



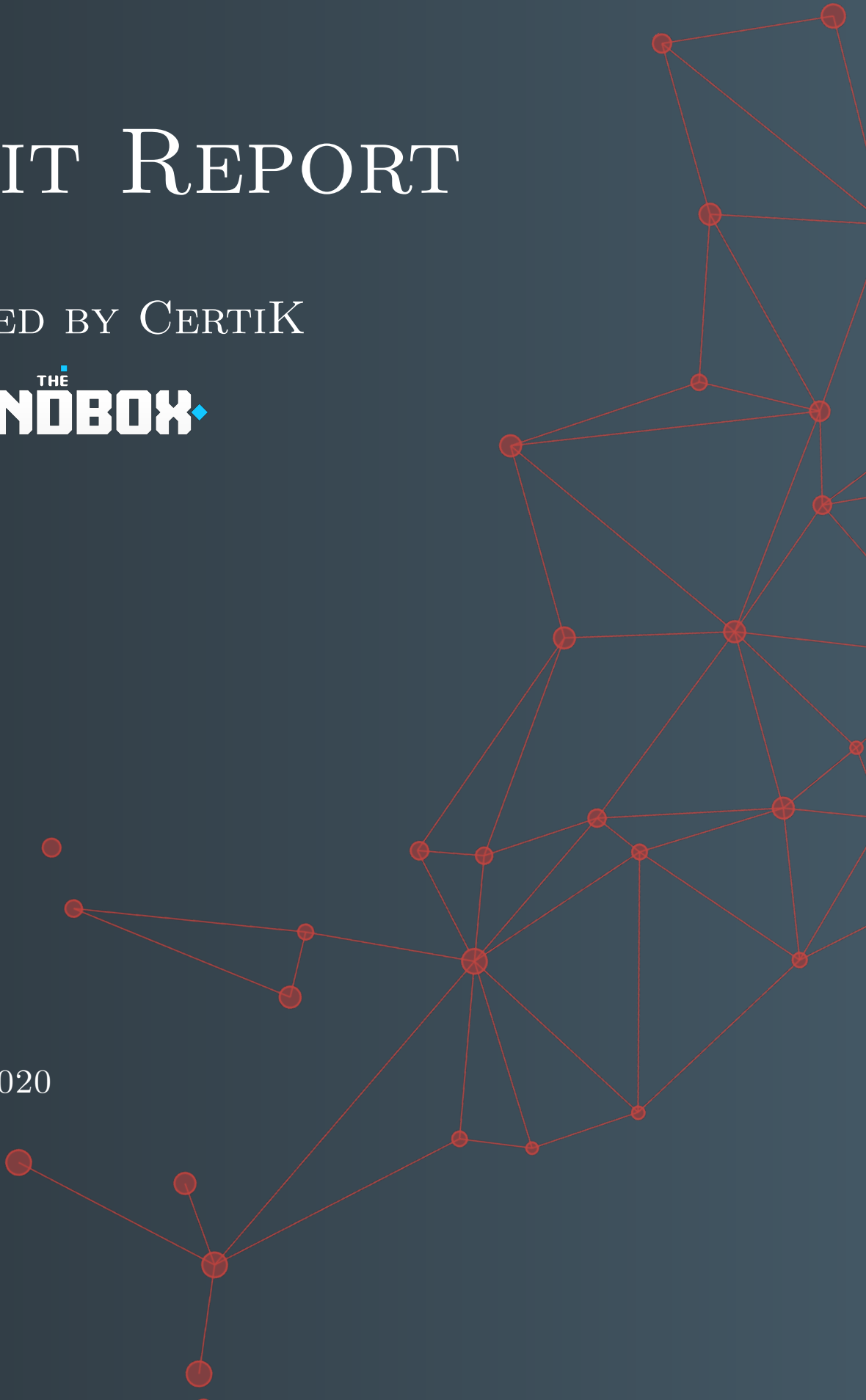
CERTIK

# AUDIT REPORT

PRODUCED BY CERTIK

FOR **THE SANDBOX**

20<sup>TH</sup> FEB, 2020



CERTIK AUDIT REPORT  
FOR THE SANDBOX



Request Date: 2020-01-31  
Revision Date: 2020-02-20  
Platform Name: Ethereum



# Contents

<b>Disclaimer</b>	<b>1</b>
<b>About CertiK</b>	<b>2</b>
<b>Executive Summary</b>	<b>3</b>
<b>Vulnerability Classification</b>	<b>3</b>
<b>Testing Summary</b>	<b>4</b>
Audit Score . . . . .	4
Type of Issues . . . . .	4
Vulnerability Details . . . . .	5
<b>Manual Review Notes</b>	<b>6</b>
<b>Static Analysis Results</b>	<b>7</b>
<b>Formal Verification Results</b>	<b>8</b>
How to read . . . . .	8
<b>Source Code with CertiK Labels</b>	<b>46</b>

## 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 LandBaseToken, Land and LandSaleWithETHAndDAI 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.

Feb 20, 2020



## 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/Underflow	An overflow/underflow occurs when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function Incorrectness	Function implementation does not meet specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker can write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by miners to some degree.	1	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.	1	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. msg.sender instead.	Use 0	SWC-115

Title	Description	Issues	SWC ID
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	SWC-131

## Vulnerability Details

### Critical

No issue found.

### Medium

No issue found.

### Low

No issue found.



# Manual Review Notes

## Source Code SHA-256 Checksum<sup>1</sup>

- **LandSaleWithReferral.sol**  
cb97e3bbb6927783a8ff10fcfb89bbd931fb78f3016142afcf15d007c6b04f81
- **ReferralValidator.sol**  
366423893bc8823e336d2febcb356fb4071b44c98e57ea9a9e9ce454ee8693901
- **SigUtil.sol**  
9037b60bdf791be547bfe03aae9443ab2e5e76b2e16fb8be41acc4982965b9af

## 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.

### **ReferralValidator.sol** commit 3dc6d2f6434a26c7a62a05051f409dc60472c9c0

1. **INFO** constructor() and updateMaxCommissionRate(): Since `_maxCommissionRate` is important, recommend adding checks to make sure `_maxCommissionRate` is set in a reasonable interval.
2. **INFO** handleReferralWithETH() and handleReferralWithERC20(): Recommend adding a check to make sure `destination != address(0)`.
3. **INFO** handleReferralWithERC20(): It's not necessary to specify the parameter `destination` as `payable`.

### **SigUtil.sol** commit 3dc6d2f6434a26c7a62a05051f409dc60472c9c0

1. **INFO** recover(): Recommend adding error messages for `require()` calls.

---

<sup>1</sup>Commit: 3dc6d2f6434a26c7a62a05051f409dc60472c9c0

## Static Analysis Results

### INSECURE\_COMPILER\_VERSION

Line 1 in File SigUtil.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 3 in File LandSaleWithReferral.sol

```
3 pragma solidity 0.5.9;
```

**!** Version to compile has the following bug: 0.5.9: SignedArrayStorageCopy, ABIEncoderV2StorageArrayV

### TIMESTAMP\_DEPENDENCY

Line 196 in File LandSaleWithReferral.sol

```
196 require(block.timestamp < _expiryTime, "sale is over");
```

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

### INSECURE\_COMPILER\_VERSION

Line 3 in File ReferralValidator.sol

```
3 pragma solidity 0.5.9;
```

**!** Version to compile has the following bug: 0.5.9: SignedArrayStorageCopy, ABIEncoderV2StorageArrayV

### TIMESTAMP\_DEPENDENCY

Line 61 in File ReferralValidator.sol

```
61 _previousSigningWallets[_signingWallet] = now + _previousSigningDelay;
```

**!** "now" can be influenced by miners to some degree

### TIMESTAMP\_DEPENDENCY

Line 209 in File ReferralValidator.sol

```
209 if (commissionRate > _maxCommissionRate || referrer == referee || now > expiryTime) {
```




**!** "now" 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.

 20, Feb 2020

 40.95 ms

Line 4 in File SigUtil.sol

```
4 // @CTK_NO_OVERFLOW
```

Line 7-33 in File SigUtil.sol


```
7 function recover(bytes32 hash, bytes memory sig)
8     internal
9     pure
10    returns (address recovered)
11    {
12        require(sig.length == 65);
13
14        bytes32 r;
15        bytes32 s;
16        uint8 v;
17        assembly {
18            r := mload(add(sig, 32))
19            s := mload(add(sig, 64))
20            v := byte(0, mload(add(sig, 96)))
21        }
22
23        // Version of signature should be 27 or 28, but 0 and 1 are also possible versions
24        if (v < 27) {
25            v += 27;
26        }
27        require(v == 27 || v == 28);
28
29        recovered = ecrecover(hash, v, r, s);
30        require(recovered != address(0));
31    }
```

 The code meets the specification.

## Formal Verification Request 2

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

 20, Feb 2020

 4.02 ms

Line 5 in File SigUtil.sol

```
5 // @CTK_NO_BUF_OVERFLOW
```

Line 7-33 in File SigUtil.sol

```
7 function recover(bytes32 hash, bytes memory sig)
8     internal
9     pure
10    returns (address recovered)
11    {
```

```

12     require(sig.length == 65);
13
14     bytes32 r;
15     bytes32 s;
16     uint8 v;
17     assembly {
18         r := mload(add(sig, 32))
19         s := mload(add(sig, 64))
20         v := byte(0, mload(add(sig, 96)))
21     }
22
23     // Version of signature should be 27 or 28, but 0 and 1 are also possible versions
24     if (v < 27) {
25         v += 27;
26     }
27     require(v == 27 || v == 28);
28
29     recovered = ecrecover(hash, v, r, s);
30     require(recovered != address(0));
31 }

```

✔ The code meets the specification.

## Formal Verification Request 3

Method will not encounter an assertion failure.

📅 20, Feb 2020

🕒 3.78 ms

Line 6 in File SigUtil.sol

```
6 // @CTK NO_ASF
```

Line 7-33 in File SigUtil.sol

```

7     function recover(bytes32 hash, bytes memory sig)
8         internal
9         pure
10        returns (address recovered)
11    {
12        require(sig.length == 65);
13
14        bytes32 r;
15        bytes32 s;
16        uint8 v;
17        assembly {
18            r := mload(add(sig, 32))
19            s := mload(add(sig, 64))
20            v := byte(0, mload(add(sig, 96)))
21        }
22
23        // Version of signature should be 27 or 28, but 0 and 1 are also possible versions
24        if (v < 27) {
25            v += 27;
26        }
27        require(v == 27 || v == 28);
28

```

```

29     recovered = ecrecover(hash, v, r, s);
30     require(recovered != address(0));
31 }


```

✔ The code meets the specification.

