @pre msg.sender == _admin
87      @post __post._feeCollector == feeCollector
88      @post __post._fee10000th == fee10000th
89     */
```

Line 90-95 in File AssetSignedAuction.sol

```
90    function setFee(address payable feeCollector, uint256 fee10000th) external {
91        require(msg.sender == _admin, "only admin can change fee");
92        _feeCollector = feeCollector;
93        _fee10000th = fee10000th;
94        emit FeeSetup(feeCollector, fee10000th);
95    }
```

✅ The code meets the specification.

Formal Verification Request 86

If method completes, integer overflow would not happen.

📅 10, Dec 2019
⏱ 152.48 ms

Line 97 in File AssetSignedAuction.sol

```
97    //@CTK NO_OVERFLOW
```

Line 109-135 in File AssetSignedAuction.sol

```
109    function _verifyParameters(
110        address buyer,
111        address payable seller,
112        address token,
113        uint256 buyAmount,
114        uint256[] memory auctionData,
115        uint256[] memory ids,
116        uint256[] memory amounts
117    ) internal view {
118        require(ids.length == amounts.length, "ids and amounts length not matching");
119        require(buyer == msg.sender || (token != address(0) && _metaTransactionContracts[msg
               .sender]), "not authorized");
120        uint256 amountAlreadyClaimed = claimed[seller][auctionData[AuctionData_OfferId]];
121        require(amountAlreadyClaimed != MAX_UINT256, "Auction cancelled");
122
123        uint256 total = amountAlreadyClaimed.add(buyAmount);
124        require(total >= amountAlreadyClaimed, "overflow");
125        require(total <= auctionData[AuctionData_Packs], "Buy amount exceeds sell amount");
126
127        require(
128            auctionData[AuctionData_StartedAt] <= block.timestamp,
129            "Auction didn't start yet"
130        );
131        require(
132            auctionData[AuctionData_StartedAt].add(auctionData[AuctionData_Duration]) > block
                   .timestamp,
133            "Auction finished"
134        );
135    }
```

✅ The code meets the specification.

Formal Verification Request 87

Buffer overflow / array index out of bound would never happen.

📅 10, Dec 2019
⏱ 50.48 ms

Line 98 in File AssetSignedAuction.sol

```
98    //@CTK FAIL NO_BUF_OVERFLOW
```

Line 109-135 in File AssetSignedAuction.sol

```
109    function _verifyParameters(
110        address buyer,
111        address payable seller,
112        address token,
113        uint256 buyAmount,
114        uint256[] memory auctionData,
115        uint256[] memory ids,
116        uint256[] memory amounts
117    ) internal view {
118        require(ids.length == amounts.length, "ids and amounts length not matching");
119        require(buyer == msg.sender || (token != address(0) && _metaTransactionContracts[msg
               .sender]), "not authorized");
120        uint256 amountAlreadyClaimed = claimed[seller][auctionData[AuctionData_OfferId]];
```

```
        require(amountAlreadyClaimed != MAX_UINT256, "Auction cancelled");

        uint256 total = amountAlreadyClaimed.add(buyAmount);
        require(total >= amountAlreadyClaimed, "overflow");
        require(total <= auctionData[AuctionData_Packs], "Buy amount exceeds sell amount");

        require(
            auctionData[AuctionData_StartedAt] <= block.timestamp,
            "Auction didn't start yet"
        );
        require(
            auctionData[AuctionData_StartedAt].add(auctionData[AuctionData_Duration]) > block
                .timestamp,
            "Auction finished"
        );
    }
```

This code violates the specification.

```
Counter Example:
Before Execution:
    Input = {
        amounts = []
        auctionData = [
          0
        ]
        buyAmount = 0
        buyer = 0
        ids = []
        seller = 0
        token = 64
    }
    This = 0
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 0
        }
    }
    Other = {
        block = {
          "number": 0,
          "timestamp": 32
        }
    }
    Address_Map = [
      {
        "key": "ALL_OTHERS",
        "value": {
          "contract_name": "AssetSignedAuction",
          "balance": 0,
          "contract": {
            "AUCTION_TYPEHASH": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
```

```
          "MAX_UINT256": 1,
          "AuctionData_OfferId": 128,
          "AuctionData_StartingPrice": 0,
          "AuctionData_EndingPrice": 0,
          "AuctionData_StartedAt": 0,
          "AuctionData_Duration": 0,
          "AuctionData_Packs": 0,
          "claimed": [
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": 0
                }
              ]
            }
          ],
          "_asset": 0,
          "_fee10000th": 0,
          "_feeCollector": 0,
          "_metaTransactionContracts": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "_admin": 0,
          "EIP712DOMAIN_TYPEHASH": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
          "DOMAIN_SEPARATOR": "IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII",
          "_initialised": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "ERC1271_MAGICVALUE": "AAAA",
          "ERC1654_MAGICVALUE": "AAAA"
        }
      }
    }
  ]

After Execution:
  Input = {
    amounts = []
    auctionData = [
      0
    ]
    buyAmount = 0
    buyer = 0
    ids = []
    seller = 0
    token = 64
  }
  This = 0
  Internal = {
    __has_assertion_failure = false
```

```
      __has_buf_overflow = true
      __has_overflow = false
      __has_returned = false
      __reverted = false
      msg = {
        "gas": 0,
        "sender": 0,
        "value": 0
      }
    }
    Other = {
      block = {
        "number": 0,
        "timestamp": 32
      }
    }
    Address_Map = [
      {
      "key": "ALL_OTHERS",
      "value": {
        "contract_name": "AssetSignedAuction",
        "balance": 0,
        "contract": {
          "AUCTION_TYPEHASH": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
          "MAX_UINT256": 1,
          "AuctionData_OfferId": 128,
          "AuctionData_StartingPrice": 0,
          "AuctionData_EndingPrice": 0,
          "AuctionData_StartedAt": 0,
          "AuctionData_Duration": 0,
          "AuctionData_Packs": 0,
          "claimed": [
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": 0
                }
              ]
            }
          ],
          "_asset": 0,
          "_fee10000th": 0,
          "_feeCollector": 0,
          "_metaTransactionContracts": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "_admin": 0,
          "EIP712DOMAIN_TYPEHASH": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
          "DOMAIN_SEPARATOR": "IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII",
          "_initialised": [
            {
              "key": "ALL_OTHERS",
              "value": false
```

```
            }
          ],
          "ERC1271_MAGICVALUE": "AAAA",
          "ERC1654_MAGICVALUE": "AAAA"
        }
      }
    }
  ]
```

Formal Verification Request 88

Method will not encounter an assertion failure.

10, Dec 2019
25.66 ms

Line 99 in File AssetSignedAuction.sol

