

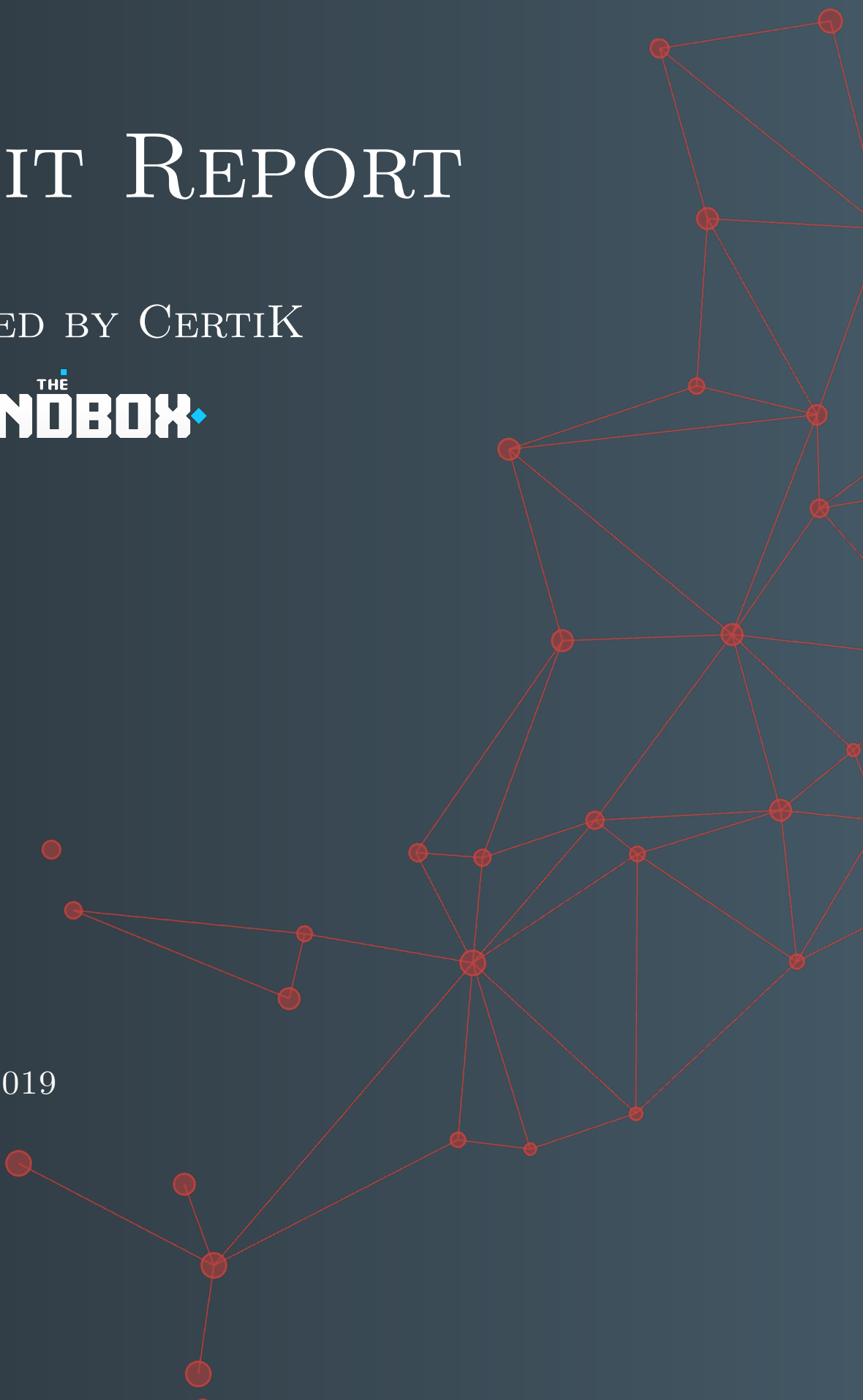


AUDIT REPORT

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	1
About CertiK	2
Executive Summary	3
Vulnerability Classification	3
Testing Summary	4
Audit Score	4
Type of Issues	4
Vulnerability Details	5
Manual Review Notes	6
Static Analysis Results	8
Formal Verification Results	10
How to read	10
Source Code with CertiK Labels	76

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 Function	An overflow/underflow occurs when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
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 Checksum¹

- **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

¹Commit: `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 `ids.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 `ids.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

```
1 pragma solidity ^0.5.2;
```

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

INSECURE_COMPILER_VERSION

Line 1 in File CommonMinter.sol

```
1 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

```
1 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

```
1 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

```
1 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

```
1 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

```
128 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	<pre> 30 /*@CTK FAIL "transferFrom to same address" 31 @tag assume_completion 32 @pre from == to 33 @post __post.allowed[from][msg.sender] == 34 */ </pre>
Raw code location	Line 35-41 in File howtoread.sol
Raw code	<pre> 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 } </pre>
Counterexample	 This code violates the specification
Initial environment	<pre> 1 Counter Example: 2 Before Execution: 3 Input = { 4 from = 0x0 5 to = 0x0 6 tokens = 0x6c 7 } 8 This = 0 </pre>
Post environment	<pre> 52 } 53 balance: 0x0 54 } 55 } 56 57 After Execution: 58 Input = { 59 from = 0x0 60 to = 0x0 61 tokens = 0x6c </pre>

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


```

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 }

```

✗ This code violates the specification.

```


1 Counter Example:
2 Before Execution:
3   Input = {
4     a = 0
5     b = 0
6   }
7   Internal = {
8     __has_assertion_failure = false
9     __has_buf_overflow = false
10    __has_overflow = false
11    __has_returned = false
12    __reverted = false
13    msg = {
14      "gas": 0,
15      "sender": 0,
16      "value": 0
17    }
18  }
19  Other = {
20    __return = 0
21    block = {
22      "number": 0,
23      "timestamp": 0
24    }
25  }
26  Address_Map = [
27    {
28      "key": "ALL_OTHERS",
29      "value": "EmptyAddress"
30    }
31  ]
32
33 Function invocation is reverted.

```

Formal Verification Request 8

div

 10, Dec 2019

 0.28 ms

Line 37-40 in File SafeMathWithRequire.sol

```

37 /*@CTK div
38   @tag assume_completion
39   @post __return == a / b
40 */

```

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 9

If method completes, integer overflow would not happen.



10, Dec 2019



10.72 ms

Line 51 in File SafeMathWithRequire.sol

```
51 // @CTK_NO_OVERFLOW
```

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 10

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



10, Dec 2019



0.44 ms

Line 52 in File SafeMathWithRequire.sol

```
52 // @CTK_NO_BUF_OVERFLOW
```

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 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, "underflow");
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, "underflow");
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

```
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 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
31         feeReceiver)
32     public
33     {
34         _sand = sand;
35         _asset = asset;
36         _feePerCopy = feePerCopy;
37         _admin = admin;
38         _feeReceiver = feeReceiver;
39         _setMetaTransactionProcessor(address(sand), true);
40     }
```

✓ 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
31         feeReceiver)
32     public
33     {
34         _sand = sand;
35         _asset = asset;
36         _feePerCopy = feePerCopy;
37         _admin = admin;
38         _feeReceiver = feeReceiver;
```

```
38     _setMetaTransactionProcessor(address(sand), true);
39 }
```

✓ 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

```
20 //CTK NO_ASF
```

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 20

CommonMinter



10, Dec 2019



2.13 ms

Line 21-29 in File CommonMinter.sol

```
21 /*CTK CommonMinter
22     @tag assume_completion
23     @post __post._sand == sand
24     @post __post._asset == asset
25     @post __post._feePerCopy == feePerCopy
26     @post __post._admin == admin
27     @post __post._feeReceiver == feeReceiver
28     @post __post._metaTransactionContracts[address(sand)] == true
29 */
```

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 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;
```

```
81     _receivingWallet = receivingWallet;  
82 }
```

✓ 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

```

94      /*CTK setReceivingWallet_change
95      @tag assume_completion
96      @pre newWallet != address(0)
97      @pre msg.sender == _admin
98      @post __post._receivingWallet == newWallet
99      */

```

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 45

If method completes, integer overflow would not happen.



10, Dec 2019



7.85 ms

Line 106 in File BundleSandSale.sol

```

106     //@CTK NO_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.
124             sales[saleIndex].amounts[j] == sales[saleIndex].amounts[j] * numPacks)
125         @post i == sales[saleIndex].ids.length
126         */
127         for (uint256 i = 0; i < numIds; i++) {
128             amounts[i] = amounts[i].mul(numPacks);
129         }
130         _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.
124          sales[saleIndex].amounts[j] == sales[saleIndex].amounts[j] * numPacks)
125      @post i == sales[saleIndex].ids.length
126      */
127     for (uint256 i = 0; i < numIds; i++) {
128         amounts[i] = amounts[i].mul(numPacks);
129     }
130     _asset.safeBatchTransferFrom(address(this), to, ids, amounts, "");
131 }
```

✗ 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     }
```

```

21 Other = {
22     block = {
23         "number": 0,
24         "timestamp": 0
25     }
26 }
27 Address_Map = [
28     {
29         "key": "ALL_OTHERS",
30         "value": {
31             "contract_name": "BundleSandSale",
32             "balance": 0,
33             "contract": {
34                 "ERC1155_RECEIVED": "AAAA",
35                 "ERC1155_BATCH_RECEIVED": "IIII",
36                 "_medianizer": 0,
37                 "_dai": 0,
38                 "_sand": 0,
39                 "_asset": 0,
40                 "_receivingWallet": 0,
41                 "sales": [
42                     {
43                         "ids": [
44                             {
45                                 "key": "ALL_OTHERS",
46                                 "value": 8
47                             }
48                         ],
49                         "amounts": [
50                             {
51                                 "key": "ALL_OTHERS",
52                                 "value": 8
53                             }
54                         ],
55                         "sandAmount": 128,
56                         "priceUSD": 128,
57                         "numPacksLeft": 128
58                     }
59                 ],
60                 "_admin": 0
61             }
62         }
63     }
64 ]

```

66 After Execution:

```

67 Input = {
68     numPacks = 0
69     saleIndex = 128
70     to = 0
71 }
72 This = 0
73 Internal = {
74     __has_assertion_failure = false
75     __has_buf_overflow = true
76     __has_overflow = false
77     __has_returned = false
78     __reverted = false

```

```

79     msg = {
80         "gas": 0,
81         "sender": 0,
82         "value": 0
83     }
84 }
85 Other = {
86     block = {
87         "number": 0,
88         "timestamp": 0
89     }
90 }
91 Address_Map = [
92     {
93         "key": "ALL_OTHERS",
94         "value": {
95             "contract_name": "BundleSandSale",
96             "balance": 0,
97             "contract": {
98                 "ERC1155_RECEIVED": "AAAA",
99                 "ERC1155_BATCH_RECEIVED": "IIII",
100                 "_medianizer": 0,
101                 "_dai": 0,
102                 "_sand": 0,
103                 "_asset": 0,
104                 "_receivingWallet": 0,
105                 "sales": [
106                     {
107                         "ids": [
108                             {
109                                 "key": "ALL_OTHERS",
110                                 "value": 8
111                             }
112                         ],
113                         "amounts": [
114                             {
115                                 "key": "ALL_OTHERS",
116                                 "value": 8
117                             }
118                         ],
119                         "sandAmount": 128,
120                         "priceUSD": 128,
121                         "numPacksLeft": 128
122                     }
123                 ],
124                 "_admin": 0
125             }
126         }
127     }
128 ]

```

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.
124           sales[saleIndex].amounts[j] == sales[saleIndex].amounts[j] * numPacks)
125       @post i == sales[saleIndex].ids.length
126       */
127      for (uint256 i = 0; i < numIds; i++) {
128          amounts[i] = amounts[i].mul(numPacks);
129      }
130      _asset.safeBatchTransferFrom(address(this), to, ids, amounts, "");
131  }
```

✓ 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.
124           sales[saleIndex].amounts[j] == sales[saleIndex].amounts[j] * numPacks)
125       @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 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
151         payable {
152         require(saleId > 0, "invalid saleId");
153         uint256 saleIndex = saleId - 1;
154         uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
155         require(numPacksLeft >= numPacks, "not enough packs on sale");
156         sales[saleIndex].numPacksLeft = numPacksLeft - numPacks;
157
158         uint256 USDRequired = numPacks.mul(sales[saleIndex].priceUSD);
159         uint256 ETHRequired = getEtherAmountWithUSD(USDRequired);
160         require(msg.value >= ETHRequired, "not enough ether sent");
161         uint256 leftOver = msg.value - ETHRequired;
162         if(leftOver > 0) {
163             msg.sender.transfer(leftOver); // refund extra
164         }
165         address(_receivingWallet).transfer(ETHRequired);
166         _transferPack(saleIndex, numPacks, to);
167         emit BundleSold(saleId, msg.sender, numPacks, address(0), ETHRequired);

```

✓ 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

```

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.

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     numPacks = 17
5     saleId = 8
6     to = 0
7   }
8   This = 16
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": 17
19     }
20   }
21   Other = {
22     block = {
23       "number": 0,
24       "timestamp": 0
25     }
26   }
27   Address_Map = [
28     {
29       "key": 16,
30       "value": {
31         "contract_name": "BundleSandSale",
32         "balance": 0,
33         "contract": {
34           "ERC1155_RECEIVED": "BBBB",
35           "ERC1155_BATCH_RECEIVED": "BBBB",
36           "_medianizer": 0,
37           "_dai": 64,

```

```


38         "_sand": 0,
39         "_asset": 0,
40         "_receivingWallet": 0,
41         "sales": [],
42         "_admin": 0
43     }
44 }
45 },
46 {
47     "key": "ALL_OTHERS",
48     "value": "EmptyAddress"
49 }
50 ]
51
52 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.

1 Counter Example:

2 Before Execution:

```

3     Input = {
4         numPacks = 128
5         saleId = 32
6         to = 0
7     }
8     This = 1

```



```

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": 59
19          }
20      }
21      Other = {
22          block = {
23              "number": 0,
24              "timestamp": 0
25          }
26      }
27      Address_Map = [
28          {
29              "key": 1,
30              "value": {
31                  "contract_name": "BundleSandSale",
32                  "balance": 0,
33                  "contract": {
34                      "ERC1155_RECEIVED": "||||",
35                      "ERC1155_BATCH_RECEIVED": "||||",
36                      "_medianizer": 0,
37                      "_dai": 0,
38                      "_sand": 0,
39                      "_asset": 0,
40                      "_receivingWallet": 0,
41                      "sales": [
42                          {
43                              "key": 31,
44                              "value": {
45                                  "ids": [],
46                                  "amounts": [],
47                                  "sandAmount": 0,
48                                  "priceUSD": 0,
49                                  "numPacksLeft": 128
50                              }
51                          },
52                          {
53                              "key": "ALL_OTHERS",
54                              "value": {
55                                  "ids": [
56                                      {
57                                          "key": "ALL_OTHERS",
58                                          "value": 128
59                                      }
60                                  ],
61                                  "amounts": [
62                                      {
63                                          "key": "ALL_OTHERS",
64                                          "value": 128
65                                      }
66                                  ],

```

```


67         "sandAmount": 128,
68         "priceUSD": 128,
69         "numPacksLeft": 128
70     }
71 }
72 ],
73     "_admin": 0
74 }
75 }
76 },
77 {
78     "key": "ALL_OTHERS",
79     "value": "EmptyAddress"
80 }
81 ]
82
83 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
151    payable {
152      require(saleId > 0, "invalid saleId");
153      uint256 saleIndex = saleId - 1;
154      uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
155      require(numPacksLeft >= numPacks, "not enough packs on sale");
156      sales[saleIndex].numPacksLeft = numPacksLeft - numPacks;
157
158      uint256 USDRequired = numPacks.mul(sales[saleIndex].priceUSD);
159      uint256 ETHRequired = getEtherAmountWithUSD(USDRequired);
160      require(msg.value >= ETHRequired, "not enough ether sent");
161      uint256 leftOver = msg.value - ETHRequired;
162      if(leftOver > 0) {
163          msg.sender.transfer(leftOver); // refund extra
164      }
165      address(_receivingWallet).transfer(ETHRequired);
166      _transferPack(saleIndex, numPacks, to);
167      emit BundleSold(saleId, msg.sender, numPacks, address(0), ETHRequired);

```

 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
194          transfer dai");
195      _transferPack(saleIndex, numPacks, to);
196
197      emit BundleSold(saleId, msg.sender, numPacks, address(_dai), USDRequired);
198  }
```

✓ 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
194          transfer dai");
195      _transferPack(saleIndex, numPacks, to);
196
197      emit BundleSold(saleId, msg.sender, numPacks, address(_dai), USDRequired);
198  }
```

✗ This code violates the specification.

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     numPacks = 70
5     saleId = 17
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  }
21  Other = {
22    block = {
23      "number": 0,
24      "timestamp": 0
25    }
26  }
27  Address_Map = [
28    {
29      "key": 0,
30      "value": {
31        "contract_name": "BundleSandSale",
32        "balance": 0,
33        "contract": {
34          "ERC1155_RECEIVED": "\u00b1\u00b1\u00b1\u00b1",
35          "ERC1155_BATCH_RECEIVED": "\u00b1\u00b1\u00b1\u00b1",
36          "_medianizer": 0,
37          "_dai": 0,
38          "_sand": 0,
39          "_asset": 0,
40          "_receivingWallet": 0,
41          "sales": [
42            {
43              "ids": [
44                {
45                  "key": "ALL_OTHERS",
46                  "value": 112
47                }
48              ],
49              "amounts": [
50                {
51                  "key": "ALL_OTHERS",
52                  "value": 112
53                }
54              ],
55              "sandAmount": 112,
56              "priceUSD": 112,
57              "numPacksLeft": 112
58            }
59          ]
60        }
61      }
62    }
63  ]

```

```

59         ],
60         "_admin": 0
61     }
62 }
63 },
64 {
65     "key": "ALL_OTHERS",
66     "value": "EmptyAddress"
67 }
68 ]
69
70 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
194         transfer dai");
195     _transferPack(saleIndex, numPacks, to);
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

```

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 }

```

✗ This code violates the specification.

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     saleId = 129
5   }
6   This = 0
7   Internal = {
8     __has_assertion_failure = false
9     __has_buf_overflow = false
10    __has_overflow = false
11    __has_returned = false
12    __reverted = false
13    msg = {
14      "gas": 0,
15      "sender": 0,
16      "value": 0
17    }
18  }
19  Other = {
20    block = {
21      "number": 0,
22      "timestamp": 0
23    }
24    numPacksLeft = 0
25    priceUSD = 0
26  }
27  Address_Map = [
28    {
29      "key": "ALL_OTHERS",
30      "value": {
31        "contract_name": "BundleSandSale",
32        "balance": 0,
33        "contract": {
34          "ERC1155_RECEIVED": "AAAA",
35          "ERC1155_BATCH_RECEIVED": "AAAA",
36          "_medianizer": 0,
37          "_dai": 0,
38          "_sand": 0,
39          "_asset": 0,
40          "_receivingWallet": 0,
41          "sales": [
42            {
43              "ids": [],
44              "amounts": [],
45              "sandAmount": 0,
46              "priceUSD": 0,
47              "numPacksLeft": 0
48            }
49          ],

```