## Formal Verification Request 4

If method completes, integer overflow would not happen.

 20, Feb 2020

 18.04 ms

Line 35 in File SigUtil.sol

```

35     // @CTK_NO_OVERFLOW

```

Line 38-68 in File SigUtil.sol

```

38     function recoverWithZeroOnFailure(bytes32 hash, bytes memory sig)
39         internal
40         pure
41         returns (address)
42     {
43         if (sig.length != 65) {
44             return address(0);
45         }
46
47         bytes32 r;
48         bytes32 s;
49         uint8 v;
50         assembly {
51             r := mload(add(sig, 32))
52             s := mload(add(sig, 64))
53             v := byte(0, mload(add(sig, 96)))
54         }
55
56         // Version of signature should be 27 or 28, but 0 and 1 are also possible versions
57         if (v < 27) {
58             v += 27;
59         }
60
61         if (v != 27 && v != 28) {
62             return address(0);
63         } else {
64             return ecrecover(hash, v, r, s);
65         }
66     }


```

✔ The code meets the specification.

## Formal Verification Request 5

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

 20, Feb 2020

 0.61 ms

Line 36 in File SigUtil.sol

```
36 // @CTK_NO_BUF_OVERFLOW
```

Line 38-68 in File SigUtil.sol


```
38 function recoverWithZeroOnFailure(bytes32 hash, bytes memory sig)
39     internal
40     pure
41     returns (address)
42 {
43     if (sig.length != 65) {
44         return address(0);
45     }
46
47     bytes32 r;
48     bytes32 s;
49     uint8 v;
50     assembly {
51         r := mload(add(sig, 32))
52         s := mload(add(sig, 64))
53         v := byte(0, mload(add(sig, 96)))
54     }
55
56     // Version of signature should be 27 or 28, but 0 and 1 are also possible versions
57     if (v < 27) {
58         v += 27;
59     }
60
61     if (v != 27 && v != 28) {
62         return address(0);
63     } else {
64         return ecrecover(hash, v, r, s);
65     }
66 }
```

✔ The code meets the specification.

## Formal Verification Request 6

Method will not encounter an assertion failure.

 20, Feb 2020

 0.49 ms

Line 37 in File SigUtil.sol

```
37 // @CTK_NO_ASF
```

Line 38-68 in File SigUtil.sol

```
38 function recoverWithZeroOnFailure(bytes32 hash, bytes memory sig)
39     internal
40     pure
41     returns (address)
42 {
43     if (sig.length != 65) {
44         return address(0);
```

```

45     }
46
47     bytes32 r;
48     bytes32 s;
49     uint8 v;
50     assembly {
51         r := mload(add(sig, 32))
52         s := mload(add(sig, 64))
53         v := byte(0, mload(add(sig, 96)))
54     }
55
56     // Version of signature should be 27 or 28, but 0 and 1 are also possible versions
57     if (v < 27) {
58         v += 27;
59     }
60
61     if (v != 27 && v != 28) {
62         return (address(0));
63     } else {
64         return ecrecover(hash, v, r, s);
65     }
66 }

```

✔ The code meets the specification.

## Formal Verification Request 7

If method completes, integer overflow would not happen.

📅 20, Feb 2020

🕒 103.34 ms

Line 46 in File LandSaleWithReferral.sol

```
46 // @CTK NO_OVERFLOW
```

Line 63-88 in File LandSaleWithReferral.sol

```

63     constructor(
64         address landAddress,
65         address sandContractAddress,
66         address initialMetaTx,
67         address admin,
68         address payable initialWalletAddress,
69         bytes32 merkleRoot,
70         uint256 expiryTime,
71         address medianizerContractAddress,
72         address daiTokenContractAddress,
73         address initialSigningWallet,
74         uint256 initialMaxCommissionRate
75     ) public ReferralValidator(
76         initialSigningWallet,
77         initialMaxCommissionRate
78     ) {
79         _land = Land(landAddress);
80         _sand = ERC20(sandContractAddress);
81         _setMetaTransactionProcessor(initialMetaTx, true);
82         _wallet = initialWalletAddress;

```



```

83     _merkleRoot = merkleRoot;
84     _expiryTime = expiryTime;
85     _medianizer = Medianizer(medianizerContractAddress);
86     _dai = ERC20(daiTokenContractAddress);
87     _admin = admin;
88 }


```

✔ The code meets the specification.

## Formal Verification Request 8

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

 20, Feb 2020

 0.86 ms

Line 47 in File LandSaleWithReferral.sol

```
47 // @CTK_NO_BUF_OVERFLOW
```

Line 63-88 in File LandSaleWithReferral.sol

```

63     constructor(
64         address landAddress,
65         address sandContractAddress,
66         address initialMetaTx,
67         address admin,
68         address payable initialWalletAddress,
69         bytes32 merkleRoot,
70         uint256 expiryTime,
71         address medianizerContractAddress,
72         address daiTokenContractAddress,
73         address initialSigningWallet,
74         uint256 initialMaxCommissionRate
75     ) public ReferralValidator(
76         initialSigningWallet,
77         initialMaxCommissionRate
78     ) {
79         _land = Land(landAddress);
80         _sand = ERC20(sandContractAddress);
81         _setMetaTransactionProcessor(initialMetaTx, true);
82         _wallet = initialWalletAddress;
83         _merkleRoot = merkleRoot;
84         _expiryTime = expiryTime;
85         _medianizer = Medianizer(medianizerContractAddress);
86         _dai = ERC20(daiTokenContractAddress);
87         _admin = admin;
88     }


```

✔ The code meets the specification.

## Formal Verification Request 9

Method will not encounter an assertion failure.

 20, Feb 2020

 0.89 ms

Line 48 in File LandSaleWithReferral.sol

```
48 // @CTK NO_ASF
```

Line 63-88 in File LandSaleWithReferral.sol

```
63 constructor(
64     address landAddress,
65     address sandContractAddress,
66     address initialMetaTx,
67     address admin,
68     address payable initialWalletAddress,
69     bytes32 merkleRoot,
70     uint256 expiryTime,
71     address medianizerContractAddress,
72     address daiTokenContractAddress,
73     address initialSigningWallet,
74     uint256 initialMaxCommissionRate
75 ) public ReferralValidator(
76     initialSigningWallet,
77     initialMaxCommissionRate
78 ) {
79     _land = Land(landAddress);
80     _sand = ERC20(sandContractAddress);
81     _setMetaTransactionProcessor(initialMetaTx, true);
82     _wallet = initialWalletAddress;
83     _merkleRoot = merkleRoot;
84     _expiryTime = expiryTime;
85     _medianizer = Medianizer(medianizerContractAddress);
86     _dai = ERC20(daiTokenContractAddress);
87     _admin = admin;
88 }
```

✔ The code meets the specification.

## Formal Verification Request 10

LandSale

📅 20, Feb 2020

🕒 6.44 ms

Line 49-62 in File LandSaleWithReferral.sol

```
49 /* @CTK LandSale
50     @tag assume_completion
51     @post __post._land == landAddress
52     @post __post._sand == sandContractAddress
53     @post __post._metaTransactionContracts[initialMetaTx] == true
54     @post __post._admin == admin
55     @post __post._wallet == initialWalletAddress
56     @post __post._merkleRoot == merkleRoot
57     @post __post._expiryTime == expiryTime
58     @post __post._medianizer == medianizerContractAddress
59     @post __post._dai == daiTokenContractAddress
60     @post __post._signingWallet == initialSigningWallet
61     @post __post._maxCommissionRate == initialMaxCommissionRate
62 */
```

Line 63-88 in File LandSaleWithReferral.sol

```
63     constructor(  
64         address landAddress,  
65         address sandContractAddress,  
66         address initialMetaTx,  
67         address admin,  
68         address payable initialWalletAddress,  
69         bytes32 merkleRoot,  
70         uint256 expiryTime,  
71         address medianizerContractAddress,  
72         address daiTokenContractAddress,  
73         address initialSigningWallet,  
74         uint256 initialMaxCommissionRate  
75     ) public ReferralValidator(  
76         initialSigningWallet,  
77         initialMaxCommissionRate  
78     ) {  
79         _land = Land(landAddress);  
80         _sand = ERC20(sandContractAddress);  
81         _setMetaTransactionProcessor(initialMetaTx, true);  
82         _wallet = initialWalletAddress;  
83         _merkleRoot = merkleRoot;  
84         _expiryTime = expiryTime;  
85         _medianizer = Medianizer(medianizerContractAddress);  
86         _dai = ERC20(daiTokenContractAddress);  
87         _admin = admin;  
88     }
```

✔ The code meets the specification.

## Formal Verification Request 11

setReceivingWallet\_require

📅 20, Feb 2020

🕒 26.15 ms

Line 92-96 in File LandSaleWithReferral.sol

```
92     /*@CTK setReceivingWallet_require  
93     @tag assume_completion  
94     @post newWallet != address(0)  
95     @post msg.sender == _admin  
96     */
```

Line 101-105 in File LandSaleWithReferral.sol


```
101     function setReceivingWallet(address payable newWallet) external {  
102         require(newWallet != address(0), "receiving wallet cannot be zero address");  
103         require(msg.sender == _admin, "only admin can change the receiving wallet");  
104         _wallet = newWallet;  
105     }
```

✔ The code meets the specification.

## Formal Verification Request 12

setReceivingWallet\_change

 20, Feb 2020

 3.56 ms

Line 97-100 in File LandSaleWithReferral.sol

```
97  /*@CTK setReceivingWallet_change
98  @tag assume_completion
99  @post __post._wallet == newWallet
100  */
```

Line 101-105 in File LandSaleWithReferral.sol


```
101  function setReceivingWallet(address payable newWallet) external{
102  require(newWallet != address(0), "receiving wallet cannot be zero address");
103  require(msg.sender == _admin, "only admin can change the receiving wallet");
104  _wallet = newWallet;
105  }
```

 The code meets the specification.

## Formal Verification Request 13

setDAIEnabled\_require

 20, Feb 2020

 15.81 ms

Line 109-112 in File LandSaleWithReferral.sol

```
109  /*@CTK setDAIEnabled_require
110  @tag assume_completion
111  @post msg.sender == _admin
112  */
```

Line 117-120 in File LandSaleWithReferral.sol


```
117  function setDAIEnabled(bool enabled) external {
118  require(msg.sender == _admin, "only admin can enable/disable DAI");
119  _daiEnabled = enabled;
120  }
```

 The code meets the specification.