```
99    //@CTK NO_ASF
```

Line 109-135 in File AssetSignedAuction.sol

```
109   function _verifyParameters(
110       address buyer,
111       address payable seller,
112       address token,
113       uint256 buyAmount,
114       uint256[] memory auctionData,
115       uint256[] memory ids,
116       uint256[] memory amounts
117   ) internal view {
118       require(ids.length == amounts.length, "ids and amounts length not matching");
119       require(buyer == msg.sender || (token != address(0) && _metaTransactionContracts[msg
              .sender]), "not authorized");
120       uint256 amountAlreadyClaimed = claimed[seller][auctionData[AuctionData_OfferId]];
121       require(amountAlreadyClaimed != MAX_UINT256, "Auction cancelled");
122
123       uint256 total = amountAlreadyClaimed.add(buyAmount);
124       require(total >= amountAlreadyClaimed, "overflow");
125       require(total <= auctionData[AuctionData_Packs], "Buy amount exceeds sell amount");
126
127       require(
128           auctionData[AuctionData_StartedAt] <= block.timestamp,
129           "Auction didn't start yet"
130       );
131       require(
132           auctionData[AuctionData_StartedAt].add(auctionData[AuctionData_Duration]) > block
                  .timestamp,
133           "Auction finished"
134       );
135   }
```

The code meets the specification.

Formal Verification Request 89

_verifyParameters

10, Dec 2019
128.26 ms

Line 100-108 in File AssetSignedAuction.sol

```
100 /*@CTK _verifyParameters
101   @tag assume_completion
102   @post ids.length == amounts.length
103   @post buyer == msg.sender \/ (token != address(0) /\ _metaTransactionContracts[msg.
        sender])
104   @post claimed[seller][auctionData[AuctionData_OfferId]] != MAX_UINT256
105   @post claimed[seller][auctionData[AuctionData_OfferId]] + buyAmount <= auctionData[
        AuctionData_Packs]
106   @post auctionData[AuctionData_StartedAt] <= block.timestamp
107   @post auctionData[AuctionData_StartedAt] + auctionData[AuctionData_Duration] > block.
        timestamp
108  */
```

Line 109-135 in File AssetSignedAuction.sol

```
109 function _verifyParameters(
110     address buyer,
111     address payable seller,
112     address token,
113     uint256 buyAmount,
114     uint256[] memory auctionData,
115     uint256[] memory ids,
116     uint256[] memory amounts
117 ) internal view {
118     require(ids.length == amounts.length, "ids and amounts length not matching");
119     require(buyer == msg.sender || (token != address(0) && _metaTransactionContracts[msg
          .sender]), "not authorized");
120     uint256 amountAlreadyClaimed = claimed[seller][auctionData[AuctionData_OfferId]];
121     require(amountAlreadyClaimed != MAX_UINT256, "Auction cancelled");
122
123     uint256 total = amountAlreadyClaimed.add(buyAmount);
124     require(total >= amountAlreadyClaimed, "overflow");
125     require(total <= auctionData[AuctionData_Packs], "Buy amount exceeds sell amount");
126
127     require(
128         auctionData[AuctionData_StartedAt] <= block.timestamp,
129         "Auction didn't start yet"
130     );
131     require(
132         auctionData[AuctionData_StartedAt].add(auctionData[AuctionData_Duration]) > block
              .timestamp,
133         "Auction finished"
134     );
135 }
```

The code meets the specification.

Formal Verification Request 90

If method completes, integer overflow would not happen.

10, Dec 2019
9.4 ms

Line 448 in File AssetSignedAuction.sol

```
448    //@CTK NO_OVERFLOW
```

Line 454-457 in File AssetSignedAuction.sol

```
454    function cancelSellerOffer(uint256 offerId) external {
455        claimed[msg.sender][offerId] = MAX_UINT256;
456        emit OfferCancelled(msg.sender, offerId);
457    }
```

The code meets the specification.

Formal Verification Request 91

Buffer overflow / array index out of bound would never happen.

10, Dec 2019
0.29 ms

Line 449 in File AssetSignedAuction.sol

```
449    //@CTK NO_BUF_OVERFLOW
```

Line 454-457 in File AssetSignedAuction.sol

```
454    function cancelSellerOffer(uint256 offerId) external {
455        claimed[msg.sender][offerId] = MAX_UINT256;
456        emit OfferCancelled(msg.sender, offerId);
457    }
```

The code meets the specification.

Formal Verification Request 92

Method will not encounter an assertion failure.

10, Dec 2019
0.28 ms

Line 450 in File AssetSignedAuction.sol

```
450    //@CTK NO_ASF
```

Line 454-457 in File AssetSignedAuction.sol

```
454    function cancelSellerOffer(uint256 offerId) external {
455        claimed[msg.sender][offerId] = MAX_UINT256;
456        emit OfferCancelled(msg.sender, offerId);
457    }
```

The code meets the specification.

Formal Verification Request 93

cancelSellerOffer

10, Dec 2019
1.9 ms

Line 451-453 in File AssetSignedAuction.sol

```
451     /*@CTK cancelSellerOffer
452       @post __post.claimed[msg.sender][offerId] == MAX_UINT256
453      */
```

Line 454-457 in File AssetSignedAuction.sol

```
454     function cancelSellerOffer(uint256 offerId) external {
455         claimed[msg.sender][offerId] = MAX_UINT256;
456         emit OfferCancelled(msg.sender, offerId);
457     }
```

The code meets the specification.

Formal Verification Request 94

__executeDeal__loop___Generated

10, Dec 2019
471.5 ms

(Loop) Line 425-431 in File AssetSignedAuction.sol

```
425         /*@CTK FAIL "_executeDeal_loop"
426           @tag assume_completion
427           @inv i <= packAmounts.length
428           @post forall j: uint256. (j >= 0 /\ j < packAmounts.length) -> packAmounts[j] ==
                  amounts[j] * purchase[0]
429           @post i == packAmounts.length
430           @post !__should_return
431          */
```

(Loop) Line 425-434 in File AssetSignedAuction.sol

```
425         /*@CTK FAIL "_executeDeal_loop"
426           @tag assume_completion
427           @inv i <= packAmounts.length
428           @post forall j: uint256. (j >= 0 /\ j < packAmounts.length) -> packAmounts[j] ==
                  amounts[j] * purchase[0]
429           @post i == packAmounts.length
430           @post !__should_return
431          */
432         for (uint256 i = 0; i < packAmounts.length; i++) {
433             packAmounts[i] = amounts[i].mul(purchase[0]);
434         }
```

This code violates the specification.