```

50     "_admin": 0
51   }
52 }
53 }
54 ]
55
56 After Execution:
57   Input = {
58     saleId = 129
59   }
60   This = 0
61   Internal = {
62     __has_assertion_failure = false
63     __has_buf_overflow = true
64     __has_overflow = false
65     __has_returned = false
66     __reverted = false
67     msg = {
68       "gas": 0,
69       "sender": 0,
70       "value": 0
71     }
72   }
73   Other = {
74     block = {
75       "number": 0,
76       "timestamp": 0
77     }
78     numPacksLeft = 0
79     priceUSD = 0
80   }
81   Address_Map = [
82     {
83       "key": "ALL_OTHERS",
84       "value": {
85         "contract_name": "BundleSandSale",
86         "balance": 0,
87         "contract": {
88           "ERC1155_RECEIVED": "AAAA",
89           "ERC1155_BATCH_RECEIVED": "AAAA",
90           "_medianizer": 0,
91           "_dai": 0,
92           "_sand": 0,
93           "_asset": 0,
94           "_receivingWallet": 0,
95           "sales": [
96             {
97               "ids": [],
98               "amounts": [],
99               "sandAmount": 0,
100              "priceUSD": 0,
101              "numPacksLeft": 0
102            }
103          ],
104          "_admin": 0
105        }
106      }
107    ]


```


108]

Formal Verification Request 59

Method will not encounter an assertion failure.

 10, Dec 2019

 0.48 ms

Line 203 in File BundleSandSale.sol

203 // @CTK NO_ASF

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 60

getSaleInfo__require

 10, Dec 2019

 0.53 ms

Line 204-207 in File BundleSandSale.sol

```
204 /* @CTK getSaleInfo__require
205     @tag assume_completion
206     @post saleId > 0
207 */
```

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 61

getSaleInfo__change

 10, Dec 2019

 1.79 ms

Line 208-213 in File BundleSandSale.sol

```

208  /*@CTK getSaleInfo_change
209  @tag assume_completion
210  @pre saleId > 0
211  @post priceUSD == sales[saleId - 1].priceUSD
212  @post numPacksLeft == sales[saleId - 1].numPacksLeft
213  */

```

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 62

If method completes, integer overflow would not happen.

📅 10, Dec 2019

🕒 38.08 ms

Line 221 in File BundleSandSale.sol

```

221  //@CTK NO_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  }

```

✓ 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
```

```

18     }
19 }
20 Other = {
21     block = {
22         "number": 0,
23         "timestamp": 0
24     }
25 }
26 Address_Map = [
27     {
28         "key": 0,
29         "value": {
30             "contract_name": "BundleSandSale",
31             "balance": 0,
32             "contract": {
33                 "ERC1155_RECEIVED": "\u00c1\u00c1\u00c1\u00c1",
34                 "ERC1155_BATCH_RECEIVED": "\u00c1\u00c1\u00c1\u00c1",
35                 "_medianizer": 0,
36                 "_dai": 0,
37                 "_sand": 0,
38                 "_asset": 0,
39                 "_receivingWallet": 0,
40                 "sales": [
41                     {
42                         "ids": [
43                             {
44                                 "key": "ALL_OTHERS",
45                                 "value": 128
46                             }
47                         ],
48                         "amounts": [
49                             {
50                                 "key": "ALL_OTHERS",
51                                 "value": 128
52                             }
53                         ],
54                         "sandAmount": 128,
55                         "priceUSD": 128,
56                         "numPacksLeft": 128
57                     },
58                     {
59                         "ids": [
60                             {
61                                 "key": "ALL_OTHERS",
62                                 "value": 128
63                             }
64                         ],
65                         "amounts": [
66                             {
67                                 "key": "ALL_OTHERS",
68                                 "value": 128
69                             }
70                         ],
71                         "sandAmount": 128,
72                         "priceUSD": 128,
73                         "numPacksLeft": 128
74                     },
75                     {

```

```

76         "ids": [
77             {
78                 "key": "ALL_OTHERS",
79                 "value": 128
80             }
81         ],
82         "amounts": [
83             {
84                 "key": "ALL_OTHERS",
85                 "value": 128
86             }
87         ],
88         "sandAmount": 128,
89         "priceUSD": 128,
90         "numPacksLeft": 128
91     },
92     {
93         "ids": [
94             {
95                 "key": "ALL_OTHERS",
96                 "value": 128
97             }
98         ],
99         "amounts": [
100             {
101                 "key": "ALL_OTHERS",
102                 "value": 128
103             }
104         ],
105         "sandAmount": 128,
106         "priceUSD": 128,
107         "numPacksLeft": 128
108     }
109 ],
110 "_admin": 0
111 }
112 }
113 },
114 {
115     "key": "ALL_OTHERS",
116     "value": "EmptyAddress"
117 }
118 ]
119

```

```

120 After Execution:
121     Input = {
122         saleId = 129
123         to = 0
124     }
125     This = 0
126     Internal = {
127         __has_assertion_failure = false
128         __has_buf_overflow = true
129         __has_overflow = false
130         __has_returned = false
131         __reverted = false
132         msg = {
133             "gas": 0,

```

```

134     "sender": 0,
135     "value": 0
136   }
137 }
138 Other = {
139   block = {
140     "number": 0,
141     "timestamp": 0
142   }
143 }
144 Address_Map = [
145   {
146     "key": 0,
147     "value": {
148       "contract_name": "BundleSandSale",
149       "balance": 0,
150       "contract": {
151         "ERC1155_RECEIVED": "\u00c1\u00c1\u00c1\u00c1",
152         "ERC1155_BATCH_RECEIVED": "\u00c1\u00c1\u00c1\u00c1",
153         "_medianizer": 0,
154         "_dai": 0,
155         "_sand": 0,
156         "_asset": 0,
157         "_receivingWallet": 0,
158         "sales": [
159           {
160             "ids": [
161               {
162                 "key": "ALL_OTHERS",
163                 "value": 128
164               }
165             ],
166             "amounts": [
167               {
168                 "key": "ALL_OTHERS",
169                 "value": 128
170               }
171             ],
172             "sandAmount": 128,
173             "priceUSD": 128,
174             "numPacksLeft": 128
175           },
176           {
177             "ids": [
178               {
179                 "key": "ALL_OTHERS",
180                 "value": 128
181               }
182             ],
183             "amounts": [
184               {
185                 "key": "ALL_OTHERS",
186                 "value": 128
187               }
188             ],
189             "sandAmount": 128,
190             "priceUSD": 128,
191             "numPacksLeft": 128

```

```

192     },
193     {
194         "ids": [
195             {
196                 "key": "ALL_OTHERS",
197                 "value": 128
198             }
199         ],
200         "amounts": [
201             {
202                 "key": "ALL_OTHERS",
203                 "value": 128
204             }
205         ],
206         "sandAmount": 128,
207         "priceUSD": 128,
208         "numPacksLeft": 128
209     },
210     {
211         "ids": [
212             {
213                 "key": "ALL_OTHERS",
214                 "value": 128
215             }
216         ],
217         "amounts": [
218             {
219                 "key": "ALL_OTHERS",
220                 "value": 128
221             }
222         ],
223         "sandAmount": 128,
224         "priceUSD": 128,
225         "numPacksLeft": 128
226     }
227 ],
228 "_admin": 0
229 }
230 }
231 },
232 {
233     "key": "ALL_OTHERS",
234     "value": "EmptyAddress"
235 }
236 ]

```

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)

```



```

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 66

withdrawSale_change



10, Dec 2019



2.98 ms

Line 229-234 in File BundleSandSale.sol

```

229     /*@CTK withdrawSale_change
230     @tag assume_completion
231     @pre msg.sender == _admin
232     @pre saleId > 0
233     @post __post.sales[saleId-1].numPacksLeft == 0
234     */

```

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 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.

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     usdAmount = 96
5   }
6   This = 0
7   Internal = {
8     __has_assertion_failure = false
9     __has_buf_overflow = false
10    __has_overflow = false
11    __has_returned = false
12    __reverted = false
13    msg = {
14      "gas": 0,
15      "sender": 0,
16      "value": 0
17    }
18  }
19  Other = {
20    __return = 0
21    block = {
22      "number": 0,
23      "timestamp": 0
24    }
25  }
26  Address_Map = [
27    {
28      "key": "ALL_OTHERS",
29      "value": {
30        "contract_name": "BundleSandSale",
31        "balance": 0,
32        "contract": {
33          "ERC1155_RECEIVED": "AAAA",
34          "ERC1155_BATCH_RECEIVED": "CCCC",
35          "_medianizer": 0,
36          "_dai": 0,
37          "_sand": 0,
38          "_asset": 0,
39          "_receivingWallet": 0,
40          "sales": [],
41          "_admin": 0
42        }
43      }
44    }
45  ]
46
47 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
65         feeCollector, uint256 fee10000th) public {
66         _asset = asset;
67         _feeCollector = feeCollector;
68         _fee10000th = fee10000th;
69         emit FeeSetup(feeCollector, fee10000th);
70         _admin = admin;
71         _setMetaTransactionProcessor(initialMetaTx, true);
72         init712();
73     }
```

✓ 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
65         feeCollector, uint256 fee10000th) public {
66         _asset = asset;
67         _feeCollector = feeCollector;
68         _fee10000th = fee10000th;
69         emit FeeSetup(feeCollector, fee10000th);
70         _admin = admin;
71         _setMetaTransactionProcessor(initialMetaTx, true);
72         init712();
73     }
```

✓ 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
65         feeCollector, uint256 fee10000th) public {
66         _asset = asset;
67         _feeCollector = feeCollector;
68         _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 80

AssetSignedAuction



10, Dec 2019



20.15 ms

Line 57-63 in File AssetSignedAuction.sol

```

57  /*@CTK AssetSignedAuction
58     @post __post._asset == asset
59     @post __post._feeCollector == feeCollector
60     @post __post._fee10000th == fee10000th
61     @post __post._admin == admin
62     @post __post._metaTransactionContracts[initialMetaTx] == true
63  */

```

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 81

If method completes, integer overflow would not happen.