## Formal Verification Request 14

setDAIEnabled\_change

 20, Feb 2020

 2.95 ms

Line 113-116 in File LandSaleWithReferral.sol

```
113  /*@CTK setDAIEnabled_change
114     @tag assume_completion
115     @post __post._daiEnabled == enabled
116     */
```

Line 117-120 in File LandSaleWithReferral.sol

```
117  function setDAIEnabled(bool enabled) external {
118     require(msg.sender == _admin, "only admin can enable/disable DAI");
119     _daiEnabled = enabled;
120  }
```

✔ The code meets the specification.

## Formal Verification Request 15

isDAIEnabled

📅 20, Feb 2020

🕒 3.61 ms

Line 124-126 in File LandSaleWithReferral.sol

```
124  /*@CTK isDAIEnabled
125     @post __return == _daiEnabled
126     */
```

Line 127-129 in File LandSaleWithReferral.sol

```
127  function isDAIEnabled() external view returns (bool) {
128     return _daiEnabled;
129  }
```

✔ The code meets the specification.

## Formal Verification Request 16

setETHEnabled\_require

📅 20, Feb 2020

🕒 15.92 ms

Line 133-136 in File LandSaleWithReferral.sol

```
133  /*@CTK setETHEnabled_require
134     @tag assume_completion
135     @post msg.sender == _admin
136     */
```

Line 141-144 in File LandSaleWithReferral.sol


```
141  function setETHEnabled(bool enabled) external {
142     require(msg.sender == _admin, "only admin can enable/disable ETH");
143     _etherEnabled = enabled;
144  }
```

✔ The code meets the specification.

## Formal Verification Request 17

setETHEnabled\_change

 20, Feb 2020

 2.84 ms

Line 137-140 in File LandSaleWithReferral.sol

```
137  /*@CTK setETHEnabled_change
138     @tag assume_completion
139     @post __post._etherEnabled == enabled
140  */
```

Line 141-144 in File LandSaleWithReferral.sol


```
141  function setETHEnabled(bool enabled) external {
142     require(msg.sender == _admin, "only admin can enable/disable ETH");
143     _etherEnabled = enabled;
144  }
```

 The code meets the specification.

## Formal Verification Request 18

isETHEnabled

 20, Feb 2020

 3.54 ms

Line 148-150 in File LandSaleWithReferral.sol

```
148  /*@CTK isETHEnabled
149     @post __return == _etherEnabled
150  */
```

Line 151-153 in File LandSaleWithReferral.sol


```
151  function isETHEnabled() external view returns (bool) {
152     return _etherEnabled;
153  }
```

 The code meets the specification.

## Formal Verification Request 19

setSANDEnabled\_require

 20, Feb 2020

 15.4 ms

Line 157-160 in File LandSaleWithReferral.sol

```
157  /*@CTK setSANDEnabled_require
158     @tag assume_completion
159     @post msg.sender == _admin
160  */
```

Line 165-168 in File LandSaleWithReferral.sol

```
165     function setSANDEnabled(bool enabled) external {
166         require(msg.sender == _admin, "only admin can enable/disable SAND");
167         _sandEnabled = enabled;
168     }
```

✓ The code meets the specification.

## Formal Verification Request 20

setSANDEnabled\_change

📅 20, Feb 2020

🕒 2.84 ms

Line 161-164 in File LandSaleWithReferral.sol

```
161     /*@CTK setSANDEnabled_change
162     @tag assume_completion
163     @post __post._sandEnabled == enabled
164     */
```

Line 165-168 in File LandSaleWithReferral.sol

```
165     function setSANDEnabled(bool enabled) external {
166         require(msg.sender == _admin, "only admin can enable/disable SAND");
167         _sandEnabled = enabled;
168     }
```

✓ The code meets the specification.

## Formal Verification Request 21

isSANDEnabled

📅 20, Feb 2020

🕒 3.81 ms

Line 172-174 in File LandSaleWithReferral.sol

```
172     /*@CTK isSANDEnabled
173     @post __return == _sandEnabled
174     */
```

Line 175-177 in File LandSaleWithReferral.sol


```
175     function isSANDEnabled() external view returns (bool) {
176         return _sandEnabled;
177     }
```

✓ The code meets the specification.

## Formal Verification Request 22

`_checkValidity`

 20, Feb 2020

 200.28 ms

Line 179-184 in File LandSaleWithReferral.sol

```
179  /*@CTK _checkValidity
180     @tag assume_completion
181     @post now < _expiryTime
182     @post buyer == msg.sender \\/ _metaTransactionContracts[msg.sender] == true
183     @post reserved == address(0) \\/ reserved == buyer
184     */
```

Line 185-205 in File LandSaleWithReferral.sol


```
185  function _checkValidity(
186     address buyer,
187     address reserved,
188     uint256 x,
189     uint256 y,
190     uint256 size,
191     uint256 price,
192     bytes32 salt,
193     bytes32[] memory proof
194 ) internal view {
195     /* solium-disable-next-line security/no-block-members */
196     require(block.timestamp < _expiryTime, "sale is over");
197     require(buyer == msg.sender || _metaTransactionContracts[msg.sender], "not
198         authorized");
199     require(reserved == address(0) || reserved == buyer, "cannot buy reserved Land");
200     bytes32 leaf = _generateLandHash(x, y, size, price, reserved, salt);
201     require(
202         _verify(proof, leaf),
203         "Invalid land provided"
204     );
205 }
```

 The code meets the specification.

## Formal Verification Request 23

If method completes, integer overflow would not happen.

 20, Feb 2020

 367.49 ms

Line 226 in File LandSaleWithReferral.sol

```
226  // @CTK NO_OVERFLOW
```

Line 235-259 in File LandSaleWithReferral.sol

```
235  function buyLandWithSand(
236     address buyer,
237     address to,
```



```
238     address reserved,
239     uint256 x,
240     uint256 y,
241     uint256 size,
242     uint256 priceInSand,
243     bytes32 salt,
244     bytes32[] calldata proof,
245     bytes calldata referral
246 ) external {
247     require(_sandEnabled, "sand payments not enabled");
248     _checkValidity(buyer, reserved, x, y, size, priceInSand, salt, proof);
249
250     handleReferralWithERC20(
251         buyer,
252         priceInSand,
253         referral,
254         _wallet,
255         address(_sand)
256     );
257
258     _mint(buyer, to, x, y, size, priceInSand, address(_sand), priceInSand);
259 }
```

✔ The code meets the specification.

## Formal Verification Request 24

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

📅 20, Feb 2020

🕒 89.16 ms

Line 227 in File LandSaleWithReferral.sol

```
227 // @CTK NO_BUF_OVERFLOW
```

Line 235-259 in File LandSaleWithReferral.sol

```
235     function buyLandWithSand(
236         address buyer,
237         address to,
238         address reserved,
239         uint256 x,
240         uint256 y,
241         uint256 size,
242         uint256 priceInSand,
243         bytes32 salt,
244         bytes32[] calldata proof,
245         bytes calldata referral
246 ) external {
247     require(_sandEnabled, "sand payments not enabled");
248     _checkValidity(buyer, reserved, x, y, size, priceInSand, salt, proof);
249
250     handleReferralWithERC20(
251         buyer,
252         priceInSand,
253         referral,
254         _wallet,
```

```

255     address(_sand)
256   );
257
258   _mint(buyer, to, x, y, size, priceInSand, address(_sand), priceInSand);
259 }


```

✔ The code meets the specification.

## Formal Verification Request 25

Method will not encounter an assertion failure.

 20, Feb 2020

 80.75 ms

Line 228 in File LandSaleWithReferral.sol

```

228  // @CTK_NO_ASF

```

Line 235-259 in File LandSaleWithReferral.sol

```

235  function buyLandWithSand(
236    address buyer,
237    address to,
238    address reserved,
239    uint256 x,
240    uint256 y,
241    uint256 size,
242    uint256 priceInSand,
243    bytes32 salt,
244    bytes32[] calldata proof,
245    bytes calldata referral
246  ) external {
247    require(_sandEnabled, "sand payments not enabled");
248    _checkValidity(buyer, reserved, x, y, size, priceInSand, salt, proof);
249
250    handleReferralWithERC20(
251      buyer,
252      priceInSand,
253      referral,
254      _wallet,
255      address(_sand)
256    );
257
258    _mint(buyer, to, x, y, size, priceInSand, address(_sand), priceInSand);
259 }


```

✔ The code meets the specification.

## Formal Verification Request 26

buyLandWithSand

 20, Feb 2020

 47.81 ms

Line 229-234 in File LandSaleWithReferral.sol

```

229  /*@CTK buyLandWithSand
230     @tag assume_completion
231     @post _sandEnabled == true
232     @post buyer == msg.sender \/_metaTransactionContracts[msg.sender] == true
233     @post reserved == address(0) \/_reserved == buyer
234     */

```

Line 235-259 in File LandSaleWithReferral.sol

```

235  function buyLandWithSand(
236      address buyer,
237      address to,
238      address reserved,
239      uint256 x,
240      uint256 y,
241      uint256 size,
242      uint256 priceInSand,
243      bytes32 salt,
244      bytes32[] calldata proof,
245      bytes calldata referral
246  ) external {
247      require(_sandEnabled, "sand payments not enabled");
248      _checkValidity(buyer, reserved, x, y, size, priceInSand, salt, proof);
249
250      handleReferralWithERC20(
251          buyer,
252          priceInSand,
253          referral,
254          _wallet,
255          address(_sand)
256      );
257
258      _mint(buyer, to, x, y, size, priceInSand, address(_sand), priceInSand);
259  }


```

✔ The code meets the specification.

## Formal Verification Request 27

If method completes, integer overflow would not happen.

 20, Feb 2020

 496.09 ms

Line 274 in File LandSaleWithReferral.sol

```

274  // @CTK NO_OVERFLOW

```

Line 282-311 in File LandSaleWithReferral.sol

```

282  function buyLandWithETH(
283      address buyer,
284      address to,
285      address reserved,
286      uint256 x,
287      uint256 y,
288      uint256 size,
289      uint256 priceInSand,

```

```
290     bytes32 salt,
291     bytes32[] calldata proof,
292     bytes calldata referral
293 ) external payable {
294     require(!_etherEnabled, "ether payments not enabled");
295     _checkValidity(buyer, reserved, x, y, size, priceInSand, salt, proof);
296
297     uint256 ETHRequired = getEtherAmountWithSAND(priceInSand);
298     require(msg.value >= ETHRequired, "not enough ether sent");
299
300     if (msg.value - ETHRequired > 0) {
301         msg.sender.transfer(msg.value - ETHRequired); // refund extra
302     }
303
304     handleReferralWithETH(
305         ETHRequired,
306         referral,
307         _wallet
308     );
309
310     _mint(buyer, to, x, y, size, priceInSand, address(0), ETHRequired);
311 }
```

✔ The code meets the specification.