```
1 Counter Example:
2 Before Execution:
3     Input = {
```

```
        __should_break = false
        __should_break__pre = false
        __should_return = false
        __should_return__pre = false
        amounts = [
          1,
          1,
          1,
          1,
          1,
          1,
          1,
          1
        ]
        amounts__pre = []
        i = 1
        i__pre = 0
        packAmounts = [
          0
        ]
        packAmounts__pre = []
        purchase = []
        purchase__pre = []
        this__pre = 0
    }
    This = 0
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = false
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 0
        }
    }
    Other = {
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": 0,
        "value": {
          "contract_name": "AssetSignedAuction",
          "balance": 0,
          "contract": {
            "AUCTION_TYPEHASH": "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
            "MAX_UINT256": 0,
            "AuctionData_OfferId": 0,
            "AuctionData_StartingPrice": 0,
            "AuctionData_EndingPrice": 0,
            "AuctionData_StartedAt": 0,
            "AuctionData_Duration": 0,
```

```
          "AuctionData_Packs": 0,
          "claimed": [
            {
              "key": "ALL_OTHERS",
              "value": [
                {
                  "key": "ALL_OTHERS",
                  "value": 1
                }
              ]
            }
          ],
          "_asset": 0,
          "_fee10000th": 0,
          "_feeCollector": 0,
          "_metaTransactionContracts": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "_admin": 0,
          "EIP712DOMAIN_TYPEHASH": "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
          "DOMAIN_SEPARATOR": "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
          "_initialised": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "ERC1271_MAGICVALUE": "BBBB",
          "ERC1654_MAGICVALUE": "BBBB"
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]

After Execution:
  Input = {
      __should_break = true
      __should_break__pre = false
      __should_return = false
      __should_return__pre = false
      amounts = [
        1,
        1,
        1,
        1,
        1,
        1,
        1,
        1
      ]
      amounts__pre = []
```

```
        i = 1
        i__pre = 0
        packAmounts = [
          0
        ]
        packAmounts__pre = []
        purchase = []
        purchase__pre = []
        this__pre = 0
    }
    This = 0
    Internal = {
        __has_assertion_failure = false
        __has_buf_overflow = false
        __has_overflow = false
        __has_returned = true
        __reverted = false
        msg = {
          "gas": 0,
          "sender": 0,
          "value": 0
        }
    }
    Other = {
        block = {
          "number": 0,
          "timestamp": 0
        }
    }
    Address_Map = [
      {
        "key": 0,
        "value": {
          "contract_name": "AssetSignedAuction",
          "balance": 0,
          "contract": {
            "AUCTION_TYPEHASH": "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
            "MAX_UINT256": 0,
            "AuctionData_OfferId": 0,
            "AuctionData_StartingPrice": 0,
            "AuctionData_EndingPrice": 0,
            "AuctionData_StartedAt": 0,
            "AuctionData_Duration": 0,
            "AuctionData_Packs": 0,
            "claimed": [
              {
                "key": "ALL_OTHERS",
                "value": [
                  {
                    "key": "ALL_OTHERS",
                    "value": 1
                  }
                ]
              }
            ],
            "_asset": 0,
            "_fee10000th": 0,
            "_feeCollector": 0,
```

```
          "_metaTransactionContracts": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "_admin": 0,
          "EIP712DOMAIN_TYPEHASH": "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
          "DOMAIN_SEPARATOR": "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
          "_initialised": [
            {
              "key": "ALL_OTHERS",
              "value": false
            }
          ],
          "ERC1271_MAGICVALUE": "BBBB",
          "ERC1654_MAGICVALUE": "BBBB"
        }
      }
    },
    {
      "key": "ALL_OTHERS",
      "value": "EmptyAddress"
    }
  ]
```

Source Code with CertiK Labels

File SafeMathWithRequire.sol

```
pragma solidity ^0.5.2;

/**
 * @title SafeMath
 * @dev Math operations with safety checks that revert
 */
library SafeMathWithRequire {
    /**
    * @dev Multiplies two numbers, throws on overflow.
    */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK mul
      @tag assume_completion
      @post __return == a * b
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than asserting 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "overflow");
        return c;
    }

    /**
    * @dev Integer division of two numbers, truncating the quotient.
    */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK FAIL NO_ASF
    /*@CTK div
      @tag assume_completion
      @post __return == a / b
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // assert(b > 0); // Solidity automatically throws when dividing by 0
        // uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold
        return a / b;
    }

    /**
    * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than
        minuend).
    */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
```

```
    /*@CTK sub
      @tag assume_completion
      @post __return == a - b
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "undeflow");
        return a - b;
    }

    /**
    * @dev Adds two numbers, throws on overflow.
    */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK add
      @tag assume_completion
      @post __return == a + b
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "overflow");
        return c;
    }
}
```

File CommonMinter.sol

```
pragma solidity 0.5.9;

import "../sandbox-private-contracts/src/Asset/ERC1155ERC721.sol";
import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC20.sol";
import "../sandbox-private-contracts/contracts_common/src/BaseWithStorage/
    MetaTransactionReceiver.sol";
import "../sandbox-private-contracts/contracts_common/src/Libraries/SafeMathWithRequire.sol
    ";

contract CommonMinter is MetaTransactionReceiver {
    using SafeMathWithRequire for uint256;

    uint256 _feePerCopy;

    ERC1155ERC721 _asset;
    mapping(address => bool) _minters;
    address _feeReceiver;
    ERC20 _sand;

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK CommonMinter
      @tag assume_completion
      @post __post._sand == sand
      @post __post._asset == asset
      @post __post._feePerCopy == feePerCopy
      @post __post._admin == admin
      @post __post._feeReceiver == feeReceiver
      @post __post._metaTransactionContracts[address(sand)] == true
     */
```

```
    constructor(ERC1155ERC721 asset, ERC20 sand, uint256 feePerCopy, address admin, address feeReceiver)
        public
    {
        _sand = sand;
        _asset = asset;
        _feePerCopy = feePerCopy;
        _admin = admin;
        _feeReceiver = feeReceiver;
        _setMetaTransactionProcessor(address(sand), true);
    }

    /// @notice set the receiver of the proceeds
    /// @param newFeeReceiver address of the new fee receiver
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK setFeeReceiver_change
      @tag assume_completion
      @post msg.sender == _admin
     */
    /*@CTK setFeeReceiver_change
      @tag assume_completion
      @pre msg.sender == _admin
      @post __post._feeReceiver == newFeeReceiver
     */
    function setFeeReceiver(address newFeeReceiver) external {
        require(msg.sender == _admin, "only admin can change the receiver");
        _feeReceiver = newFeeReceiver;
    }

    /// @notice set the fee in Sand for each common Asset copies
    /// @param newFee new fee in Sand
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK setFeePerCopy_change
      @tag assume_completion
      @post msg.sender == _admin
     */
    /*@CTK setFeePerCopy_change
      @tag assume_completion
      @pre msg.sender == _admin
      @post __post._feePerCopy == newFee
     */
    function setFeePerCopy(uint256 newFee) external {
        require(msg.sender == _admin, "only admin allowed to set fee");
        _feePerCopy = newFee;
    }

    /// @notice mint common Asset token by paying the Sand fee
    /// @param creator address creating the Asset, need to be the tx sender or meta tx signer
    /// @param packId unused packId that will let you predict the resulting tokenId
    /// @param hash cidv1 ipfs hash of the folder where 0.json file contains the metadata
    /// @param supply number of copies to mint, cost in Sand is relative it it
    /// @param owner address receiving the minted tokens
    /// @param data extra data
```