10, Dec 2019



18.7 ms

Line 77 in File AssetSignedAuction.sol

```

77  //@CTK NO_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 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
120         .sender]), "not authorized");
121     uint256 amountAlreadyClaimed = claimed[seller][auctionData[AuctionData_OfferId]];
122     require(amountAlreadyClaimed != MAX_UINT256, "Auction cancelled");
123
124     uint256 total = amountAlreadyClaimed.add(buyAmount);
125     require(total >= amountAlreadyClaimed, "overflow");
126     require(total <= auctionData[AuctionData_Packs], "Buy amount exceeds sell amount");
127
128     require(
129         auctionData[AuctionData_StartedAt] <= block.timestamp,
130         "Auction didn't start yet"
131     );
132     require(
133         auctionData[AuctionData_StartedAt].add(auctionData[AuctionData_Duration]) > block
134             .timestamp,
135         "Auction finished"
136     );
137 }

```

✓ 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
120         .sender]), "not authorized");
121     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
133         .timestamp,
134         "Auction finished"
135     );

```

✗ This code violates the specification.

```

1 Counter Example:
2 Before Execution:
3     Input = {
4         amounts = []
5         auctionData = [
6             0
7         ]
8         buyAmount = 0
9         buyer = 0
10        ids = []
11        seller = 0
12        token = 64
13    }
14    This = 0
15    Internal = {
16        __has_assertion_failure = false
17        __has_buf_overflow = false
18        __has_overflow = false
19        __has_returned = false
20        __reverted = false
21        msg = {
22            "gas": 0,
23            "sender": 0,
24            "value": 0
25        }
26    }
27    Other = {
28        block = {
29            "number": 0,
30            "timestamp": 32
31        }
32    }
33    Address_Map = [
34        {
35            "key": "ALL_OTHERS",
36            "value": {
37                "contract_name": "AssetSignedAuction",
38                "balance": 0,
39                "contract": {
40                    "AUCTION_TYPEHASH": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",

```

```

41     "MAX_UINT256": 1,
42     "AuctionData_OfferId": 128,
43     "AuctionData_StartingPrice": 0,
44     "AuctionData_EndingPrice": 0,
45     "AuctionData_StartedAt": 0,
46     "AuctionData_Duration": 0,
47     "AuctionData_Packs": 0,
48     "claimed": [
49         {
50             "key": "ALL_OTHERS",
51             "value": [
52                 {
53                     "key": "ALL_OTHERS",
54                     "value": 0
55                 }
56             ]
57         }
58     ],
59     "_asset": 0,
60     "_fee10000th": 0,
61     "_feeCollector": 0,
62     "_metaTransactionContracts": [
63         {
64             "key": "ALL_OTHERS",
65             "value": false
66         }
67     ],
68     "_admin": 0,
69     "EIP712DOMAIN_TYPEHASH": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
70     "DOMAIN_SEPARATOR": "IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII",
71     "_initialised": [
72         {
73             "key": "ALL_OTHERS",
74             "value": false
75         }
76     ],
77     "ERC1271_MAGICVALUE": "AAAA",
78     "ERC1654_MAGICVALUE": "AAAA"
79     }
80 }
81 }
82 ]
83

```

84 After Execution:

```

85     Input = {
86         amounts = []
87         auctionData = [
88             0
89         ]
90         buyAmount = 0
91         buyer = 0
92         ids = []
93         seller = 0
94         token = 64
95     }
96     This = 0
97     Internal = {
98         __has_assertion_failure = false

```

```

99     __has_buf_overflow = true
100     __has_overflow = false
101     __has_returned = false
102     __reverted = false
103     msg = {
104         "gas": 0,
105         "sender": 0,
106         "value": 0
107     }
108 }
109 Other = {
110     block = {
111         "number": 0,
112         "timestamp": 32
113     }
114 }
115 Address_Map = [
116     {
117         "key": "ALL_OTHERS",
118         "value": {
119             "contract_name": "AssetSignedAuction",
120             "balance": 0,
121             "contract": {
122                 "AUCTION_TYPEHASH": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
123                 "MAX_UINT256": 1,
124                 "AuctionData_OfferId": 128,
125                 "AuctionData_StartingPrice": 0,
126                 "AuctionData_EndingPrice": 0,
127                 "AuctionData_StartedAt": 0,
128                 "AuctionData_Duration": 0,
129                 "AuctionData_Packs": 0,
130                 "claimed": [
131                     {
132                         "key": "ALL_OTHERS",
133                         "value": [
134                             {
135                                 "key": "ALL_OTHERS",
136                                 "value": 0
137                             }
138                         ]
139                     }
140                 ],
141                 "_asset": 0,
142                 "_fee10000th": 0,
143                 "_feeCollector": 0,
144                 "_metaTransactionContracts": [
145                     {
146                         "key": "ALL_OTHERS",
147                         "value": false
148                     }
149                 ],
150                 "_admin": 0,
151                 "EIP712DOMAIN_TYPEHASH": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
152                 "DOMAIN_SEPARATOR": "IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII",
153                 "_initialised": [
154                     {
155                         "key": "ALL_OTHERS",
156                         "value": false

```

```


157     }
158   ],
159   "ERC1271_MAGICVALUE": "AAAA",
160   "ERC1654_MAGICVALUE": "AAAA"
161 }
162 }
163 }
164 ]

```

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 = {

```

```

4      __should_break = false
5      __should_break__pre = false
6      __should_return = false
7      __should_return__pre = false
8      amounts = [
9          1,
10         1,
11         1,
12         1,
13         1,
14         1,
15         1,
16         1
17     ]
18     amounts__pre = []
19     i = 1
20     i__pre = 0
21     packAmounts = [
22         0
23     ]
24     packAmounts__pre = []
25     purchase = []
26     purchase__pre = []
27     this__pre = 0
28 }
29 This = 0
30 Internal = {
31     __has_assertion_failure = false
32     __has_buf_overflow = false
33     __has_overflow = false
34     __has_returned = false
35     __reverted = false
36     msg = {
37         "gas": 0,
38         "sender": 0,
39         "value": 0
40     }
41 }
42 Other = {
43     block = {
44         "number": 0,
45         "timestamp": 0
46     }
47 }
48 Address_Map = [
49     {
50         "key": 0,
51         "value": {
52             "contract_name": "AssetSignedAuction",
53             "balance": 0,
54             "contract": {
55                 "AUCTION_TYPEHASH": "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
56                 "MAX_UINT256": 0,
57                 "AuctionData_OfferId": 0,
58                 "AuctionData_StartingPrice": 0,
59                 "AuctionData_EndingPrice": 0,
60                 "AuctionData_StartedAt": 0,
61                 "AuctionData_Duration": 0,

```

```

62     "AuctionData_Packs": 0,
63     "claimed": [
64         {
65             "key": "ALL_OTHERS",
66             "value": [
67                 {
68                     "key": "ALL_OTHERS",
69                     "value": 1
70                 }
71             ]
72         }
73     ],
74     "_asset": 0,
75     "_fee10000th": 0,
76     "_feeCollector": 0,
77     "_metaTransactionContracts": [
78         {
79             "key": "ALL_OTHERS",
80             "value": false
81         }
82     ],
83     "_admin": 0,
84     "EIP712DOMAIN_TYPEHASH": "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
85     "DOMAIN_SEPARATOR": "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
86     "_initialised": [
87         {
88             "key": "ALL_OTHERS",
89             "value": false
90         }
91     ],
92     "ERC1271_MAGICVALUE": "BBBB",
93     "ERC1654_MAGICVALUE": "BBBB"
94     }
95     }
96 },
97 {
98     "key": "ALL_OTHERS",
99     "value": "EmptyAddress"
100 }
101 ]
102

```

103 After Execution:

```

104     Input = {
105         __should_break = true
106         __should_break__pre = false
107         __should_return = false
108         __should_return__pre = false
109         amounts = [
110             1,
111             1,
112             1,
113             1,
114             1,
115             1,
116             1,
117             1
118         ]
119         amounts__pre = []

```

```

120     i = 1
121     i__pre = 0
122     packAmounts = [
123         0
124     ]
125     packAmounts__pre = []
126     purchase = []
127     purchase__pre = []
128     this__pre = 0
129 }
130 This = 0
131 Internal = {
132     __has_assertion_failure = false
133     __has_buf_overflow = false
134     __has_overflow = false
135     __has_returned = true
136     __reverted = false
137     msg = {
138         "gas": 0,
139         "sender": 0,
140         "value": 0
141     }
142 }
143 Other = {
144     block = {
145         "number": 0,
146         "timestamp": 0
147     }
148 }
149 Address_Map = [
150     {
151         "key": 0,
152         "value": {
153             "contract_name": "AssetSignedAuction",
154             "balance": 0,
155             "contract": {
156                 "AUCTION_TYPEHASH": "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
157                 "MAX_UINT256": 0,
158                 "AuctionData_OfferId": 0,
159                 "AuctionData_StartingPrice": 0,
160                 "AuctionData_EndingPrice": 0,
161                 "AuctionData_StartedAt": 0,
162                 "AuctionData_Duration": 0,
163                 "AuctionData_Packs": 0,
164                 "claimed": [
165                     {
166                         "key": "ALL_OTHERS",
167                         "value": [
168                             {
169                                 "key": "ALL_OTHERS",
170                                 "value": 1
171                             }
172                         ]
173                     }
174                 ],
175                 "_asset": 0,
176                 "_fee10000th": 0,
177                 "_feeCollector": 0,

```

```

178     "_metaTransactionContracts": [
179         {
180             "key": "ALL_OTHERS",
181             "value": false
182         }
183     ],
184     "_admin": 0,
185     "EIP712DOMAIN_TYPEHASH": "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
186     "DOMAIN_SEPARATOR": "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
187     "_initialised": [
188         {
189             "key": "ALL_OTHERS",
190             "value": false
191         }
192     ],
193     "ERC1271_MAGICVALUE": "BBBB",
194     "ERC1654_MAGICVALUE": "BBBB"
195 }
196 }
197 },
198 {
199     "key": "ALL_OTHERS",
200     "value": "EmptyAddress"
201 }
202 ]