## Formal Verification Request 28

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

📅 20, Feb 2020

🕒 138.76 ms

Line 275 in File LandSaleWithReferral.sol

```
275 // @CTK_NO_BUF_OVERFLOW
```

Line 282-311 in File LandSaleWithReferral.sol

```
282     function buyLandWithETH(
283         address buyer,
284         address to,
285         address reserved,
286         uint256 x,
287         uint256 y,
288         uint256 size,
289         uint256 priceInSand,
290         bytes32 salt,
291         bytes32[] calldata proof,
292         bytes calldata referral
293 ) external payable {
294     require(!_etherEnabled, "ether payments not enabled");
295     _checkValidity(buyer, reserved, x, y, size, priceInSand, salt, proof);
296
297     uint256 ETHRequired = getEtherAmountWithSAND(priceInSand);
298     require(msg.value >= ETHRequired, "not enough ether sent");
299
300     if (msg.value - ETHRequired > 0) {
301         msg.sender.transfer(msg.value - ETHRequired); // refund extra
```

```

302     }
303
304     handleReferralWithETH(
305         ETHRequired,
306         referral,
307         _wallet
308     );
309
310     _mint(buyer, to, x, y, size, priceInSand, address(0), ETHRequired);
311 }


```

✔ The code meets the specification.

## Formal Verification Request 29

buyLandWithETH\_require

 20, Feb 2020

 217.2 ms

Line 276-281 in File LandSaleWithReferral.sol

```

276     /*@CTK buyLandWithETH_require
277         @tag assume_completion
278         @post _sandEnabled == true
279         @post buyer == msg.sender \\/ _metaTransactionContracts[msg.sender] == true
280         @post reserved == address(0) \\/ reserved == buyer
281     */

```

Line 282-311 in File LandSaleWithReferral.sol

```

282     function buyLandWithETH(
283         address buyer,
284         address to,
285         address reserved,
286         uint256 x,
287         uint256 y,
288         uint256 size,
289         uint256 priceInSand,
290         bytes32 salt,
291         bytes32[] calldata proof,
292         bytes calldata referral
293     ) external payable {
294         require(_etherEnabled, "ether payments not enabled");
295         _checkValidity(buyer, reserved, x, y, size, priceInSand, salt, proof);
296
297         uint256 ETHRequired = getEtherAmountWithSAND(priceInSand);
298         require(msg.value >= ETHRequired, "not enough ether sent");
299
300         if (msg.value - ETHRequired > 0) {
301             msg.sender.transfer(msg.value - ETHRequired); // refund extra
302         }
303
304         handleReferralWithETH(
305             ETHRequired,
306             referral,
307             _wallet
308         );

```

```
309
310     _mint(buyer, to, x, y, size, priceInSand, address(0), ETHRequired);
311 }
```

✔ The code meets the specification.

## Formal Verification Request 30

If method completes, integer overflow would not happen.

📅 20, Feb 2020

🕒 344.69 ms

Line 325 in File LandSaleWithReferral.sol

```
325 // @CTK_NO_OVERFLOW
```

Line 333-359 in File LandSaleWithReferral.sol

```
333 function buyLandWithDAI(
334     address buyer,
335     address to,
336     address reserved,
337     uint256 x,
338     uint256 y,
339     uint256 size,
340     uint256 priceInSand,
341     bytes32 salt,
342     bytes32[] calldata proof,
343     bytes calldata referral
344 ) external {
345     require(_daiEnabled, "dai payments not enabled");
346     _checkValidity(buyer, reserved, x, y, size, priceInSand, salt, proof);
347
348     uint256 DAIRequired = priceInSand.mul(daiPrice).div(1000000000000000000);
349
350     handleReferralWithERC20(
351         buyer,
352         DAIRequired,
353         referral,
354         _wallet,
355         address(_dai)
356     );
357
358     _mint(buyer, to, x, y, size, priceInSand, address(_dai), DAIRequired);
359 }
```

✔ The code meets the specification.

## Formal Verification Request 31

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

📅 20, Feb 2020

🕒 140.75 ms

Line 326 in File LandSaleWithReferral.sol

326 `//@CTK NO_BUF_OVERFLOW`

Line 333-359 in File LandSaleWithReferral.sol

```

333     function buyLandWithDAI(
334         address buyer,
335         address to,
336         address reserved,
337         uint256 x,
338         uint256 y,
339         uint256 size,
340         uint256 priceInSand,
341         bytes32 salt,
342         bytes32[] calldata proof,
343         bytes calldata referral
344     ) external {
345         require(_daiEnabled, "dai payments not enabled");
346         _checkValidity(buyer, reserved, x, y, size, priceInSand, salt, proof);
347
348         uint256 DAIRequired = priceInSand.mul(daiPrice).div(1000000000000000000);
349
350         handleReferralWithERC20(
351             buyer,
352             DAIRequired,
353             referral,
354             _wallet,
355             address(_dai)
356         );
357
358         _mint(buyer, to, x, y, size, priceInSand, address(_dai), DAIRequired);
359     }


```

✓ The code meets the specification.

## Formal Verification Request 32

buyLandWithDAI

 20, Feb 2020

 31.8 ms

Line 327-332 in File LandSaleWithReferral.sol

```

327     /*@CTK buyLandWithDAI
328         @tag assume_completion
329         @post _sandEnabled == true
330         @post buyer == msg.sender \\/ _metaTransactionContracts[msg.sender] == true
331         @post reserved == address(0) \\/ reserved == buyer
332     */

```

Line 333-359 in File LandSaleWithReferral.sol

```

333     function buyLandWithDAI(
334         address buyer,
335         address to,
336         address reserved,
337         uint256 x,
338         uint256 y,

```

```

339     uint256 size,
340     uint256 priceInSand,
341     bytes32 salt,
342     bytes32[] calldata proof,
343     bytes calldata referral
344 ) external {
345     require(_daiEnabled, "dai payments not enabled");
346     _checkValidity(buyer, reserved, x, y, size, priceInSand, salt, proof);
347
348     uint256 DAIRequired = priceInSand.mul(daiPrice).div(1000000000000000000);
349
350     handleReferralWithERC20(
351         buyer,
352         DAIRequired,
353         referral,
354         _wallet,
355         address(_dai)
356     );
357
358     _mint(buyer, to, x, y, size, priceInSand, address(_dai), DAIRequired);
359 }


```

✔ The code meets the specification.

## Formal Verification Request 33

getExpiryTime

 20, Feb 2020

 5.12 ms

Line 365-367 in File LandSaleWithReferral.sol

```

365     /*@CTK getExpiryTime
366     @post __return == _expiryTime
367     */

```

Line 368-370 in File LandSaleWithReferral.sol

```

368     function getExpiryTime() external view returns(uint256) {
369         return _expiryTime;
370     }


```

✔ The code meets the specification.

## Formal Verification Request 34

merkleRoot

 20, Feb 2020

 5.23 ms

Line 376-378 in File LandSaleWithReferral.sol

```

376     /*@CTK merkleRoot
377     @post __return == _merkleRoot
378     */

```



Line 379-381 in File LandSaleWithReferral.sol


```
379     function merkleRoot() external view returns(bytes32) {
380         return _merkleRoot;
381     }
```

✔ The code meets the specification.

## Formal Verification Request 35

If method completes, integer overflow would not happen.

 20, Feb 2020

 0.4 ms

Line 383 in File LandSaleWithReferral.sol

```
383     //@CTK_NO_OVERFLOW
```

Line 386-406 in File LandSaleWithReferral.sol


```
386     function _generateLandHash(
387         uint256 x,
388         uint256 y,
389         uint256 size,
390         uint256 price,
391         address reserved,
392         bytes32 salt
393     ) internal pure returns (
394         bytes32
395     ) {
396         return keccak256(
397             abi.encodePacked(
398                 x,
399                 y,
400                 size,
401                 price,
402                 reserved,
403                 salt
404             )
405         );
406     }
```

✔ The code meets the specification.

## Formal Verification Request 36

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

 20, Feb 2020

 0.38 ms

Line 384 in File LandSaleWithReferral.sol

```
384     //@CTK_NO_BUF_OVERFLOW
```

Line 386-406 in File LandSaleWithReferral.sol

```

386     function _generateLandHash(
387         uint256 x,
388         uint256 y,
389         uint256 size,
390         uint256 price,
391         address reserved,
392         bytes32 salt
393     ) internal pure returns (
394         bytes32
395     ) {
396         return keccak256(
397             abi.encodePacked(
398                 x,
399                 y,
400                 size,
401                 price,
402                 reserved,
403                 salt
404             )
405         );
406     }

```

✔ The code meets the specification.

## Formal Verification Request 37

Method will not encounter an assertion failure.

📅 20, Feb 2020

🕒 0.38 ms

Line 385 in File LandSaleWithReferral.sol

```
385 // @CTK_NO_ASF
```

Line 386-406 in File LandSaleWithReferral.sol

```

386     function _generateLandHash(
387         uint256 x,
388         uint256 y,
389         uint256 size,
390         uint256 price,
391         address reserved,
392         bytes32 salt
393     ) internal pure returns (
394         bytes32
395     ) {
396         return keccak256(
397             abi.encodePacked(
398                 x,
399                 y,
400                 size,
401                 price,
402                 reserved,
403                 salt
404             )
405         );
406     }

```

✔ The code meets the specification.

## Formal Verification Request 38

If method completes, integer overflow would not happen.

📅 20, Feb 2020

🕒 0.58 ms

Line 408 in File LandSaleWithReferral.sol

408 `//@CTK NO_OVERFLOW`

Line 411-428 in File LandSaleWithReferral.sol

```

411 function _verify(bytes32[] memory proof, bytes32 leaf) internal view returns (bool) {
412     bytes32 computedHash = leaf;
413
414     /*@CTK ForLoop_verify
415      @inv true
416      */
417     for (uint256 i = 0; i < proof.length; i++) {
418         bytes32 proofElement = proof[i];
419
420         if (computedHash < proofElement) {
421             computedHash = keccak256(abi.encodePacked(computedHash, proofElement));
422         } else {
423             computedHash = keccak256(abi.encodePacked(proofElement, computedHash));
424         }
425     }
426
427     return computedHash == _merkleRoot;
428 }

```

✔ The code meets the specification.