```
    /// @param feePerCopy fee in Sand for each copies
    //@FIXME NO_OVERFLOW
    //@FIXME NO_BUF_OVERFLOW
    //@FIXME NO_ASF
    /*@FIXME mintFor_require
      @tag assume_completion
      @post creator == msg.sender \/ _metaTransactionContracts[msg.sender] == true
      @post feePerCopy == _feePerCopy
     */
    function mintFor(
        address creator,
        uint40 packId,
        bytes32 hash,
        uint32 supply,
        address owner,
        bytes calldata data,
        uint256 feePerCopy
    ) external returns (uint256) {
        require(creator == msg.sender || _metaTransactionContracts[msg.sender], "not
            authorized");
        require(feePerCopy == _feePerCopy, "invalid fee");
        require(_sand.transferFrom(creator, _feeReceiver, uint256(supply).mul(feePerCopy)),
            "failed to transfer SAND");
        return _asset.mint(creator, packId, hash, supply, 0, owner, data);
    }

    /// @notice mint multiple common Asset tokena by paying the Sand fee
    /// @param creator address creating the Asset, need to be the tx sender or meta tx
        signer
    /// @param packId unused packId that will let you predict the resulting tokenId
    /// @param hash cidv1 ipfs hash of the folder where 0.json file contains the metadata
    /// @param supplies number of copies to mint for each Asset, cost in Sand is relative it
         it
    /// @param owner address receiving the minted tokens
    /// @param data extra data
    /// @param feePerCopy fee in Sand for each copies
    //@FIXME NO_OVERFLOW
    //@FIXME NO_BUF_OVERFLOW
    //@FIXME NO_ASF
    /*@FIXME mintFor_require
      @tag assume_completion
      @post creator == msg.sender \/ _metaTransactionContracts[msg.sender] == true
      @post feePerCopy == _feePerCopy
     */
    function mintMultipleFor(
        address creator,
        uint40 packId,
        bytes32 hash,
        uint256[] calldata supplies,
        address owner,
        bytes calldata data,
        uint256 feePerCopy
    ) external returns (uint256[] memory ids) {
        require(creator == msg.sender || _metaTransactionContracts[msg.sender], "not
            authorized");
        require(feePerCopy == _feePerCopy, "invalid fee");
        uint256 totalCopies = 0;
        uint256 numAssetTypes = supplies.length;
```

```
        for (uint256 i = 0; i < numAssetTypes; i++) {
            totalCopies = totalCopies.add(supplies[i]);
        }
        require(_sand.transferFrom(creator, _feeReceiver, totalCopies.mul(feePerCopy)), "
            failed to transfer SAND");
        return
            _asset.mintMultiple(
                creator,
                packId,
                hash,
                supplies,
                "",
                owner,
                data
            );
    }
}
```

File Asset.sol

```
pragma solidity 0.5.9;

import "../sandbox-private-contracts/src/Asset/ERC1155ERC721.sol";

contract Asset is ERC1155ERC721 {

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK Asset
      @tag assume_completion
      @post __post._metaTransactionContracts[metaTransactionContract] == true
      @post __post._admin == assetAdmin
      @post __post._bouncerAdmin == bouncerAdmin
     */
    constructor(
        address metaTransactionContract,
        address assetAdmin,
        address bouncerAdmin
    ) public ERC1155ERC721(metaTransactionContract, assetAdmin, bouncerAdmin) {}
}
```

File BundleSandSale.sol

```
pragma solidity 0.5.9;

import "../sandbox-private-contracts/contracts_common/src/Libraries/SafeMathWithRequire.sol
    ";
import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC20.sol";
import "../sandbox-private-contracts/contracts_common/src/Interfaces/Medianizer.sol";
import "../sandbox-private-contracts/contracts_common/src/BaseWithStorage/Admin.sol";
import "../sandbox-private-contracts/src/Asset/ERC1155ERC721.sol";


contract BundleSandSale is Admin {

    bytes4 private constant ERC1155_RECEIVED = 0xf23a6e61;
    bytes4 private constant ERC1155_BATCH_RECEIVED = 0xbc197c81;

    event BundleSale(
```

```
        uint256 indexed saleId,
        uint256[] ids,
        uint256[] amounts,
        uint256 sandAmount,
        uint256 priceUSD,
        uint256 numPacks
    );

    event BundleSold(
        uint256 indexed saleId,
        address indexed buyer,
        uint256 numPacks,
        address token,
        uint256 tokenAmount
    );

    using SafeMathWithRequire for uint256;

    Medianizer private _medianizer;
    ERC20 private _dai;
    ERC20 private _sand;
    ERC1155ERC721 private _asset;

    address payable private _receivingWallet;

    struct Sale {
        uint256[] ids;
        uint256[] amounts;
        uint256 sandAmount;
        uint256 priceUSD;
        uint256 numPacksLeft;
    }

    Sale[] private sales;

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK BundleSandSale_require
      @tag assume_completion
      @post receivingWallet != address(0)
     */
    /*@CTK BundleSandSale_change
      @tag assume_completion
      @post __post._medianizer == medianizerContractAddress
      @post __post._sand == sandTokenContractAddress
      @post __post._asset == assetTokenContractAddress
      @post __post._dai == daiTokenContractAddress
      @post __post._admin == admin
      @post __post._receivingWallet == receivingWallet
     */
    constructor(
        address sandTokenContractAddress,
        address assetTokenContractAddress,
        address medianizerContractAddress,
        address daiTokenContractAddress,
        address admin,
        address payable receivingWallet
```

```
    ) public {
        require(receivingWallet != address(0), "need a wallet to receive funds");
        _medianizer = Medianizer(medianizerContractAddress);
        _sand = ERC20(sandTokenContractAddress);
        _asset = ERC1155ERC721(assetTokenContractAddress);
        _dai = ERC20(daiTokenContractAddress);
        _admin = admin;
        _receivingWallet = receivingWallet;
    }

    /// @notice set the wallet receiving the proceeds
    /// @param newWallet address of the new receiving wallet
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK setReceivingWallet_require
      @tag assume_completion
      @post newWallet != address(0)
      @post msg.sender == _admin
     */
    /*@CTK setReceivingWallet_change
      @tag assume_completion
      @pre newWallet != address(0)
      @pre msg.sender == _admin
      @post __post._receivingWallet == newWallet
     */
    function setReceivingWallet(address payable newWallet) external {
        require(newWallet != address(0), "receiving wallet cannot be zero address");
        require(msg.sender == _admin, "only admin can change the receiving wallet");
        _receivingWallet = newWallet;
    }