```

Source Code with CertiK Labels

File SafeMathWithRequire.sol

```

1  pragma solidity ^0.5.2;
2
3  /**
4   * @title SafeMath
5   * @dev Math operations with safety checks that revert
6   */
7  library SafeMathWithRequire {
8      /**
9       * @dev Multiplies two numbers, throws on overflow.
10      */
11      //@CTK NO_OVERFLOW
12      //@CTK NO_BUF_OVERFLOW
13      //@CTK NO_ASF
14      /*@CTK mul
15       @tag assume_completion
16       @post __return == a * b
17      */
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      }
30
31      /**
32       * @dev Integer division of two numbers, truncating the quotient.
33      */
34      //@CTK NO_OVERFLOW
35      //@CTK NO_BUF_OVERFLOW
36      //@CTK FAIL NO_ASF
37      /*@CTK div
38       @tag assume_completion
39       @post __return == a / b
40      */
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      }
47
48      /**
49       * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than
50       minuend).
51      */
52      //@CTK NO_OVERFLOW
53      //@CTK NO_BUF_OVERFLOW
54      //@CTK NO_ASF

```

```

54  /*@CTK sub
55     @tag assume_completion
56     @post __return == a - b
57  */
58  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
59     require(b <= a, "underflow");
60     return a - b;
61  }
62
63  /**
64   * @dev Adds two numbers, throws on overflow.
65  */
66  //@CTK NO_OVERFLOW
67  //@CTK NO_BUF_OVERFLOW
68  //@CTK NO_ASF
69  /*@CTK add
70     @tag assume_completion
71     @post __return == a + b
72  */
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  }
78  }

```

File CommonMinter.sol

```

1  pragma solidity 0.5.9;
2
3  import "../sandbox-private-contracts/src/Asset/ERC1155ERC721.sol";
4  import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC20.sol";
5  import "../sandbox-private-contracts/contracts_common/src/BaseWithStorage/
6  MetaTransactionReceiver.sol";
7
8  contract CommonMinter is MetaTransactionReceiver {
9     using SafeMathWithRequire for uint256;
10
11     uint256 _feePerCopy;
12
13     ERC1155ERC721 _asset;
14     mapping(address => bool) _minters;
15     address _feeReceiver;
16     ERC20 _sand;
17
18     //@CTK NO_OVERFLOW
19     //@CTK NO_BUF_OVERFLOW
20     //@CTK NO_ASF
21     /*@CTK CommonMinter
22        @tag assume_completion
23        @post __post._sand == sand
24        @post __post._asset == asset
25        @post __post._feePerCopy == feePerCopy
26        @post __post._admin == admin
27        @post __post._feeReceiver == feeReceiver
28        @post __post._metaTransactionContracts[address(sand)] == true
29    */

```



```

30     constructor(ERC1155ERC721 asset, ERC20 sand, uint256 feePerCopy, address admin, address
31         feeReceiver)
32     public
33     {
34         _sand = sand;
35         _asset = asset;
36         _feePerCopy = feePerCopy;
37         _admin = admin;
38         _feeReceiver = feeReceiver;
39         _setMetaTransactionProcessor(address(sand), true);
40     }
41
42     /// @notice set the receiver of the proceeds
43     /// @param newFeeReceiver address of the new fee receiver
44     ///@CTK NO_OVERFLOW
45     ///@CTK NO_BUF_OVERFLOW
46     ///@CTK NO_ASF
47     /*@CTK setFeeReceiver_change
48         @tag assume_completion
49         @post msg.sender == _admin
50     */
51     /*@CTK setFeeReceiver_change
52         @tag assume_completion
53         @pre msg.sender == _admin
54         @post __post._feeReceiver == newFeeReceiver
55     */
56     function setFeeReceiver(address newFeeReceiver) external {
57         require(msg.sender == _admin, "only admin can change the receiver");
58         _feeReceiver = newFeeReceiver;
59     }
60
61     /// @notice set the fee in Sand for each common Asset copies
62     /// @param newFee new fee in Sand
63     ///@CTK NO_OVERFLOW
64     ///@CTK NO_BUF_OVERFLOW
65     ///@CTK NO_ASF
66     /*@CTK setFeePerCopy_change
67         @tag assume_completion
68         @post msg.sender == _admin
69     */
70     /*@CTK setFeePerCopy_change
71         @tag assume_completion
72         @pre msg.sender == _admin
73         @post __post._feePerCopy == newFee
74     */
75     function setFeePerCopy(uint256 newFee) external {
76         require(msg.sender == _admin, "only admin allowed to set fee");
77         _feePerCopy = newFee;
78     }
79
80     /// @notice mint common Asset token by paying the Sand fee
81     /// @param creator address creating the Asset, need to be the tx sender or meta tx
82     /// @param packId unused packId that will let you predict the resulting tokenId
83     /// @param hash cidv1 ipfs hash of the folder where 0.json file contains the metadata
84     /// @param supply number of copies to mint, cost in Sand is relative it it
85     /// @param owner address receiving the minted tokens
86     /// @param data extra data

```

```

86  /// @param feePerCopy fee in Sand for each copies
87  //@FIXME NO_OVERFLOW
88  //@FIXME NO_BUF_OVERFLOW
89  //@FIXME NO_ASF
90  /*@FIXME mintFor_require
91   @tag assume_completion
92   @post creator == msg.sender \/_metaTransactionContracts[msg.sender] == true
93   @post feePerCopy == _feePerCopy
94   */
95  function mintFor(
96      address creator,
97      uint40 packId,
98      bytes32 hash,
99      uint32 supply,
100     address owner,
101     bytes calldata data,
102     uint256 feePerCopy
103 ) external returns (uint256) {
104     require(creator == msg.sender || _metaTransactionContracts[msg.sender], "not
        authorized");
105     require(feePerCopy == _feePerCopy, "invalid fee");
106     require(_sand.transferFrom(creator, _feeReceiver, uint256(supply).mul(feePerCopy)),
        "failed to transfer SAND");
107     return _asset.mint(creator, packId, hash, supply, 0, owner, data);
108 }
109
110 /// @notice mint multiple common Asset tokena by paying the Sand fee
111 /// @param creator address creating the Asset, need to be the tx sender or meta tx
    signer
112 /// @param packId unused packId that will let you predict the resulting tokenId
113 /// @param hash cidv1 ipfs hash of the folder where 0.json file contains the metadata
114 /// @param supplies number of copies to mint for each Asset, cost in Sand is relative it
    it
115 /// @param owner address receiving the minted tokens
116 /// @param data extra data
117 /// @param feePerCopy fee in Sand for each copies
118 //@FIXME NO_OVERFLOW
119 //@FIXME NO_BUF_OVERFLOW
120 //@FIXME NO_ASF
121 /*@FIXME mintFor_require
122  @tag assume_completion
123  @post creator == msg.sender \/_metaTransactionContracts[msg.sender] == true
124  @post feePerCopy == _feePerCopy
125  */
126  function mintMultipleFor(
127      address creator,
128      uint40 packId,
129      bytes32 hash,
130      uint256[] calldata supplies,
131      address owner,
132      bytes calldata data,
133      uint256 feePerCopy
134 ) external returns (uint256[] memory ids) {
135     require(creator == msg.sender || _metaTransactionContracts[msg.sender], "not
        authorized");
136     require(feePerCopy == _feePerCopy, "invalid fee");
137     uint256 totalCopies = 0;
138     uint256 numAssetTypes = supplies.length;

```

```

139     for (uint256 i = 0; i < numAssetTypes; i++) {
140         totalCopies = totalCopies.add(supplies[i]);
141     }
142     require(_sand.transferFrom(creator, _feeReceiver, totalCopies.mul(feePerCopy)), "
        failed to transfer SAND");
143     return
144         _asset.mintMultiple(
145             creator,
146             packId,
147             hash,
148             supplies,
149             "",
150             owner,
151             data
152         );
153 }
154 }

```

File Asset.sol

```

1  pragma solidity 0.5.9;
2
3  import "../sandbox-private-contracts/src/Asset/ERC1155ERC721.sol";
4
5  contract Asset is ERC1155ERC721 {
6
7      //@CTK NO_OVERFLOW
8      //@CTK NO_BUF_OVERFLOW
9      //@CTK NO_ASF
10     /*@CTK Asset
11         @tag assume_completion
12         @post __post._metaTransactionContracts[metaTransactionContract] == true
13         @post __post._admin == assetAdmin
14         @post __post._bouncerAdmin == bouncerAdmin
15     */
16     constructor(
17         address metaTransactionContract,
18         address assetAdmin,
19         address bouncerAdmin
20     ) public ERC1155ERC721(metaTransactionContract, assetAdmin, bouncerAdmin) {}
21 }

```

File BundleSandSale.sol

```

1  pragma solidity 0.5.9;
2
3  import "../sandbox-private-contracts/contracts_common/src/Libraries/SafeMathWithRequire.sol";
4  import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC20.sol";
5  import "../sandbox-private-contracts/contracts_common/src/Interfaces/Medianizer.sol";
6  import "../sandbox-private-contracts/contracts_common/src/BaseWithStorage/Admin.sol";
7  import "../sandbox-private-contracts/src/Asset/ERC1155ERC721.sol";
8
9
10 contract BundleSandSale is Admin {
11
12     bytes4 private constant ERC1155_RECEIVED = 0xf23a6e61;
13     bytes4 private constant ERC1155_BATCH_RECEIVED = 0xbc197c81;
14
15     event BundleSale(