## Formal Verification Request 39

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

📅 20, Feb 2020

🕒 0.71 ms

Line 409 in File LandSaleWithReferral.sol

409 `//@CTK NO_BUF_OVERFLOW`

Line 411-428 in File LandSaleWithReferral.sol

```

411 function _verify(bytes32[] memory proof, bytes32 leaf) internal view returns (bool) {
412     bytes32 computedHash = leaf;
413
414     /*@CTK ForLoop_verify
415      @inv true
416      */
417     for (uint256 i = 0; i < proof.length; i++) {
418         bytes32 proofElement = proof[i];

```

```

419
420     if (computedHash < proofElement) {
421         computedHash = keccak256(abi.encodePacked(computedHash, proofElement));
422     } else {
423         computedHash = keccak256(abi.encodePacked(proofElement, computedHash));
424     }
425 }
426
427 return computedHash == _merkleRoot;
428 }


```

 The code meets the specification.

## Formal Verification Request 40

Method will not encounter an assertion failure.

 20, Feb 2020

 0.53 ms

Line 410 in File LandSaleWithReferral.sol

```
410 // @CTK NO_ASF
```

Line 411-428 in File LandSaleWithReferral.sol

```

411 function _verify(bytes32[] memory proof, bytes32 leaf) internal view returns (bool) {
412     bytes32 computedHash = leaf;
413
414     /* @CTK ForLoop_verify
415        @inv true
416        */
417     for (uint256 i = 0; i < proof.length; i++) {
418         bytes32 proofElement = proof[i];
419
420         if (computedHash < proofElement) {
421             computedHash = keccak256(abi.encodePacked(computedHash, proofElement));
422         } else {
423             computedHash = keccak256(abi.encodePacked(proofElement, computedHash));
424         }
425     }
426
427     return computedHash == _merkleRoot;
428 }


```

 The code meets the specification.

## Formal Verification Request 41

If method completes, integer overflow would not happen.

 20, Feb 2020

 0.95 ms

Line 435 in File LandSaleWithReferral.sol

435 `//@CTK NO_OVERFLOW`

Line 437-440 in File LandSaleWithReferral.sol


```
437     function getEtherAmountWithSAND(uint256 sandAmount) public view returns (uint256) {
438         uint256 ethUsdPair = getEthUsdPair();
439         return sandAmount.mul(daiPrice).div(ethUsdPair);
440     }
```

 The code meets the specification.

## Formal Verification Request 42

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

 20, Feb 2020

 1.08 ms

Line 436 in File LandSaleWithReferral.sol

436 `//@CTK NO_BUF_OVERFLOW`

Line 437-440 in File LandSaleWithReferral.sol


```
437     function getEtherAmountWithSAND(uint256 sandAmount) public view returns (uint256) {
438         uint256 ethUsdPair = getEthUsdPair();
439         return sandAmount.mul(daiPrice).div(ethUsdPair);
440     }
```

 The code meets the specification.

## Formal Verification Request 43

ForLoop\_verify\_\_\_Generated

 20, Feb 2020

 30.77 ms

(Loop) Line 414-416 in File LandSaleWithReferral.sol

```
414     /*@CTK ForLoop_verify
415         @inv true
416     */
```

(Loop) Line 414-425 in File LandSaleWithReferral.sol

```
414     /*@CTK ForLoop_verify
415         @inv true
416     */
417     for (uint256 i = 0; i < proof.length; i++) {
418         bytes32 proofElement = proof[i];
419
420         if (computedHash < proofElement) {
421             computedHash = keccak256(abi.encodePacked(computedHash, proofElement));
422         } else {
423             computedHash = keccak256(abi.encodePacked(proofElement, computedHash));
424         }
425     }
```

✔ The code meets the specification.

## Formal Verification Request 44

### ReferralValidator

📅 20, Feb 2020

🕒 13.53 ms

Line 31-34 in File ReferralValidator.sol

```
31  /*@CTK ReferralValidator
32     @post __post._signingWallet == initialSigningWallet
33     @post __post._maxCommissionRate == initialMaxCommissionRate
34  */
```

Line 35-41 in File ReferralValidator.sol

```
35  constructor(
36     address initialSigningWallet,
37     uint256 initialMaxCommissionRate
38  ) public {
39     _signingWallet = initialSigningWallet;
40     _maxCommissionRate = initialMaxCommissionRate;
41  }
```

✔ The code meets the specification.

## Formal Verification Request 45

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

📅 20, Feb 2020

🕒 47.84 ms

Line 47 in File ReferralValidator.sol

```
47  //@CTK NO_BUF_OVERFLOW
```

Line 59-63 in File ReferralValidator.sol

```
59  function updateSigningWallet(address newSigningWallet) external {
60     require(_admin == msg.sender, "Sender not admin");
61     _previousSigningWallets[_signingWallet] = now + _previousSigningDelay;
62     _signingWallet = newSigningWallet;
63  }
```

✔ The code meets the specification.

## Formal Verification Request 46

Method will not encounter an assertion failure.

📅 20, Feb 2020

🕒 0.66 ms

Line 48 in File ReferralValidator.sol

```
48 // @CTK NO_ASF
```

Line 59-63 in File ReferralValidator.sol


```
59 function updateSigningWallet(address newSigningWallet) external {
60     require(_admin == msg.sender, "Sender not admin");
61     _previousSigningWallets[_signingWallet] = now + _previousSigningDelay;
62     _signingWallet = newSigningWallet;
63 }
```

 The code meets the specification.

## Formal Verification Request 47

updateSigningWallet\_require

 20, Feb 2020

 1.55 ms

Line 49-52 in File ReferralValidator.sol

```
49 /* @CTK updateSigningWallet_require
50     @tag assume_completion
51     @post _admin == msg.sender
52 */
```

Line 59-63 in File ReferralValidator.sol


```
59 function updateSigningWallet(address newSigningWallet) external {
60     require(_admin == msg.sender, "Sender not admin");
61     _previousSigningWallets[_signingWallet] = now + _previousSigningDelay;
62     _signingWallet = newSigningWallet;
63 }
```

 The code meets the specification.

## Formal Verification Request 48

updateSigningWallet\_change

 20, Feb 2020

 2.79 ms

Line 53-58 in File ReferralValidator.sol

```
53 /* @CTK updateSigningWallet_change
54     @tag assume_completion
55     @pre _admin == msg.sender
56     @post __post._previousSigningWallets[_signingWallet] == now + _previousSigningDelay
57     @post __post._signingWallet == newSigningWallet
58 */
```

Line 59-63 in File ReferralValidator.sol

```

59     function updateSigningWallet(address newSigningWallet) external {
60         require(_admin == msg.sender, "Sender not admin");
61         _previousSigningWallets[_signingWallet] = now + _previousSigningDelay;
62         _signingWallet = newSigningWallet;
63     }


```

✔ The code meets the specification.

## Formal Verification Request 49

updateMaxCommissionRate\_require

 20, Feb 2020

 20.26 ms

Line 70-73 in File ReferralValidator.sol

```

70     /*@CTK updateMaxCommissionRate_require
71         @tag assume_completion
72         @post _admin == msg.sender
73     */

```

Line 79-82 in File ReferralValidator.sol

```

79     function updateMaxCommissionRate(uint256 newMaxCommissionRate) external {
80         require(_admin == msg.sender, "Sender not admin");
81         _maxCommissionRate = newMaxCommissionRate;
82     }


```

✔ The code meets the specification.

## Formal Verification Request 50

updateMaxCommissionRate\_change

 20, Feb 2020

 1.59 ms

Line 74-78 in File ReferralValidator.sol

```

74     /*@CTK updateMaxCommissionRate_change
75         @tag assume_completion
76         @pre _admin == msg.sender
77         @post __post._maxCommissionRate == newMaxCommissionRate
78     */

```

Line 79-82 in File ReferralValidator.sol

```

79     function updateMaxCommissionRate(uint256 newMaxCommissionRate) external {
80         require(_admin == msg.sender, "Sender not admin");
81         _maxCommissionRate = newMaxCommissionRate;
82     }

```


✔ The code meets the specification.



## Formal Verification Request 51

If method completes, integer overflow would not happen.

 20, Feb 2020

 288.26 ms

Line 84 in File ReferralValidator.sol

84 `//@CTK NO_OVERFLOW`

Line 86-130 in File ReferralValidator.sol

```

86     function handleReferralWithETH(
87         uint256 amount,
88         bytes memory referral,
89         address payable destination
90     ) internal {
91         uint256 amountForDestination = amount;
92
93         if (referral.length > 0) {
94             (
95                 bytes memory signature,
96                 address referrer,
97                 address referee,
98                 uint256 expiryTime,
99                 uint256 commissionRate
100            ) = decodeReferral(referral);
101
102            uint256 commission = 0;
103
104            if (isReferralValid(signature, referrer, referee, expiryTime, commissionRate)) {
105                commission = SafeMathWithRequire.div(
106                    SafeMathWithRequire.mul(amount, commissionRate),
107                    10000
108                );
109
110                emit ReferralUsed(
111                    referrer,
112                    referee,
113                    address(0),
114                    amount,
115                    commission,
116                    commissionRate
117                );
118                amountForDestination = SafeMathWithRequire.sub(
119                    amountForDestination,
120                    commission
121                );
122            }
123
124            if (commission > 0) {
125                address(uint160(referrer)).transfer(commission);
126            }
127        }
128
129        destination.transfer(amountForDestination);
130    }


```

 The code meets the specification.

## Formal Verification Request 52

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

 20, Feb 2020

 44.33 ms

Line 85 in File ReferralValidator.sol

85 `//@CTK_NO_BUF_OVERFLOW`

Line 86-130 in File ReferralValidator.sol

```

86     function handleReferralWithETH(
87         uint256 amount,
88         bytes memory referral,
89         address payable destination
90     ) internal {
91         uint256 amountForDestination = amount;
92
93         if (referral.length > 0) {
94             (
95                 bytes memory signature,
96                 address referrer,
97                 address referee,
98                 uint256 expiryTime,
99                 uint256 commissionRate
100            ) = decodeReferral(referral);
101
102            uint256 commission = 0;
103
104            if (isReferralValid(signature, referrer, referee, expiryTime, commissionRate)) {
105                commission = SafeMathWithRequire.div(
106                    SafeMathWithRequire.mul(amount, commissionRate),
107                    10000
108                );
109
110                emit ReferralUsed(
111                    referrer,
112                    referee,
113                    address(0),
114                    amount,
115                    commission,
116                    commissionRate
117                );
118                amountForDestination = SafeMathWithRequire.sub(
119                    amountForDestination,
120                    commission
121                );
122            }
123
124            if (commission > 0) {
125                address(uint160(referrer)).transfer(commission);
126            }
127        }
128
129        destination.transfer(amountForDestination);
130    }


```

 The code meets the specification.