    //@CTK NO_OVERFLOW
    //@CTK FAIL NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK _transferPack
      @tag assume_completion
     */
    function _transferPack(uint256 saleIndex, uint256 numPacks, address to) internal {
        uint256 sandAmountPerPack = sales[saleIndex].sandAmount;
        require(
            _sand.transferFrom(address(this), to, sandAmountPerPack.mul(numPacks)),
            "Sand Transfer failed"
        );
        uint256[] memory ids = sales[saleIndex].ids;
        uint256[] memory amounts = sales[saleIndex].amounts;
        uint256 numIds = ids.length;
        /*@FIXME FAIL "_transferPack_loop"
          @inv i <= sales[saleIndex].ids.length
          @post forall j: uint256. (j >= 0 /\ j < sales[saleIndex].ids.length) -> (__post.
              sales[saleIndex].amounts[j] == sales[saleIndex].amounts[j] * numPacks)
          @post i == sales[saleIndex].ids.length
         */
        for (uint256 i = 0; i < numIds; i++) {
            amounts[i] = amounts[i].mul(numPacks);
        }
        _asset.safeBatchTransferFrom(address(this), to, ids, amounts, "");
    }
```

```

    /**
     * @notice Buys Sand Bundle with Ether
     * @param saleId id of the bundle
     * @param numPacks the amount of packs to buy
     * @param to The address that will receive the SAND
     */
    //@CTK NO_OVERFLOW
    //@CTK FAIL NO_BUF_OVERFLOW
    //@CTK FAIL NO_ASF
    /*@CTK buyBundleWithEther_require
      @tag assume_completion
      @post saleId > 0
      @post sales[saleId - 1].numPacksLeft >= numPacks
     */
    function buyBundleWithEther(uint256 saleId, uint256 numPacks, address to) external
        payable {
        require(saleId > 0, "invalid saleId");
        uint256 saleIndex = saleId - 1;
        uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
        require(numPacksLeft >= numPacks, "not enough packs on sale");
        sales[saleIndex].numPacksLeft = numPacksLeft - numPacks;

        uint256 USDRequired = numPacks.mul(sales[saleIndex].priceUSD);
        uint256 ETHRequired = getEtherAmountWithUSD(USDRequired);
        require(msg.value >= ETHRequired, "not enough ether sent");
        uint256 leftOver = msg.value - ETHRequired;
        if(leftOver > 0) {
            msg.sender.transfer(leftOver); // refund extra
        }
        address(_receivingWallet).transfer(ETHRequired);
        _transferPack(saleIndex, numPacks, to);
        emit BundleSold(saleId, msg.sender, numPacks, address(0), ETHRequired);
    }

    /**
     * @notice Buys Sand Bundle with DAI
     * @param saleId id of the bundle
     * @param numPacks the amount of packs to buy
     * @param to The address that will receive the SAND
     */
    //@CTK NO_OVERFLOW
    //@CTK FAIL NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK buyBundleWithDai_require
      @tag assume_completion
      @post saleId > 0
      @post sales[saleId - 1].numPacksLeft >= numPacks
     */
    function buyBundleWithDai(uint256 saleId, uint256 numPacks, address to) external {
        require(saleId > 0, "invalid saleId");
        uint256 saleIndex = saleId - 1;
        uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
        require(numPacksLeft >= numPacks, "not enough packs on sale");
        sales[saleIndex].numPacksLeft = numPacksLeft - numPacks;

        uint256 USDRequired = numPacks.mul(sales[saleIndex].priceUSD);
        require(_dai.transferFrom(msg.sender, _receivingWallet, USDRequired), "failed to
```

```
        transfer dai");
    _transferPack(saleIndex, numPacks, to);

    emit BundleSold(saleId, msg.sender, numPacks, address(_dai), USDRequired);
}

//@CTK NO_OVERFLOW
//@CTK FAIL NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK getSaleInfo_require
  @tag assume_completion
  @post saleId > 0
 */
/*@CTK getSaleInfo_change
  @tag assume_completion
  @pre saleId > 0
  @post priceUSD == sales[saleId - 1].priceUSD
  @post numPacksLeft == sales[saleId - 1].numPacksLeft
 */
function getSaleInfo(uint256 saleId) external view returns(uint256 priceUSD, uint256
    numPacksLeft) {
    require(saleId > 0, "invalid saleId");
    uint256 saleIndex = saleId - 1;
    priceUSD = sales[saleIndex].priceUSD;
    numPacksLeft = sales[saleIndex].numPacksLeft;
}

//@CTK NO_OVERFLOW
//@CTK FAIL NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK withdrawSale_require
  @tag assume_completion
  @post msg.sender == _admin
  @post saleId > 0
 */
/*@CTK withdrawSale_change
  @tag assume_completion
  @pre msg.sender == _admin
  @pre saleId > 0
  @post __post.sales[saleId-1].numPacksLeft == 0
 */
function withdrawSale(uint256 saleId, address to) external onlyAdmin() {
    require(saleId > 0, "invalid saleId");
    uint256 saleIndex = saleId - 1;
    uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
    sales[saleIndex].numPacksLeft = 0;

    uint256[] memory ids = sales[saleIndex].ids;
    uint256[] memory amounts = sales[saleIndex].amounts;
    uint256 numIds = ids.length;
    /*@FIXME FAIL "withdrawSale_loop"
      @inv i <= sales[saleId - 1].ids.length
      @post forall j: uint256. (j >= 0 /\ j < sales[saleId - 1].ids.length) -> (__post.
          sales[saleId - 1].amounts[j] == sales[saleId - 1].amounts[j] * sales[saleId -
          1].numPacksLeft)
      @post i == sales[saleId - 1].ids.length
     */
    for (uint256 i = 0; i < numIds; i++) {
```

```
            amounts[i] = amounts[i].mul(numPacksLeft);
        }
        require(_sand.transferFrom(address(this), to, numPacksLeft.mul(sales[saleIndex].
            sandAmount)), "transfer fo Sand failed");
        _asset.safeBatchTransferFrom(address(this), to, ids, amounts, "");
    }

    /**
     * @notice Returns the amount of ETH for a specific amount of USD
     * @param usdAmount An amount of USD
     * @return The amount of ETH
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK FAIL NO_ASF
    function getEtherAmountWithUSD(uint256 usdAmount) public view returns (uint256) {
        uint256 ethUsdPair = getEthUsdPair();
        return usdAmount.mul(1000000000000000000).div(ethUsdPair);
    }

    /**
     * @notice Gets the ETHUSD pair from the Medianizer contract
     * @return The pair as an uint256
     */
    function getEthUsdPair() internal view returns (uint256) {
        bytes32 pair = _medianizer.read();
        return uint256(pair);
    }

    function onERC1155Received(
        address operator,
        address from,
        uint256 id,
        uint256 value,
        bytes calldata data
    ) external returns (bytes4) {
        require(
            address(_asset) == msg.sender,
            "only accept asset as sender"
        );
        require(from == operator, "only self executed transfer allowed");
        require(value > 0, "no Asset transfered");
        require(data.length > 0, "data need to contains the sale data");

        (
            uint256 numPacks,
            uint256 sandAmountPerPack,
            uint256 priceUSDPerPack
        ) = abi.decode(data, (uint256, uint256, uint256));

        uint256 amount = value.div(numPacks);
        require(amount.mul(numPacks) == value, "invalid amounts, not divisible by numPacks")
            ;
        uint256[] memory amounts = new uint256[](1);
        amounts[0] = amount;
        uint256[] memory ids = new uint256[](1);
        ids[0] = id;
        _setupBundle(from, sandAmountPerPack, numPacks, ids, amounts, priceUSDPerPack);
```