```

```

16     uint256 indexed saleId,
17     uint256[] ids,
18     uint256[] amounts,
19     uint256 sandAmount,
20     uint256 priceUSD,
21     uint256 numPacks
22 );
23
24 event BundleSold(
25     uint256 indexed saleId,
26     address indexed buyer,
27     uint256 numPacks,
28     address token,
29     uint256 tokenAmount
30 );
31
32 using SafeMathWithRequire for uint256;
33
34 Medianizer private _medianizer;
35 ERC20 private _dai;
36 ERC20 private _sand;
37 ERC1155ERC721 private _asset;
38
39 address payable private _receivingWallet;
40
41 struct Sale {
42     uint256[] ids;
43     uint256[] amounts;
44     uint256 sandAmount;
45     uint256 priceUSD;
46     uint256 numPacksLeft;
47 }
48
49 Sale[] private sales;
50
51 //@CTK NO_OVERFLOW
52 //@CTK NO_BUF_OVERFLOW
53 //@CTK NO_ASF
54 /*@CTK BundleSandSale_require
55   @tag assume_completion
56   @post receivingWallet != address(0)
57 */
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 */
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     }
83
84     /// @notice set the wallet receiving the proceeds
85     /// @param newWallet address of the new receiving wallet
86     ///@CTK NO_OVERFLOW
87     ///@CTK NO_BUF_OVERFLOW
88     ///@CTK NO_ASF
89     /*@CTK setReceivingWallet_require
90         @tag assume_completion
91         @post newWallet != address(0)
92         @post msg.sender == _admin
93     */
94     /*@CTK setReceivingWallet_change
95         @tag assume_completion
96         @pre newWallet != address(0)
97         @pre msg.sender == _admin
98         @post __post._receivingWallet == newWallet
99     */
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    }
105
106    ///@CTK NO_OVERFLOW
107    ///@CTK FAIL NO_BUF_OVERFLOW
108    ///@CTK NO_ASF
109    /*@CTK _transferPack
110        @tag assume_completion
111    */
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.
124                sales[saleIndex].amounts[j] == sales[saleIndex].amounts[j] * numPacks)
125            @post i == sales[saleIndex].ids.length
126        */
127        for (uint256 i = 0; i < numIds; i++) {
128            amounts[i] = amounts[i].mul(numPacks);
129        }
130        _asset.safeBatchTransferFrom(address(this), to, ids, amounts, "");
131    }

```

```

131
132 /**
133  * @notice Buys Sand Bundle with Ether
134  * @param saleId id of the bundle
135  * @param numPacks the amount of packs to buy
136  * @param to The address that will receive the SAND
137  */
138 //@CTK NO_OVERFLOW
139 //@CTK FAIL NO_BUF_OVERFLOW
140 //@CTK FAIL NO_ASF
141 /*@CTK buyBundleWithEther_require
142  @tag assume_completion
143  @post saleId > 0
144  @post sales[saleId - 1].numPacksLeft >= numPacks
145 */
146 function buyBundleWithEther(uint256 saleId, uint256 numPacks, address to) external
    payable {
147     require(saleId > 0, "invalid saleId");
148     uint256 saleIndex = saleId - 1;
149     uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
150     require(numPacksLeft >= numPacks, "not enough packs on sale");
151     sales[saleIndex].numPacksLeft = numPacksLeft - numPacks;
152
153     uint256 USDRequired = numPacks.mul(sales[saleIndex].priceUSD);
154     uint256 ETHRequired = getEtherAmountWithUSD(USDRequired);
155     require(msg.value >= ETHRequired, "not enough ether sent");
156     uint256 leftOver = msg.value - ETHRequired;
157     if(leftOver > 0) {
158         msg.sender.transfer(leftOver); // refund extra
159     }
160     address(_receivingWallet).transfer(ETHRequired);
161     _transferPack(saleIndex, numPacks, to);
162     emit BundleSold(saleId, msg.sender, numPacks, address(0), ETHRequired);
163 }
164
165 /**
166  * @notice Buys Sand Bundle with DAI
167  * @param saleId id of the bundle
168  * @param numPacks the amount of packs to buy
169  * @param to The address that will receive the SAND
170  */
171 //@CTK NO_OVERFLOW
172 //@CTK FAIL NO_BUF_OVERFLOW
173 //@CTK NO_ASF
174 /*@CTK buyBundleWithDai_require
175  @tag assume_completion
176  @post saleId > 0
177  @post sales[saleId - 1].numPacksLeft >= numPacks
178 */
179 function buyBundleWithDai(uint256 saleId, uint256 numPacks, address to) external {
180     require(saleId > 0, "invalid saleId");
181     uint256 saleIndex = saleId - 1;
182     uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
183     require(numPacksLeft >= numPacks, "not enough packs on sale");
184     sales[saleIndex].numPacksLeft = numPacksLeft - numPacks;
185
186     uint256 USDRequired = numPacks.mul(sales[saleIndex].priceUSD);
187     require(_dai.transferFrom(msg.sender, _receivingWallet, USDRequired), "failed to

```

```

    transfer dai");
188     _transferPack(saleIndex, numPacks, to);
189
190     emit BundleSold(saleId, msg.sender, numPacks, address(_dai), USDRequired);
191 }
192
193 //@CTK NO_OVERFLOW
194 //@CTK FAIL NO_BUF_OVERFLOW
195 //@CTK NO_ASF
196 /*@CTK getSaleInfo_require
197   @tag assume_completion
198   @post saleId > 0
199 */
200 /*@CTK getSaleInfo_change
201   @tag assume_completion
202   @pre saleId > 0
203   @post priceUSD == sales[saleId - 1].priceUSD
204   @post numPacksLeft == sales[saleId - 1].numPacksLeft
205 */
206 function getSaleInfo(uint256 saleId) external view returns(uint256 priceUSD, uint256
    numPacksLeft) {
207     require(saleId > 0, "invalid saleId");
208     uint256 saleIndex = saleId - 1;
209     priceUSD = sales[saleIndex].priceUSD;
210     numPacksLeft = sales[saleIndex].numPacksLeft;
211 }
212
213 //@CTK NO_OVERFLOW
214 //@CTK FAIL NO_BUF_OVERFLOW
215 //@CTK NO_ASF
216 /*@CTK withdrawSale_require
217   @tag assume_completion
218   @post msg.sender == _admin
219   @post saleId > 0
220 */
221 /*@CTK withdrawSale_change
222   @tag assume_completion
223   @pre msg.sender == _admin
224   @pre saleId > 0
225   @post __post.sales[saleId-1].numPacksLeft == 0
226 */
227 function withdrawSale(uint256 saleId, address to) external onlyAdmin() {
228     require(saleId > 0, "invalid saleId");
229     uint256 saleIndex = saleId - 1;
230     uint256 numPacksLeft = sales[saleIndex].numPacksLeft;
231     sales[saleIndex].numPacksLeft = 0;
232
233     uint256[] memory ids = sales[saleIndex].ids;
234     uint256[] memory amounts = sales[saleIndex].amounts;
235     uint256 numIds = ids.length;
236     /*@FIXME FAIL "withdrawSale_loop"
237       @inv i <= sales[saleId - 1].ids.length
238       @post forall j: uint256. (j >= 0 /\ j < sales[saleId - 1].ids.length) -> (__post.
239         sales[saleId - 1].amounts[j] == sales[saleId - 1].amounts[j] * sales[saleId -
240         1].numPacksLeft)
241       @post i == sales[saleId - 1].ids.length
242     */
243     for (uint256 i = 0; i < numIds; i++) {

```

```

242     amounts[i] = amounts[i].mul(numPacksLeft);
243 }
244 require(_sand.transferFrom(address(this), to, numPacksLeft.mul(sales[saleIndex].
    sandAmount)), "transfer fo Sand failed");
245 _asset.safeBatchTransferFrom(address(this), to, ids, amounts, "");
246 }
247
248 /**
249  * @notice Returns the amount of ETH for a specific amount of USD
250  * @param usdAmount An amount of USD
251  * @return The amount of ETH
252  */
253 // @CTK NO_OVERFLOW
254 // @CTK NO_BUF_OVERFLOW
255 // @CTK FAIL NO_ASF
256 function getEtherAmountWithUSD(uint256 usdAmount) public view returns (uint256) {
257     uint256 ethUsdPair = getEthUsdPair();
258     return usdAmount.mul(1000000000000000000).div(ethUsdPair);
259 }
260
261 /**
262  * @notice Gets the ETHUSD pair from the Medianizer contract
263  * @return The pair as an uint256
264  */
265 function getEthUsdPair() internal view returns (uint256) {
266     bytes32 pair = _medianizer.read();
267     return uint256(pair);
268 }
269
270 function onERC1155Received(
271     address operator,
272     address from,
273     uint256 id,
274     uint256 value,
275     bytes calldata data
276 ) external returns (bytes4) {
277     require(
278         address(_asset) == msg.sender,
279         "only accept asset as sender"
280     );
281     require(from == operator, "only self executed transfer allowed");
282     require(value > 0, "no Asset transfered");
283     require(data.length > 0, "data need to contains the sale data");
284
285     (
286         uint256 numPacks,
287         uint256 sandAmountPerPack,
288         uint256 priceUSDPerPack
289     ) = abi.decode(data, (uint256, uint256, uint256));
290
291     uint256 amount = value.div(numPacks);
292     require(amount.mul(numPacks) == value, "invalid amounts, not divisible by numPacks");
293
294     ;
295     uint256[] memory amounts = new uint256[](1);
296     amounts[0] = amount;
297     uint256[] memory ids = new uint256[](1);
298     ids[0] = id;
299     _setupBundle(from, sandAmountPerPack, numPacks, ids, amounts, priceUSDPerPack);

```



```

298     return ERC1155_RECEIVED;
299 }
300 function onERC1155BatchReceived(
301     address operator,
302     address from,
303     uint256[] calldata ids,
304     uint256[] calldata values,
305     bytes calldata data
306 ) external returns (bytes4) {
307     require(
308         address(_asset) == msg.sender,
309         "only accept asset as sender"
310     );
311     require(from == operator, "only self executed transfer allowed");
312     require(ids.length > 0, "need to contains Asset");
313     require(data.length > 0, "data need to contains the sale data");
314
315     (
316         uint256 numPacks,
317         uint256 sandAmountPerPack,
318         uint256 priceUSDPerPack
319     ) = abi.decode(data, (uint256, uint256, uint256));
320
321     uint256[] memory amounts = new uint256[](ids.length); // TODO
322     for(uint256 i = 0; i < amounts.length; i++) {
323         require(values[i] > 0, "asset transfer with zero values");
324         uint256 amount = values[i].div(numPacks);
325         require(amount.mul(numPacks) == values[i], "invalid amounts, not divisible by
326             numPacks");
327         amounts[i] = amount;
328     }
329
330     _setupBundle(from, sandAmountPerPack, numPacks, ids, amounts, priceUSDPerPack);
331     return ERC1155_BATCH_RECEIVED;
332 }
333 function _setupBundle(
334     address from,
335     uint256 sandAmountPerPack,
336     uint256 numPacks,
337     uint256[] memory ids,
338     uint256[] memory amounts,
339     uint256 priceUSDPerPack
340 ) internal {
341     require(_sand.transferFrom(from, address(this), sandAmountPerPack.mul(numPacks)), "
342         failed to transfer Sand");
343     uint256 saleId = sales.push(Sale({
344         ids: ids,
345         amounts : amounts,
346         sandAmount: sandAmountPerPack,
347         priceUSD: priceUSDPerPack,
348         numPacksLeft: numPacks
349     }));
350     emit BundleSale(saleId, ids, amounts, sandAmountPerPack, priceUSDPerPack, numPacks);
351 }