## Formal Verification Request 53

If method completes, integer overflow would not happen.

 20, Feb 2020

 146.71 ms

Line 132 in File ReferralValidator.sol

132 `//@CTK NO_OVERFLOW`

Line 135-186 in File ReferralValidator.sol

```

135     function handleReferralWithERC20(
136         address buyer,
137         uint256 amount,
138         bytes memory referral,
139         address payable destination,
140         address tokenAddress
141     ) internal {
142         ERC20 token = ERC20(tokenAddress);
143         uint256 amountForDestination = amount;
144
145         if (referral.length > 0) {
146             (
147                 bytes memory signature,
148                 address referrer,
149                 address referee,
150                 uint256 expiryTime,
151                 uint256 commissionRate
152             ) = decodeReferral(referral);
153
154             uint256 commission = 0;
155
156             if (isReferralValid(signature, referrer, referee, expiryTime, commissionRate)) {
157                 commission = SafeMathWithRequire.div(
158                     SafeMathWithRequire.mul(amount, commissionRate),
159                     10000
160                 );
161
162                 emit ReferralUsed(
163                     referrer,
164                     referee,
165                     tokenAddress,
166                     amount,
167                     commission,
168                     commissionRate
169                 );
170                 amountForDestination = SafeMathWithRequire.sub(
171                     amountForDestination,
172                     commission
173                 );
174             }
175
176             if (commission > 0) {
177                 token.transferFrom(buyer, referrer, commission);
178             }
179         }
180     }

```

```
181     token.transferFrom(buyer, destination, amountForDestination);
182 }
```

✔ The code meets the specification.

## Formal Verification Request 54

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

📅 20, Feb 2020

🕒 2.03 ms

Line 133 in File ReferralValidator.sol

```
133 // @CTK_NO_BUF_OVERFLOW
```

Line 135-186 in File ReferralValidator.sol

```
135     function handleReferralWithERC20(
136         address buyer,
137         uint256 amount,
138         bytes memory referral,
139         address payable destination,
140         address tokenAddress
141     ) internal {
142         ERC20 token = ERC20(tokenAddress);
143         uint256 amountForDestination = amount;
144
145         if (referral.length > 0) {
146             (
147                 bytes memory signature,
148                 address referrer,
149                 address referee,
150                 uint256 expiryTime,
151                 uint256 commissionRate
152             ) = decodeReferral(referral);
153
154             uint256 commission = 0;
155
156             if (isReferralValid(signature, referrer, referee, expiryTime, commissionRate)) {
157                 commission = SafeMathWithRequire.div(
158                     SafeMathWithRequire.mul(amount, commissionRate),
159                     10000
160                 );
161
162                 emit ReferralUsed(
163                     referrer,
164                     referee,
165                     tokenAddress,
166                     amount,
167                     commission,
168                     commissionRate
169                 );
170                 amountForDestination = SafeMathWithRequire.sub(
171                     amountForDestination,
172                     commission
173                 );
174             }
175         }
176     }
```

```

175
176     if (commission > 0) {
177         token.transferFrom(buyer, referrer, commission);
178     }
179 }
180
181     token.transferFrom(buyer, destination, amountForDestination);
182 }

```

✔ The code meets the specification.

## Formal Verification Request 55

Method will not encounter an assertion failure.

📅 20, Feb 2020

🕒 2.42 ms

Line 134 in File ReferralValidator.sol

```
134 // @CTK NO_ASF
```

Line 135-186 in File ReferralValidator.sol

```

135     function handleReferralWithERC20(
136         address buyer,
137         uint256 amount,
138         bytes memory referral,
139         address payable destination,
140         address tokenAddress
141     ) internal {
142         ERC20 token = ERC20(tokenAddress);
143         uint256 amountForDestination = amount;
144
145         if (referral.length > 0) {
146             (
147                 bytes memory signature,
148                 address referrer,
149                 address referee,
150                 uint256 expiryTime,
151                 uint256 commissionRate
152             ) = decodeReferral(referral);
153
154             uint256 commission = 0;
155
156             if (isReferralValid(signature, referrer, referee, expiryTime, commissionRate)) {
157                 commission = SafeMathWithRequire.div(
158                     SafeMathWithRequire.mul(amount, commissionRate),
159                     10000
160                 );
161
162                 emit ReferralUsed(
163                     referrer,
164                     referee,
165                     tokenAddress,
166                     amount,
167                     commission,
168                     commissionRate

```

```

169         );
170         amountForDestination = SafeMathWithRequire.sub(
171             amountForDestination,
172             commission
173         );
174     }
175
176     if (commission > 0) {
177         token.transferFrom(buyer, referrer, commission);
178     }
179 }
180
181 token.transferFrom(buyer, destination, amountForDestination);
182 }

```

✔ The code meets the specification.

## Formal Verification Request 56

If method completes, integer overflow would not happen.

📅 20, Feb 2020

🕒 0.89 ms

Line 197 in File ReferralValidator.sol

```
197 // @CTK_NO_OVERFLOW
```

Line 200-236 in File ReferralValidator.sol

```

200 function isReferralValid(
201     bytes memory signature,
202     address referrer,
203     address referee,
204     uint256 expiryTime,
205     uint256 commissionRate
206 ) public view returns (
207     bool
208 ) {
209     if (commissionRate > _maxCommissionRate || referrer == referee || now > expiryTime) {
210         return false;
211     }
212
213     bytes32 hashedData = keccak256(
214         abi.encodePacked(
215             referrer,
216             referee,
217             expiryTime,
218             commissionRate
219         )
220     );
221
222     address signer = SigUtil.recover(
223         keccak256(
224             abi.encodePacked("\x19Ethereum Signed Message:\n32", hashedData)
225         ),
226         signature

```

```

227     );
228
229     if (_previousSigningWallets[signer] >= now) {
230         return true;
231     }
232
233     return _signingWallet == signer;
234 }

```

✔ The code meets the specification.

## Formal Verification Request 57

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

📅 20, Feb 2020

🕒 0.53 ms

Line 198 in File ReferralValidator.sol

```
198 // @CTK_NO_BUF_OVERFLOW
```

Line 200-236 in File ReferralValidator.sol

```

200     function isReferralValid(
201         bytes memory signature,
202         address referrer,
203         address referee,
204         uint256 expiryTime,
205         uint256 commissionRate
206     ) public view returns (
207         bool
208     ) {
209         if (commissionRate > _maxCommissionRate || referrer == referee || now > expiryTime) {
210
211             return false;
212         }
213
214         bytes32 hashedData = keccak256(
215             abi.encodePacked(
216                 referrer,
217                 referee,
218                 expiryTime,
219                 commissionRate
220             )
221         );
222
223         address signer = SigUtil.recover(
224             keccak256(
225                 abi.encodePacked("\x19Ethereum Signed Message:\n32", hashedData)
226             ),
227             signature
228         );
229
230         if (_previousSigningWallets[signer] >= now) {
231             return true;
232         }

```

```
233     return _signingWallet == signer;
234 }
```

✔ The code meets the specification.

## Formal Verification Request 58

Method will not encounter an assertion failure.

📅 20, Feb 2020

🕒 0.49 ms

Line 199 in File ReferralValidator.sol

```
199 // @CTK NO_ASF
```

Line 200-236 in File ReferralValidator.sol

```
200     function isReferralValid(
201         bytes memory signature,
202         address referrer,
203         address referee,
204         uint256 expiryTime,
205         uint256 commissionRate
206     ) public view returns (
207         bool
208     ) {
209         if (commissionRate > _maxCommissionRate || referrer == referee || now > expiryTime) {
210             return false;
211         }
212
213         bytes32 hashedData = keccak256(
214             abi.encodePacked(
215                 referrer,
216                 referee,
217                 expiryTime,
218                 commissionRate
219             )
220         );
221
222         address signer = SigUtil.recover(
223             keccak256(
224                 abi.encodePacked("\x19Ethereum Signed Message:\n32", hashedData)
225             ),
226             signature
227         );
228
229         if (_previousSigningWallets[signer] >= now) {
230             return true;
231         }
232
233         return _signingWallet == signer;
234     }
```

✔ The code meets the specification.



## Source Code with CertiK Labels

File SigUtil.sol

```

1  pragma solidity ^0.5.2;
2
3  library SigUtil {
4      //@CTK_NO_OVERFLOW
5      //@CTK_NO_BUF_OVERFLOW
6      //@CTK_NO_ASF
7      function recover(bytes32 hash, bytes memory sig)
8          internal
9          pure
10         returns (address recovered)
11     {
12         require(sig.length == 65);
13
14         bytes32 r;
15         bytes32 s;
16         uint8 v;
17         assembly {
18             r := mload(add(sig, 32))
19             s := mload(add(sig, 64))
20             v := byte(0, mload(add(sig, 96)))
21         }
22
23         // Version of signature should be 27 or 28, but 0 and 1 are also possible versions
24         if (v < 27) {
25             v += 27;
26         }
27         require(v == 27 || v == 28);
28
29         recovered = ecrecover(hash, v, r, s);
30         require(recovered != address(0));
31     }
32
33     //@CTK_NO_OVERFLOW
34     //@CTK_NO_BUF_OVERFLOW
35     //@CTK_NO_ASF
36     function recoverWithZeroOnFailure(bytes32 hash, bytes memory sig)
37         internal
38         pure
39         returns (address)
40     {
41         if (sig.length != 65) {
42             return address(0);
43         }
44
45         bytes32 r;
46         bytes32 s;
47         uint8 v;
48         assembly {
49             r := mload(add(sig, 32))
50             s := mload(add(sig, 64))
51             v := byte(0, mload(add(sig, 96)))
52         }
53
54         // Version of signature should be 27 or 28, but 0 and 1 are also possible versions

```

```

55     if (v < 27) {
56         v += 27;
57     }
58
59     if (v != 27 && v != 28) {
60         return (address(0));
61     } else {
62         return ecrecover(hash, v, r, s);
63     }
64 }
65
66 // Builds a prefixed hash to mimic the behavior of eth_sign.
67 function prefixed(bytes32 hash) internal pure returns (bytes memory) {
68     return abi.encodePacked("\x19Ethereum Signed Message:\n32", hash);
69 }
70 }