```
        return ERC1155_RECEIVED;
    }
    function onERC1155BatchReceived(
        address operator,
        address from,
        uint256[] calldata ids,
        uint256[] calldata values,
        bytes calldata data
    ) external returns (bytes4) {
        require(
            address(_asset) == msg.sender,
            "only accept asset as sender"
        );
        require(from == operator, "only self executed transfer allowed");
        require(ids.length > 0, "need to contains Asset");
        require(data.length > 0, "data need to contains the sale data");

        (
            uint256 numPacks,
            uint256 sandAmountPerPack,
            uint256 priceUSDPerPack
        ) = abi.decode(data, (uint256, uint256, uint256));

        uint256[] memory amounts = new uint256[](ids.length); // TODO
        for(uint256 i = 0; i < amounts.length; i ++) {
            require(values[i] > 0, "asset transfer with zero values");
            uint256 amount = values[i].div(numPacks);
            require(amount.mul(numPacks) == values[i], "invalid amounts, not divisible by
                numPacks");
            amounts[i] = amount;
        }

        _setupBundle(from, sandAmountPerPack, numPacks, ids, amounts, priceUSDPerPack);
        return ERC1155_BATCH_RECEIVED;
    }
    function _setupBundle(
        address from,
        uint256 sandAmountPerPack,
        uint256 numPacks,
        uint256[] memory ids,
        uint256[] memory amounts,
        uint256 priceUSDPerPack
    ) internal {
        require(_sand.transferFrom(from, address(this), sandAmountPerPack.mul(numPacks)), "
            failed to transfer Sand");
        uint256 saleId = sales.push(Sale({
            ids: ids,
            amounts : amounts,
            sandAmount: sandAmountPerPack,
            priceUSD: priceUSDPerPack,
            numPacksLeft: numPacks
        }));
        emit BundleSale(saleId, ids, amounts, sandAmountPerPack, priceUSDPerPack, numPacks);
    }
}
```

File TheSandbox712.sol

```
pragma solidity 0.5.9;
```

```

import "../sandbox-private-contracts/contracts_common/src/BaseWithStorage/
    ProxyImplementation.sol";

contract TheSandbox712 is ProxyImplementation {
    bytes32 constant EIP712DOMAIN_TYPEHASH = keccak256(
        "EIP712Domain(string name,string version,address verifyingContract)"
    );
    bytes32 DOMAIN_SEPARATOR;

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK init712
      @tag assume_completion
      @post __post._initialised["712"] == true
     */
    function init712() public phase("712") {
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                EIP712DOMAIN_TYPEHASH,
                keccak256("The Sandbox 3D"),
                keccak256("1"),
                address(this)
            )
        );
    }

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function domainSeparator() internal view returns (bytes32) {
        return DOMAIN_SEPARATOR;
    }
}
```

File AssetSignedAuction.sol

```
pragma solidity 0.5.9;

import "../sandbox-private-contracts/contracts_common/src/Libraries/SigUtil.sol";
import "../sandbox-private-contracts/contracts_common/src/Libraries/PriceUtil.sol";
import "../sandbox-private-contracts/src/Sand.sol";
import "../sandbox-private-contracts/src/Asset.sol";
import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC20.sol";
import "../sandbox-private-contracts/src/TheSandbox712.sol";
import "../sandbox-private-contracts/contracts_common/src/BaseWithStorage/
    MetaTransactionReceiver.sol";
import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC1271.sol";
import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC1271Constants.sol";
import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC1654.sol";
import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC1654Constants.sol";
import "../sandbox-private-contracts/contracts_common/src/Libraries/SafeMathWithRequire.sol
    ";

contract AssetSignedAuction is ERC1654Constants, ERC1271Constants, TheSandbox712,
    MetaTransactionReceiver {
    using SafeMathWithRequire for uint256;

```

```
    enum SignatureType { DIRECT, EIP1654, EIP1271 }

    bytes32 constant AUCTION_TYPEHASH = keccak256(
        "Auction(address from,address token,uint256 offerId,uint256 startingPrice,uint256
            endingPrice,uint256 startedAt,uint256 duration,uint256 packs,bytes ids,bytes
            amounts)"
    );

    event OfferClaimed(
        address indexed seller,
        address indexed buyer,
        uint256 indexed offerId,
        uint256 amount,
        uint256 pricePaid,
        uint256 feePaid
    );
    event OfferCancelled(address indexed seller, uint256 indexed offerId);

    uint256 constant MAX_UINT256 = 0
        xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff;

    // Stack too deep, grouping parameters
    // AuctionData:
    uint256 constant AuctionData_OfferId = 0;
    uint256 constant AuctionData_StartingPrice = 1;
    uint256 constant AuctionData_EndingPrice = 2;
    uint256 constant AuctionData_StartedAt = 3;
    uint256 constant AuctionData_Duration = 4;
    uint256 constant AuctionData_Packs = 5;

    mapping(address => mapping(uint256 => uint256)) claimed;

    Asset _asset;
    uint256 _fee10000th = 0;
    address payable _feeCollector;

    event FeeSetup(address feeCollector, uint256 fee10000th);

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK AssetSignedAuction
      @post __post._asset == asset
      @post __post._feeCollector == feeCollector
      @post __post._fee10000th == fee10000th
      @post __post._admin == admin
      @post __post._metaTransactionContracts[initialMetaTx] == true
     */
    constructor(Asset asset, address admin, address initialMetaTx, address payable
        feeCollector, uint256 fee10000th) public {
        _asset = asset;
        _feeCollector = feeCollector;
        _fee10000th = fee10000th;
        emit FeeSetup(feeCollector, fee10000th);
        _admin = admin;
        _setMetaTransactionProcessor(initialMetaTx, true);
        init712();
    }
```

```
    /// @notice set fee parameters
    /// @param feeCollector address receiving the fee
    /// @param fee10000th fee in 10,000th
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK setFee_require
      @tag assume_completion
      @post msg.sender == _admin
     */
    /*@CTK setFee_change
      @tag assume_completion
      @pre msg.sender == _admin
      @post __post._feeCollector == feeCollector
      @post __post._fee10000th == fee10000th
     */
    function setFee(address payable feeCollector, uint256 fee10000th) external {
        require(msg.sender == _admin, "only admin can change fee");
        _feeCollector = feeCollector;
        _fee10000th = fee10000th;
        emit FeeSetup(feeCollector, fee10000th);
    }