```

File TheSandbox712.sol

```
1 pragma solidity 0.5.9;
```

```

2
3 import "../sandbox-private-contracts/contracts_common/src/BaseWithStorage/
  ProxyImplementation.sol";
4
5 contract TheSandbox712 is ProxyImplementation {
6     bytes32 constant EIP712DOMAIN_TYPEHASH = keccak256(
7         "EIP712Domain(string name,string version,address verifyingContract)"
8     );
9     bytes32 DOMAIN_SEPARATOR;
10
11     //@CTK NO_OVERFLOW
12     //@CTK NO_BUF_OVERFLOW
13     //@CTK NO_ASF
14     /*@CTK init712
15         @tag assume_completion
16         @post __post._initialised["712"] == true
17     */
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     }
28
29     //@CTK NO_OVERFLOW
30     //@CTK NO_BUF_OVERFLOW
31     //@CTK NO_ASF
32     function domainSeparator() internal view returns (bytes32) {
33         return DOMAIN_SEPARATOR;
34     }
35 }

```

File AssetSignedAuction.sol

```

1 pragma solidity 0.5.9;
2
3 import "../sandbox-private-contracts/contracts_common/src/Libraries/SigUtil.sol";
4 import "../sandbox-private-contracts/contracts_common/src/Libraries/PriceUtil.sol";
5 import "../sandbox-private-contracts/src/Sand.sol";
6 import "../sandbox-private-contracts/src/Asset.sol";
7 import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC20.sol";
8 import "../sandbox-private-contracts/src/TheSandbox712.sol";
9 import "../sandbox-private-contracts/contracts_common/src/BaseWithStorage/
  MetaTransactionReceiver.sol";
10 import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC1271.sol";
11 import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC1271Constants.sol";
12 import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC1654.sol";
13 import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC1654Constants.sol";
14 import "../sandbox-private-contracts/contracts_common/src/Libraries/SafeMathWithRequire.sol";
15
16 contract AssetSignedAuction is ERC1654Constants, ERC1271Constants, TheSandbox712,
  MetaTransactionReceiver {
17     using SafeMathWithRequire for uint256;
18

```

```

19  enum SignatureType { DIRECT, EIP1654, EIP1271 }
20
21  bytes32 constant AUCTION_TYPEHASH = keccak256(
22      "Auction(address from,address token,uint256 offerId,uint256 startingPrice,uint256
        endingPrice,uint256 startedAt,uint256 duration,uint256 packs,bytes ids,bytes
        amounts)"
23  );
24
25  event OfferClaimed(
26      address indexed seller,
27      address indexed buyer,
28      uint256 indexed offerId,
29      uint256 amount,
30      uint256 pricePaid,
31      uint256 feePaid
32  );
33  event OfferCancelled(address indexed seller, uint256 indexed offerId);
34
35  uint256 constant MAX_UINT256 = 0
        xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff;
36
37  // Stack too deep, grouping parameters
38  // AuctionData:
39  uint256 constant AuctionData_OfferId = 0;
40  uint256 constant AuctionData_StartingPrice = 1;
41  uint256 constant AuctionData_EndingPrice = 2;
42  uint256 constant AuctionData_StartedAt = 3;
43  uint256 constant AuctionData_Duration = 4;
44  uint256 constant AuctionData_Packs = 5;
45
46  mapping(address => mapping(uint256 => uint256)) claimed;
47
48  Asset _asset;
49  uint256 _fee10000th = 0;
50  address payable _feeCollector;
51
52  event FeeSetup(address feeCollector, uint256 fee10000th);
53
54  // @CTK NO_OVERFLOW
55  // @CTK NO_BUF_OVERFLOW
56  // @CTK NO_ASF
57  /* @CTK AssetSignedAuction
58      @post __post._asset == asset
59      @post __post._feeCollector == feeCollector
60      @post __post._fee10000th == fee10000th
61      @post __post._admin == admin
62      @post __post._metaTransactionContracts[initialMetaTx] == true
63  */
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  }

```

```

73
74    /// @notice set fee parameters
75    /// @param feeCollector address receiving the fee
76    /// @param fee10000th fee in 10,000th
77    ///@CTK NO_OVERFLOW
78    ///@CTK NO_BUF_OVERFLOW
79    ///@CTK NO_ASF
80    /*@CTK setFee_require
81        @tag assume_completion
82        @post msg.sender == _admin
83    */
84    /*@CTK setFee_change
85        @tag assume_completion
86        @pre msg.sender == _admin
87        @post __post._feeCollector == feeCollector
88        @post __post._fee10000th == fee10000th
89    */
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    }
96
97    ///@CTK NO_OVERFLOW
98    ///@CTK FAIL NO_BUF_OVERFLOW
99    ///@CTK NO_ASF
100    /*@CTK _verifyParameters
101        @tag assume_completion
102        @post ids.length == amounts.length
103        @post buyer == msg.sender /\ (token != address(0) /\ _metaTransactionContracts[msg.
104            sender])
105        @post claimed[seller][auctionData[AuctionData_OfferId]] != MAX_UINT256
106        @post claimed[seller][auctionData[AuctionData_OfferId]] + buyAmount <= auctionData[
107            AuctionData_Packs]
108        @post auctionData[AuctionData_StartedAt] <= block.timestamp
109        @post auctionData[AuctionData_StartedAt] + auctionData[AuctionData_Duration] > block.
110            timestamp
111    */
112    function _verifyParameters(
113        address buyer,
114        address payable seller,
115        address token,
116        uint256 buyAmount,
117        uint256[] memory auctionData,
118        uint256[] memory ids,
119        uint256[] memory amounts
120    ) internal view {
121        require(ids.length == amounts.length, "ids and amounts length not matching");
122        require(buyer == msg.sender || (token != address(0) && _metaTransactionContracts[msg.
123            sender]), "not authorized");
124        uint256 amountAlreadyClaimed = claimed[seller][auctionData[AuctionData_OfferId]];
125        require(amountAlreadyClaimed != MAX_UINT256, "Auction cancelled");
126
127        uint256 total = amountAlreadyClaimed.add(buyAmount);
128        require(total >= amountAlreadyClaimed, "overflow");
129        require(total <= auctionData[AuctionData_Packs], "Buy amount exceeds sell amount");

```

```

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
133             .timestamp,
134         "Auction finished"
135     );
136 }
137
138 /// @notice claim offer using EIP712
139 /// @param buyer address paying for the offer
140 /// @param seller address of the seller
141 /// @param token token used for payment
142 /// @param purchase buyAmount, maxTokenAmount
143 /// @param auctionData offerId, startingPrice, endingPrice, startedAt, duration, packs
144 /// @param ids ids of the Assets being sold
145 /// @param amounts amounts of Assets per pack
146 /// @param signature signature of seller
147 function claimSellerOffer(
148     address buyer,
149     address payable seller,
150     address token,
151     uint256[] calldata purchase, // buyAmount, maxTokenAmount
152     uint256[] calldata auctionData,
153     uint256[] calldata ids,
154     uint256[] calldata amounts,
155     bytes calldata signature
156 ) external payable {
157     _verifyParameters(
158         buyer,
159         seller,
160         token,
161         purchase[0],
162         auctionData,
163         ids,
164         amounts
165     );
166     _ensureCorrectSigner(seller, token, auctionData, ids, amounts, signature,
167         SignatureType.DIRECT, true);
168     _executeDeal(
169         token,
170         purchase,
171         buyer,
172         seller,
173         auctionData,
174         ids,
175         amounts
176     );
177 }
178
179 /// @notice claim offer using EIP712 and EIP1271 signature verification scheme
180 /// @param buyer address paying for the offer
181 /// @param seller address of the seller
182 /// @param token token used for payment
183 /// @param purchase buyAmount, maxTokenAmount
184 /// @param auctionData offerId, startingPrice, endingPrice, startedAt, duration, packs

```

```

183  /// @param ids ids of the Assets being sold
184  /// @param amounts amounts of Assets per pack
185  /// @param signature signature of seller
186  function claimSellerOfferViaEIP1271(
187      address buyer,
188      address payable seller,
189      address token,
190      uint256[] calldata purchase, // buyAmount, maxTokenAmount
191      uint256[] calldata auctionData,
192      uint256[] calldata ids,
193      uint256[] calldata amounts,
194      bytes calldata signature
195  ) external payable {
196      _verifyParameters(
197          buyer,
198          seller,
199          token,
200          purchase[0],
201          auctionData,
202          ids,
203          amounts
204      );
205      _ensureCorrectSigner(seller, token, auctionData, ids, amounts, signature,
206          SignatureType.EIP1271, true);
207      _executeDeal(
208          token,
209          purchase,
210          buyer,
211          seller,
212          auctionData,
213          ids,
214          amounts
215      );
216  }
217  /// @notice claim offer using EIP712 and EIP1654 signature verification scheme
218  /// @param buyer address paying for the offer
219  /// @param seller address of the seller
220  /// @param token token used for payment
221  /// @param purchase buyAmount, maxTokenAmount
222  /// @param auctionData offerId, startingPrice, endingPrice, startedAt, duration, packs
223  /// @param ids ids of the Assets being sold
224  /// @param amounts amounts of Assets per pack
225  /// @param signature signature of seller
226  function claimSellerOfferViaEIP1654(
227      address buyer,
228      address payable seller,
229      address token,
230      uint256[] calldata purchase, // buyAmount, maxTokenAmount
231      uint256[] calldata auctionData,
232      uint256[] calldata ids,
233      uint256[] calldata amounts,
234      bytes calldata signature
235  ) external payable {
236      _verifyParameters(
237          buyer,
238          seller,
239          token,