```

#### File LandSaleWithReferral.sol

```

1  /* solhint-disable not-rely-on-time, func-order */
2
3  pragma solidity 0.5.9;
4
5  import "../sandbox-private-contracts/contracts_common/src/Libraries/SafeMathWithRequire.sol";
6  import "../labeled-I-LandSale/Land.sol";
7  import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC20.sol";
8  import "../sandbox-private-contracts/contracts_common/src/BaseWithStorage/
9      MetaTransactionReceiver.sol";
10 import "../sandbox-private-contracts/contracts_common/src/Interfaces/Medianizer.sol";
11 import "./ReferralValidator.sol";
12
13 /**
14  * @title Land Sale contract with referral that supports also DAI and ETH as payment
15  * @notice This contract manages the sale of our lands
16  */
17 contract LandSaleWithReferral is MetaTransactionReceiver, ReferralValidator {
18     using SafeMathWithRequire for uint256;
19
20     uint256 internal constant GRID_SIZE = 408; // 408 is the size of the Land
21     uint256 internal constant daiPrice = 14400000000000000;
22
23     Land internal _land;
24     ERC20 internal _sand;
25     Medianizer private _medianizer;
26     ERC20 private _dai;
27
28     address payable internal _wallet;
29     uint256 internal _expiryTime;
30     bytes32 internal _merkleRoot;
31
32     bool _sandEnabled = false;
33     bool _etherEnabled = true;
34     bool _daiEnabled = false;
35
36     event LandQuadPurchased(
37         address indexed buyer,
38         address indexed to,

```

```

39     uint256 indexed topCornerId,
40     uint256 size,
41     uint256 price,
42     address token,
43     uint256 amountPaid
44 );
45
46 //@CTK NO_OVERFLOW
47 //@CTK NO_BUF_OVERFLOW
48 //@CTK NO_ASF
49 /*@CTK LandSale
50     @tag assume_completion
51     @post __post._land == landAddress
52     @post __post._sand == sandContractAddress
53     @post __post._metaTransactionContracts[initialMetaTx] == true
54     @post __post._admin == admin
55     @post __post._wallet == initialWalletAddress
56     @post __post._merkleRoot == merkleRoot
57     @post __post._expiryTime == expiryTime
58     @post __post._medianizer == medianizerContractAddress
59     @post __post._dai == daiTokenContractAddress
60     @post __post._signingWallet == initialSigningWallet
61     @post __post._maxCommissionRate == initialMaxCommissionRate
62 */
63 constructor(
64     address landAddress,
65     address sandContractAddress,
66     address initialMetaTx,
67     address admin,
68     address payable initialWalletAddress,
69     bytes32 merkleRoot,
70     uint256 expiryTime,
71     address medianizerContractAddress,
72     address daiTokenContractAddress,
73     address initialSigningWallet,
74     uint256 initialMaxCommissionRate
75 ) public ReferralValidator(
76     initialSigningWallet,
77     initialMaxCommissionRate
78 ) {
79     _land = Land(landAddress);
80     _sand = ERC20(sandContractAddress);
81     _setMetaTransactionProcessor(initialMetaTx, true);
82     _wallet = initialWalletAddress;
83     _merkleRoot = merkleRoot;
84     _expiryTime = expiryTime;
85     _medianizer = Medianizer(medianizerContractAddress);
86     _dai = ERC20(daiTokenContractAddress);
87     _admin = admin;
88 }
89
90 /// @notice set the wallet receiving the proceeds
91 /// @param newWallet address of the new receiving wallet
92 /*@CTK setReceivingWallet_require
93     @tag assume_completion
94     @post newWallet != address(0)
95     @post msg.sender == _admin
96 */

```

```

97  /*@CTK setReceivingWallet_change
98     @tag assume_completion
99     @post __post._wallet == newWallet
100  */
101  function setReceivingWallet(address payable newWallet) external{
102     require(newWallet != address(0), "receiving wallet cannot be zero address");
103     require(msg.sender == _admin, "only admin can change the receiving wallet");
104     _wallet = newWallet;
105  }
106
107  /// @notice enable/disable DAI payment for Lands
108  /// @param enabled whether to enable or disable
109  /*@CTK setDAIEnabled_require
110     @tag assume_completion
111     @post msg.sender == _admin
112  */
113  /*@CTK setDAIEnabled_change
114     @tag assume_completion
115     @post __post._daiEnabled == enabled
116  */
117  function setDAIEnabled(bool enabled) external {
118     require(msg.sender == _admin, "only admin can enable/disable DAI");
119     _daiEnabled = enabled;
120  }
121
122  /// @notice return whether DAI payments are enabled
123  /// @return whether DAI payments are enabled
124  /*@CTK isDAIEnabled
125     @post __return == _daiEnabled
126  */
127  function isDAIEnabled() external view returns (bool) {
128     return _daiEnabled;
129  }
130
131  /// @notice enable/disable ETH payment for Lands
132  /// @param enabled whether to enable or disable
133  /*@CTK setETHEnabled_require
134     @tag assume_completion
135     @post msg.sender == _admin
136  */
137  /*@CTK setETHEnabled_change
138     @tag assume_completion
139     @post __post._etherEnabled == enabled
140  */
141  function setETHEnabled(bool enabled) external {
142     require(msg.sender == _admin, "only admin can enable/disable ETH");
143     _etherEnabled = enabled;
144  }
145
146  /// @notice return whether ETH payments are enabled
147  /// @return whether ETH payments are enabled
148  /*@CTK isETHEnabled
149     @post __return == _etherEnabled
150  */
151  function isETHEnabled() external view returns (bool) {
152     return _etherEnabled;
153  }
154

```

```

155  /// @notice enable/disable the specific SAND payment for Lands
156  /// @param enabled whether to enable or disable
157  /*@CTK setSANDEnabled_require
158     @tag assume_completion
159     @post msg.sender == _admin
160  */
161  /*@CTK setSANDEnabled_change
162     @tag assume_completion
163     @post __post._sandEnabled == enabled
164  */
165  function setSANDEnabled(bool enabled) external {
166     require(msg.sender == _admin, "only admin can enable/disable SAND");
167     _sandEnabled = enabled;
168  }
169
170  /// @notice return whether the specific SAND payments are enabled
171  /// @return whether the specific SAND payments are enabled
172  /*@CTK isSANDEnabled
173     @post __return == _sandEnabled
174  */
175  function isSANDEnabled() external view returns (bool) {
176     return _sandEnabled;
177  }
178
179  /*@CTK _checkValidity
180     @tag assume_completion
181     @post now < _expiryTime
182     @post buyer == msg.sender \\/ _metaTransactionContracts[msg.sender] == true
183     @post reserved == address(0) \\/ reserved == buyer
184  */
185  function _checkValidity(
186     address buyer,
187     address reserved,
188     uint256 x,
189     uint256 y,
190     uint256 size,
191     uint256 price,
192     bytes32 salt,
193     bytes32[] memory proof
194 ) internal view {
195     /* solium-disable-next-line security/no-block-members */
196     require(block.timestamp < _expiryTime, "sale is over");
197     require(buyer == msg.sender || _metaTransactionContracts[msg.sender], "not
198         authorized");
199     require(reserved == address(0) || reserved == buyer, "cannot buy reserved Land");
200     bytes32 leaf = _generateLandHash(x, y, size, price, reserved, salt);
201     require(
202         _verify(proof, leaf),
203         "Invalid land provided"
204     );
205  }
206
207  function _mint(address buyer, address to, uint256 x, uint256 y, uint256 size, uint256
208     price, address token, uint256 tokenAmount) internal {
209     _land.mintQuad(to, size, x, y, "");
210     emit LandQuadPurchased(buyer, to, x + (y * GRID_SIZE), size, price, token,
211         tokenAmount);

```

```

210 }
211
212 /**
213  * @notice buy Land with SAND using the merkle proof associated with it
214  * @param buyer address that perform the payment
215  * @param to address that will own the purchased Land
216  * @param reserved the reserved address (if any)
217  * @param x x coordinate of the Land
218  * @param y y coordinate of the Land
219  * @param size size of the pack of Land to purchase
220  * @param priceInSand price in SAND to purchase that Land
221  * @param proof merkleProof for that particular Land
222  * @return The address of the operator
223  */
224 // @CTK NO_OVERFLOW
225 // @CTK NO_BUF_OVERFLOW
226 // @CTK NO_ASF
227 /* @CTK buyLandWithSand
228  @tag assume_completion
229  @post _sandEnabled == true
230  @post buyer == msg.sender \\/ _metaTransactionContracts[msg.sender] == true
231  @post reserved == address(0) \\/ reserved == buyer
232  */
233 function buyLandWithSand(
234     address buyer,
235     address to,
236     address reserved,
237     uint256 x,
238     uint256 y,
239     uint256 size,
240     uint256 priceInSand,
241     bytes32 salt,
242     bytes32[] calldata proof,
243     bytes calldata referral
244 ) external {
245     require(_sandEnabled, "sand payments not enabled");
246     _checkValidity(buyer, reserved, x, y, size, priceInSand, salt, proof);
247
248     handleReferralWithERC20(
249         buyer,
250         priceInSand,
251         referral,
252         _wallet,
253         address(_sand)
254     );
255
256     _mint(buyer, to, x, y, size, priceInSand, address(_sand), priceInSand);
257 }
258
259 /**
260  * @notice buy Land with ETH using the merkle proof associated with it
261  * @param buyer address that perform the payment
262  * @param to address that will own the purchased Land
263  * @param reserved the reserved address (if any)
264  * @param x x coordinate of the Land
265  * @param y y coordinate of the Land
266  * @param size size of the pack of Land to purchase
267  * @param priceInSand price in SAND to purchase that Land