    //@CTK NO_OVERFLOW
    //@CTK FAIL NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK _verifyParameters
      @tag assume_completion
      @post ids.length == amounts.length
      @post buyer == msg.sender \/ (token != address(0) /\ _metaTransactionContracts[msg.
          sender])
      @post claimed[seller][auctionData[AuctionData_OfferId]] != MAX_UINT256
      @post claimed[seller][auctionData[AuctionData_OfferId]] + buyAmount <= auctionData[
          AuctionData_Packs]
      @post auctionData[AuctionData_StartedAt] <= block.timestamp
      @post auctionData[AuctionData_StartedAt] + auctionData[AuctionData_Duration] > block.
          timestamp
     */
    function _verifyParameters(
        address buyer,
        address payable seller,
        address token,
        uint256 buyAmount,
        uint256[] memory auctionData,
        uint256[] memory ids,
        uint256[] memory amounts
    ) internal view {
        require(ids.length == amounts.length, "ids and amounts length not matching");
        require(buyer == msg.sender || (token != address(0) && _metaTransactionContracts[msg
            .sender]), "not authorized");
        uint256 amountAlreadyClaimed = claimed[seller][auctionData[AuctionData_OfferId]];
        require(amountAlreadyClaimed != MAX_UINT256, "Auction cancelled");

        uint256 total = amountAlreadyClaimed.add(buyAmount);
        require(total >= amountAlreadyClaimed, "overflow");
        require(total <= auctionData[AuctionData_Packs], "Buy amount exceeds sell amount");

```

```
        require(
            auctionData[AuctionData_StartedAt] <= block.timestamp,
            "Auction didn't start yet"
        );
        require(
            auctionData[AuctionData_StartedAt].add(auctionData[AuctionData_Duration]) > block
                .timestamp,
            "Auction finished"
        );
    }

    /// @notice claim offer using EIP712
    /// @param buyer address paying for the offer
    /// @param seller address of the seller
    /// @param token token used for payment
    /// @param purchase buyAmount, maxTokenAmount
    /// @param auctionData offerId, startingPrice, endingPrice, startedAt, duration, packs
    /// @param ids ids of the Assets being sold
    /// @param amounts amounts of Assets per pack
    /// @param signature signature of seller
    function claimSellerOffer(
        address buyer,
        address payable seller,
        address token,
        uint256[] calldata purchase, // buyAmount, maxTokenAmount
        uint256[] calldata auctionData,
        uint256[] calldata ids,
        uint256[] calldata amounts,
        bytes calldata signature
    ) external payable {
        _verifyParameters(
            buyer,
            seller,
            token,
            purchase[0],
            auctionData,
            ids,
            amounts
        );
        _ensureCorrectSigner(seller, token, auctionData, ids, amounts, signature,
            SignatureType.DIRECT, true);
        _executeDeal(
            token,
            purchase,
            buyer,
            seller,
            auctionData,
            ids,
            amounts
        );
    }

    /// @notice claim offer using EIP712 and EIP1271 signature verification scheme
    /// @param buyer address paying for the offer
    /// @param seller address of the seller
    /// @param token token used for payment
    /// @param purchase buyAmount, maxTokenAmount
    /// @param auctionData offerId, startingPrice, endingPrice, startedAt, duration, packs
```

```
    /// @param ids ids of the Assets being sold
    /// @param amounts amounts of Assets per pack
    /// @param signature signature of seller
    function claimSellerOfferViaEIP1271(
        address buyer,
        address payable seller,
        address token,
        uint256[] calldata purchase, // buyAmount, maxTokenAmount
        uint256[] calldata auctionData,
        uint256[] calldata ids,
        uint256[] calldata amounts,
        bytes calldata signature
    ) external payable {
        _verifyParameters(
            buyer,
            seller,
            token,
            purchase[0],
            auctionData,
            ids,
            amounts
        );
        _ensureCorrectSigner(seller, token, auctionData, ids, amounts, signature,
            SignatureType.EIP1271, true);
        _executeDeal(
            token,
            purchase,
            buyer,
            seller,
            auctionData,
            ids,
            amounts
        );
    }

    /// @notice claim offer using EIP712 and EIP1654 signature verification scheme
    /// @param buyer address paying for the offer
    /// @param seller address of the seller
    /// @param token token used for payment
    /// @param purchase buyAmount, maxTokenAmount
    /// @param auctionData offerId, startingPrice, endingPrice, startedAt, duration, packs
    /// @param ids ids of the Assets being sold
    /// @param amounts amounts of Assets per pack
    /// @param signature signature of seller
    function claimSellerOfferViaEIP1654(
        address buyer,
        address payable seller,
        address token,
        uint256[] calldata purchase, // buyAmount, maxTokenAmount
        uint256[] calldata auctionData,
        uint256[] calldata ids,
        uint256[] calldata amounts,
        bytes calldata signature
    ) external payable {
        _verifyParameters(
            buyer,
            seller,
            token,
```

```
            purchase[0],
            auctionData,
            ids,
            amounts
        );
        _ensureCorrectSigner(seller, token, auctionData, ids, amounts, signature,
            SignatureType.EIP1654, true);
        _executeDeal(
            token,
            purchase,
            buyer,
            seller,
            auctionData,
            ids,
            amounts
        );
    }

    /// @notice claim offer using Basic Signature
    /// @param buyer address paying for the offer
    /// @param seller address of the seller
    /// @param token token used for payment
    /// @param purchase buyAmount, maxTokenAmount
    /// @param auctionData offerId, startingPrice, endingPrice, startedAt, duration, packs
    /// @param ids ids of the Assets being sold
    /// @param amounts amounts of Assets per pack
    /// @param signature signature of seller
    function claimSellerOfferUsingBasicSig(
        address buyer,
        address payable seller,
        address token,
        uint256[] calldata purchase, // buyAmount, maxTokenAmount
        uint256[] calldata auctionData,
        uint256[] calldata ids,
        uint256[] calldata amounts,
        bytes calldata signature
    ) external payable {
        _verifyParameters(
            buyer,
            seller,
            token,
            purchase[0],
            auctionData,
            ids,
            amounts
        );
        _ensureCorrectSigner(seller, token, auctionData, ids, amounts, signature,
            SignatureType.DIRECT, false);
        _executeDeal(
            token,
            purchase,
            buyer,
            seller,
            auctionData,
            ids,
            amounts
        );
    }
```

```

    /// @notice claim offer using Basic Signature and EIP1271 signature verification scheme
    /// @param buyer address paying for the offer
    /// @param seller address of the seller
    /// @param token token used for payment
    /// @param purchase buyAmount, maxTokenAmount
    /// @param auctionData offerId, startingPrice, endingPrice, startedAt, duration, packs
    /// @param ids ids of the Assets being sold
    /// @param amounts amounts of Assets per pack
    /// @param signature signature of seller
    function claimSellerOfferUsingBasicSigViaEIP1271(
        address buyer,
        address payable seller,
        address token,
        uint256[] calldata purchase, // buyAmount, maxTokenAmount
        uint256[] calldata auctionData,
        uint256[] calldata ids,
        uint256[] calldata amounts,
        bytes calldata signature
    ) external payable {
        _verifyParameters(
            buyer,
            seller,
            token,
            purchase[0],
            auctionData,
            ids,
            amounts
        );
        _ensureCorrectSigner(seller, token, auctionData, ids, amounts, signature,
            SignatureType.EIP1271, false);
        _executeDeal(
            token,
            purchase,
            buyer,
            seller,
            auctionData,
            ids,
            amounts
        );
    }