```

```

240         purchase[0],
241         auctionData,
242         ids,
243         amounts
244     );
245     _ensureCorrectSigner(seller, token, auctionData, ids, amounts, signature,
        SignatureType.EIP1654, true);
246     _executeDeal(
247         token,
248         purchase,
249         buyer,
250         seller,
251         auctionData,
252         ids,
253         amounts
254     );
255 }
256
257 /// @notice claim offer using Basic Signature
258 /// @param buyer address paying for the offer
259 /// @param seller address of the seller
260 /// @param token token used for payment
261 /// @param purchase buyAmount, maxTokenAmount
262 /// @param auctionData offerId, startingPrice, endingPrice, startedAt, duration, packs
263 /// @param ids ids of the Assets being sold
264 /// @param amounts amounts of Assets per pack
265 /// @param signature signature of seller
266 function claimSellerOfferUsingBasicSig(
267     address buyer,
268     address payable seller,
269     address token,
270     uint256[] calldata purchase, // buyAmount, maxTokenAmount
271     uint256[] calldata auctionData,
272     uint256[] calldata ids,
273     uint256[] calldata amounts,
274     bytes calldata signature
275 ) external payable {
276     _verifyParameters(
277         buyer,
278         seller,
279         token,
280         purchase[0],
281         auctionData,
282         ids,
283         amounts
284     );
285     _ensureCorrectSigner(seller, token, auctionData, ids, amounts, signature,
        SignatureType.DIRECT, false);
286     _executeDeal(
287         token,
288         purchase,
289         buyer,
290         seller,
291         auctionData,
292         ids,
293         amounts
294     );
295 }

```

```

296
297 /// @notice claim offer using Basic Signature and EIP1271 signature verification scheme
298 /// @param buyer address paying for the offer
299 /// @param seller address of the seller
300 /// @param token token used for payment
301 /// @param purchase buyAmount, maxTokenAmount
302 /// @param auctionData offerId, startingPrice, endingPrice, startedAt, duration, packs
303 /// @param ids ids of the Assets being sold
304 /// @param amounts amounts of Assets per pack
305 /// @param signature signature of seller
306 function claimSellerOfferUsingBasicSigViaEIP1271(
307     address buyer,
308     address payable seller,
309     address token,
310     uint256[] calldata purchase, // buyAmount, maxTokenAmount
311     uint256[] calldata auctionData,
312     uint256[] calldata ids,
313     uint256[] calldata amounts,
314     bytes calldata signature
315 ) external payable {
316     _verifyParameters(
317         buyer,
318         seller,
319         token,
320         purchase[0],
321         auctionData,
322         ids,
323         amounts
324     );
325     _ensureCorrectSigner(seller, token, auctionData, ids, amounts, signature,
326         SignatureType.EIP1271, false);
327     _executeDeal(
328         token,
329         purchase,
330         buyer,
331         seller,
332         auctionData,
333         ids,
334         amounts
335     );
336 }
337
338 /// @notice claim offer using Basic Signature and EIP1654 signature verification scheme
339 /// @param buyer address paying for the offer
340 /// @param seller address of the seller
341 /// @param token token used for payment
342 /// @param purchase buyAmount, maxTokenAmount
343 /// @param auctionData offerId, startingPrice, endingPrice, startedAt, duration, packs
344 /// @param ids ids of the Assets being sold
345 /// @param amounts amounts of Assets per pack
346 /// @param signature signature of seller
347 function claimSellerOfferUsingBasicSigViaEIP1654(
348     address buyer,
349     address payable seller,
350     address token,
351     uint256[] calldata purchase, // buyAmount, maxTokenAmount
352     uint256[] calldata auctionData,
353     uint256[] calldata ids,

```



```

353     uint256[] calldata amounts,
354     bytes calldata signature
355 ) external payable {
356     _verifyParameters(
357         buyer,
358         seller,
359         token,
360         purchase[0],
361         auctionData,
362         ids,
363         amounts
364     );
365     _ensureCorrectSigner(seller, token, auctionData, ids, amounts, signature,
366         SignatureType.EIP1654, false);
367     _executeDeal(
368         token,
369         purchase,
370         buyer,
371         seller,
372         auctionData,
373         ids,
374         amounts
375     );
376 }
377 //@FIXME NO_OVERFLOW
378 //@FIXME NO_BUF_OVERFLOW
379 //@FIXME NO_ASF
380 /*@FIXME _executeDeal_require
381    @tag assume_completion
382 */
383 function _executeDeal(
384     address token,
385     uint256[] memory purchase,
386     address buyer,
387     address payable seller,
388     uint256[] memory auctionData,
389     uint256[] memory ids,
390     uint256[] memory amounts
391 ) internal {
392     uint256 offer = PriceUtil.calculateCurrentPrice(
393         auctionData[AuctionData_StartingPrice],
394         auctionData[AuctionData_EndingPrice],
395         auctionData[AuctionData_Duration],
396         block.timestamp.sub(auctionData[AuctionData_StartedAt])
397     ).mul(purchase[0]);
398     claimed[seller][auctionData[AuctionData_OfferId]] = claimed[seller][auctionData[
399         AuctionData_OfferId]].add(purchase[0]);
400     uint256 fee = 0;
401     if(_fee10000th > 0) {
402         fee = PriceUtil.calculateFee(offer, _fee10000th);
403     }
404     uint256 total = offer.add(fee);
405     require(total <= purchase[1], "offer exceeds max amount to spend");
406     if (token != address(0)) {

```

```

409         require(ERC20(token).transferFrom(buyer, seller, offer), "failed to transfer
           token price");
410     if(fee > 0) {
411         require(ERC20(token).transferFrom(buyer, _feeCollector, fee), "failed to
           collect fee");
412     }
413 } else {
414     require(msg.value >= total, "ETH < offer+fee");
415     if(msg.value > total) {
416         msg.sender.transfer(msg.value.sub(total));
417     }
418     seller.transfer(offer);
419     if(fee > 0) {
420         _feeCollector.transfer(fee);
421     }
422 }
423
424 uint256[] memory packAmounts = new uint256[](amounts.length);
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 }
435 _asset.safeBatchTransferFrom(seller, buyer, ids, packAmounts, "");
436 emit OfferClaimed(
437     seller,
438     buyer,
439     auctionData[AuctionData_OfferId],
440     purchase[0],
441     offer,
442     fee
443 );
444 }
445
446 /// @notice cancel a offer previously signed, new offer need to use a id not used yet
447 /// @param offerId offer to cancel
448 ///@CTK NO_OVERFLOW
449 ///@CTK NO_BUF_OVERFLOW
450 ///@CTK NO_ASF
451 /*@CTK cancelSellerOffer
452    @post __post.claimed[msg.sender][offerId] == MAX_UINT256
453    */
454 function cancelSellerOffer(uint256 offerId) external {
455     claimed[msg.sender][offerId] = MAX_UINT256;
456     emit OfferCancelled(msg.sender, offerId);
457 }
458
459 function _ensureCorrectSigner(
460     address from,
461     address token,
462     uint256[] memory auctionData,
463     uint256[] memory ids,

```

```

464     uint256[] memory amounts,
465     bytes memory signature,
466     SignatureType signatureType,
467     bool eip712
468 ) internal view returns (address) {
469     bytes memory dataToHash;
470
471     if(eip712) {
472         dataToHash = abi.encodePacked(
473             "\x19\x01",
474             domainSeparator(),
475             _hashAuction(from, token, auctionData, ids, amounts)
476         );
477     } else {
478         dataToHash = _encodeBasicSignatureHash(from, token, auctionData, ids, amounts);
479     }
480
481     if (signatureType == SignatureType.EIP1271) {
482         require(
483             ERC1271(from).isValidSignature(dataToHash, signature) == ERC1271_MAGICVALUE,
484             "invalid 1271 signature"
485         );
486     } else if (signatureType == SignatureType.EIP1654){
487         require(
488             ERC1654(from).isValidSignature(keccak256(dataToHash), signature) ==
489             ERC1654_MAGICVALUE,
490             "invalid 1654 signature"
491         );
492     } else {
493         address signer = SigUtil.recover(keccak256(dataToHash), signature);
494         require(signer == from, "signer != from");
495     }
496
497     function _encodeBasicSignatureHash(
498         address from,
499         address token,
500         uint256[] memory auctionData,
501         uint256[] memory ids,
502         uint256[] memory amounts
503     ) internal view returns (bytes memory) {
504         return SigUtil.prefixed(keccak256(abi.encodePacked(
505             address(this),
506             AUCTION_TYPEHASH,
507             from,
508             token,
509             auctionData[AuctionData_OfferId],
510             auctionData[AuctionData_StartingPrice],
511             auctionData[AuctionData_EndingPrice],
512             auctionData[AuctionData_StartedAt],
513             auctionData[AuctionData_Duration],
514             auctionData[AuctionData_Packs],
515             keccak256(abi.encodePacked(ids)),
516             keccak256(abi.encodePacked(amounts))
517         )));
518     }
519
520     function _hashAuction(

```

```
521     address from,
522     address token,
523     uint256[] memory auctionData,
524     uint256[] memory ids,
525     uint256[] memory amounts
526 ) internal pure returns (bytes32) {
527     return
528         keccak256(
529             abi.encode(
530                 AUCTION_TYPEHASH,
531                 from,
532                 token,
533                 auctionData[AuctionData_OfferId],
534                 auctionData[AuctionData_StartingPrice],
535                 auctionData[AuctionData_EndingPrice],
536                 auctionData[AuctionData_StartedAt],
537                 auctionData[AuctionData_Duration],
538                 auctionData[AuctionData_Packs],
539                 keccak256(abi.encodePacked(ids)),
540                 keccak256(abi.encodePacked(amounts))
541             )
542         );
543 }
544 }
```