    /// @notice claim offer using Basic Signature and EIP1654 signature verification scheme
    /// @param buyer address paying for the offer
    /// @param seller address of the seller
    /// @param token token used for payment
    /// @param purchase buyAmount, maxTokenAmount
    /// @param auctionData offerId, startingPrice, endingPrice, startedAt, duration, packs
    /// @param ids ids of the Assets being sold
    /// @param amounts amounts of Assets per pack
    /// @param signature signature of seller
    function claimSellerOfferUsingBasicSigViaEIP1654(
        address buyer,
        address payable seller,
        address token,
        uint256[] calldata purchase, // buyAmount, maxTokenAmount
        uint256[] calldata auctionData,
        uint256[] calldata ids,
```

```
        uint256[] calldata amounts,
        bytes calldata signature
    ) external payable {
        _verifyParameters(
            buyer,
            seller,
            token,
            purchase[0],
            auctionData,
            ids,
            amounts
        );
        _ensureCorrectSigner(seller, token, auctionData, ids, amounts, signature,
            SignatureType.EIP1654, false);
        _executeDeal(
            token,
            purchase,
            buyer,
            seller,
            auctionData,
            ids,
            amounts
        );
    }

    //@FIXME NO_OVERFLOW
    //@FIXME NO_BUF_OVERFLOW
    //@FIXME NO_ASF
    /*@FIXME _executeDeal_require
      @tag assume_completion
     */
    function _executeDeal(
        address token,
        uint256[] memory purchase,
        address buyer,
        address payable seller,
        uint256[] memory auctionData,
        uint256[] memory ids,
        uint256[] memory amounts
    ) internal {
        uint256 offer = PriceUtil.calculateCurrentPrice(
                auctionData[AuctionData_StartingPrice],
                auctionData[AuctionData_EndingPrice],
                auctionData[AuctionData_Duration],
                block.timestamp.sub(auctionData[AuctionData_StartedAt])
            ).mul(purchase[0]);
        claimed[seller][auctionData[AuctionData_OfferId]] = claimed[seller][auctionData[
            AuctionData_OfferId]].add(purchase[0]);

        uint256 fee = 0;
        if(_fee10000th > 0) {
            fee = PriceUtil.calculateFee(offer, _fee10000th);
        }

        uint256 total = offer.add(fee);
        require(total <= purchase[1], "offer exceeds max amount to spend");

        if (token != address(0)) {
```

```
        require(ERC20(token).transferFrom(buyer, seller, offer), "failed to transfer token price");
        if(fee > 0) {
            require(ERC20(token).transferFrom(buyer, _feeCollector, fee), "failed to collect fee");
        }
    } else {
        require(msg.value >= total, "ETH < offer+fee");
        if(msg.value > total) {
            msg.sender.transfer(msg.value.sub(total));
        }
        seller.transfer(offer);
        if(fee > 0) {
            _feeCollector.transfer(fee);
        }
    }

    uint256[] memory packAmounts = new uint256[](amounts.length);
    /*@CTK FAIL "_executeDeal_loop"
      @tag assume_completion
      @inv i <= packAmounts.length
      @post forall j: uint256. (j >= 0 /\ j < packAmounts.length) -> packAmounts[j] == amounts[j] * purchase[0]
      @post i == packAmounts.length
      @post !__should_return
     */
    for (uint256 i = 0; i < packAmounts.length; i++) {
        packAmounts[i] = amounts[i].mul(purchase[0]);
    }
    _asset.safeBatchTransferFrom(seller, buyer, ids, packAmounts, "");
    emit OfferClaimed(
        seller,
        buyer,
        auctionData[AuctionData_OfferId],
        purchase[0],
        offer,
        fee
    );
}

/// @notice cancel a offer previously signed, new offer need to use a id not used yet
/// @param offerId offer to cancel
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK cancelSellerOffer
  @post __post.claimed[msg.sender][offerId] == MAX_UINT256
 */
function cancelSellerOffer(uint256 offerId) external {
    claimed[msg.sender][offerId] = MAX_UINT256;
    emit OfferCancelled(msg.sender, offerId);
}

function _ensureCorrectSigner(
    address from,
    address token,
    uint256[] memory auctionData,
    uint256[] memory ids,
```

```
        uint256[] memory amounts,
        bytes memory signature,
        SignatureType signatureType,
        bool eip712
    ) internal view returns (address) {
        bytes memory dataToHash;

        if(eip712) {
            dataToHash = abi.encodePacked(
                "\x19\x01",
                domainSeparator(),
                _hashAuction(from, token, auctionData, ids, amounts)
            );
        } else {
            dataToHash = _encodeBasicSignatureHash(from, token, auctionData, ids, amounts);
        }

        if (signatureType == SignatureType.EIP1271) {
            require(
                ERC1271(from).isValidSignature(dataToHash, signature) == ERC1271_MAGICVALUE,
                "invalid 1271 signature"
            );
        } else if(signatureType == SignatureType.EIP1654){
            require(
                ERC1654(from).isValidSignature(keccak256(dataToHash), signature) ==
                    ERC1654_MAGICVALUE,
                "invalid 1654 signature"
            );
        } else {
            address signer = SigUtil.recover(keccak256(dataToHash), signature);
            require(signer == from, "signer != from");
        }
    }

    function _encodeBasicSignatureHash(
        address from,
        address token,
        uint256[] memory auctionData,
        uint256[] memory ids,
        uint256[] memory amounts
    ) internal view returns (bytes memory) {
        return SigUtil.prefixed(keccak256(abi.encodePacked(
                address(this),
                AUCTION_TYPEHASH,
                from,
                token,
                auctionData[AuctionData_OfferId],
                auctionData[AuctionData_StartingPrice],
                auctionData[AuctionData_EndingPrice],
                auctionData[AuctionData_StartedAt],
                auctionData[AuctionData_Duration],
                auctionData[AuctionData_Packs],
                keccak256(abi.encodePacked(ids)),
                keccak256(abi.encodePacked(amounts))
            )));
    }

    function _hashAuction(
```

```
        address from,
        address token,
        uint256[] memory auctionData,
        uint256[] memory ids,
        uint256[] memory amounts
    ) internal pure returns (bytes32) {
        return
            keccak256(
                abi.encode(
                    AUCTION_TYPEHASH,
                    from,
                    token,
                    auctionData[AuctionData_OfferId],
                    auctionData[AuctionData_StartingPrice],
                    auctionData[AuctionData_EndingPrice],
                    auctionData[AuctionData_StartedAt],
                    auctionData[AuctionData_Duration],
                    auctionData[AuctionData_Packs],
                    keccak256(abi.encodePacked(ids)),
                    keccak256(abi.encodePacked(amounts))
                )
            );
    }
}
```