```

```

268     * @param proof merkleProof for that particular Land
269     * @param referral the referral used by the buyer
270     * @return The address of the operator
271     */
272     //@CTK NO_OVERFLOW
273     //@CTK NO_BUF_OVERFLOW
274     /*@CTK buyLandWithETH_require
275     @tag assume_completion
276     @post _sandEnabled == true
277     @post buyer == msg.sender \/_metaTransactionContracts[msg.sender] == true
278     @post reserved == address(0) \/_reserved == buyer
279     */
280     function buyLandWithETH(
281         address buyer,
282         address to,
283         address reserved,
284         uint256 x,
285         uint256 y,
286         uint256 size,
287         uint256 priceInSand,
288         bytes32 salt,
289         bytes32[] calldata proof,
290         bytes calldata referral
291     ) external payable {
292         require(!_etherEnabled, "ether payments not enabled");
293         _checkValidity(buyer, reserved, x, y, size, priceInSand, salt, proof);
294
295         uint256 ETHRequired = getEtherAmountWithSAND(priceInSand);
296         require(msg.value >= ETHRequired, "not enough ether sent");
297
298         if (msg.value - ETHRequired > 0) {
299             msg.sender.transfer(msg.value - ETHRequired); // refund extra
300         }
301
302         handleReferralWithETH(
303             ETHRequired,
304             referral,
305             _wallet
306         );
307
308         _mint(buyer, to, x, y, size, priceInSand, address(0), ETHRequired);
309     }
310
311     /**
312     * @notice buy Land with DAI using the merkle proof associated with it
313     * @param buyer address that perform the payment
314     * @param to address that will own the purchased Land
315     * @param reserved the reserved address (if any)
316     * @param x x coordinate of the Land
317     * @param y y coordinate of the Land
318     * @param size size of the pack of Land to purchase
319     * @param priceInSand price in SAND to purchase that Land
320     * @param proof merkleProof for that particular Land
321     * @return The address of the operator
322     */
323     //@CTK NO_OVERFLOW
324     //@CTK NO_BUF_OVERFLOW
325     /*@CTK buyLandWithDAI

```

```

326     @tag assume_completion
327     @post _sandEnabled == true
328     @post buyer == msg.sender \\/ _metaTransactionContracts[msg.sender] == true
329     @post reserved == address(0) \\/ reserved == buyer
330     */
331     function buyLandWithDAI(
332         address buyer,
333         address to,
334         address reserved,
335         uint256 x,
336         uint256 y,
337         uint256 size,
338         uint256 priceInSand,
339         bytes32 salt,
340         bytes32[] calldata proof,
341         bytes calldata referral
342     ) external {
343         require(!_daiEnabled, "dai payments not enabled");
344         _checkValidity(buyer, reserved, x, y, size, priceInSand, salt, proof);
345
346         uint256 DAIRequired = priceInSand.mul(daiPrice).div(1000000000000000000);
347
348         handleReferralWithERC20(
349             buyer,
350             DAIRequired,
351             referral,
352             _wallet,
353             address(_dai)
354         );
355
356         _mint(buyer, to, x, y, size, priceInSand, address(_dai), DAIRequired);
357     }
358
359     /**
360     * @notice Gets the expiry time for the current sale
361     * @return The expiry time, as a unix epoch
362     */
363     /*@CTK getExpiryTime
364     @post __return == _expiryTime
365     */
366     function getExpiryTime() external view returns(uint256) {
367         return _expiryTime;
368     }
369
370     /**
371     * @notice Gets the Merkle root associated with the current sale
372     * @return The Merkle root, as a bytes32 hash
373     */
374     /*@CTK merkleRoot
375     @post __return == _merkleRoot
376     */
377     function merkleRoot() external view returns(bytes32) {
378         return _merkleRoot;
379     }
380
381     /*@CTK NO_OVERFLOW
382     /*@CTK NO_BUF_OVERFLOW
383     /*@CTK NO_ASF

```



```

384 function _generateLandHash(
385     uint256 x,
386     uint256 y,
387     uint256 size,
388     uint256 price,
389     address reserved,
390     bytes32 salt
391 ) internal pure returns (
392     bytes32
393 ) {
394     return keccak256(
395         abi.encodePacked(
396             x,
397             y,
398             size,
399             price,
400             reserved,
401             salt
402         )
403     );
404 }
405
406 //@CTK NO_OVERFLOW
407 //@CTK NO_BUF_OVERFLOW
408 //@CTK NO_ASF
409 function _verify(bytes32[] memory proof, bytes32 leaf) internal view returns (bool) {
410     bytes32 computedHash = leaf;
411
412     /*@CTK ForLoop_verify
413     @inv true
414     */
415     for (uint256 i = 0; i < proof.length; i++) {
416         bytes32 proofElement = proof[i];
417
418         if (computedHash < proofElement) {
419             computedHash = keccak256(abi.encodePacked(computedHash, proofElement));
420         } else {
421             computedHash = keccak256(abi.encodePacked(proofElement, computedHash));
422         }
423     }
424
425     return computedHash == _merkleRoot;
426 }
427
428 /**
429  * @notice Returns the amount of ETH for a specific amount of SAND
430  * @param sandAmount An amount of SAND
431  * @return The amount of ETH
432  */
433 //@CTK NO_OVERFLOW
434 //@CTK NO_BUF_OVERFLOW
435 function getEtherAmountWithSAND(uint256 sandAmount) public view returns (uint256) {
436     uint256 ethUsdPair = getEthUsdPair();
437     return sandAmount.mul(daiPrice).div(ethUsdPair);
438 }
439
440 /**
441  * @notice Gets the ETHUSD pair from the Medianizer contract

```

```

442     * @return The pair as an uint256
443     */
444     function getEthUsdPair() internal view returns (uint256) {
445         bytes32 pair = _medianizer.read();
446         return uint256(pair);
447     }
448 }

```

### File ReferralValidator.sol

```

1  /* solhint-disable not-rely-on-time, func-order */
2
3  pragma solidity 0.5.9;
4
5  import "../sandbox-private-contracts/contracts_common/src/Libraries/SigUtil.sol";
6  import "../sandbox-private-contracts/contracts_common/src/Libraries/SafeMathWithRequire.sol";
7  import "../sandbox-private-contracts/contracts_common/src/Interfaces/ERC20.sol";
8  import "../sandbox-private-contracts/contracts_common/src/BaseWithStorage/Admin.sol";
9
10
11 /**
12  * @title Referral Validator
13  * @notice This contract verifies if a referral is valid
14  */
15 contract ReferralValidator is Admin {
16     address private _signingWallet;
17     uint256 private _maxCommissionRate;
18
19     mapping (address => uint256) private _previousSigningWallets;
20     uint256 private _previousSigningDelay = 60 * 60 * 24 * 10;
21
22     event ReferralUsed(
23         address indexed referrer,
24         address indexed referee,
25         address indexed token,
26         uint256 amount,
27         uint256 commission,
28         uint256 commissionRate
29     );
30
31     /*@CTK ReferralValidator
32     @post __post._signingWallet == initialSigningWallet
33     @post __post._maxCommissionRate == initialMaxCommissionRate
34     */
35     constructor(
36         address initialSigningWallet,
37         uint256 initialMaxCommissionRate
38     ) public {
39         _signingWallet = initialSigningWallet;
40         _maxCommissionRate = initialMaxCommissionRate;
41     }
42
43     /**
44     * @notice Update the signing wallet
45     * @param newSigningWallet The new address of the signing wallet
46     */
47     /*@CTK NO_BUF_OVERFLOW
48     /*@CTK NO_ASF

```

```

49  /*@CTK updateSigningWallet_require
50     @tag assume_completion
51     @post _admin == msg.sender
52  */
53  /*@CTK updateSigningWallet_change
54     @tag assume_completion
55     @pre _admin == msg.sender
56     @post __post._previousSigningWallets[_signingWallet] == now + _previousSigningDelay
57     @post __post._signingWallet == newSigningWallet
58  */
59  function updateSigningWallet(address newSigningWallet) external {
60     require(_admin == msg.sender, "Sender not admin");
61     _previousSigningWallets[_signingWallet] = now + _previousSigningDelay;
62     _signingWallet = newSigningWallet;
63  }
64
65  // TODO: Check if this function is really useful
66  /**
67   * @notice Update the maximum commission rate
68   * @param newMaxCommissionRate The new maximum commission rate
69   */
70  /*@CTK updateMaxCommissionRate_require
71     @tag assume_completion
72     @post _admin == msg.sender
73  */
74  /*@CTK updateMaxCommissionRate_change
75     @tag assume_completion
76     @pre _admin == msg.sender
77     @post __post._maxCommissionRate == newMaxCommissionRate
78  */
79  function updateMaxCommissionRate(uint256 newMaxCommissionRate) external {
80     require(_admin == msg.sender, "Sender not admin");
81     _maxCommissionRate = newMaxCommissionRate;
82  }
83
84  /*@CTK NO_OVERFLOW
85  /*@CTK NO_BUF_OVERFLOW
86  function handleReferralWithETH(
87     uint256 amount,
88     bytes memory referral,
89     address payable destination
90 ) internal {
91     uint256 amountForDestination = amount;
92
93     if (referral.length > 0) {
94         (
95             bytes memory signature,
96             address referrer,
97             address referee,
98             uint256 expiryTime,
99             uint256 commissionRate
100        ) = decodeReferral(referral);
101
102        uint256 commission = 0;
103
104        if (isReferralValid(signature, referrer, referee, expiryTime, commissionRate)) {
105            commission = SafeMathWithRequire.div(
106                SafeMathWithRequire.mul(amount, commissionRate),

```

```

107         10000
108     );
109
110     emit ReferralUsed(
111         referrer,
112         referee,
113         address(0),
114         amount,
115         commission,
116         commissionRate
117     );
118     amountForDestination = SafeMathWithRequire.sub(
119         amountForDestination,
120         commission
121     );
122 }
123
124 if (commission > 0) {
125     address(uint160(referrer)).transfer(commission);
126 }
127 }
128
129 destination.transfer(amountForDestination);
130 }
131
132 //@CTK_NO_OVERFLOW
133 //@CTK_NO_BUF_OVERFLOW
134 //@CTK_NO_ASF
135 function handleReferralWithERC20(
136     address buyer,
137     uint256 amount,
138     bytes memory referral,
139     address payable destination,
140     address tokenAddress
141 ) internal {
142     ERC20 token = ERC20(tokenAddress);
143     uint256 amountForDestination = amount;
144
145     if (referral.length > 0) {
146         (
147             bytes memory signature,
148             address referrer,
149             address referee,
150             uint256 expiryTime,
151             uint256 commissionRate
152         ) = decodeReferral(referral);
153
154         uint256 commission = 0;
155
156         if (isReferralValid(signature, referrer, referee, expiryTime, commissionRate)) {
157             commission = SafeMathWithRequire.div(
158                 SafeMathWithRequire.mul(amount, commissionRate),
159                 10000
160             );
161
162             emit ReferralUsed(
163                 referrer,
164                 referee,

```

```

165         tokenAddress,
166         amount,
167         commission,
168         commissionRate
169     );
170     amountForDestination = SafeMathWithRequire.sub(
171         amountForDestination,
172         commission
173     );
174 }
175
176     if (commission > 0) {
177         token.transferFrom(buyer, referrer, commission);
178     }
179 }
180
181     token.transferFrom(buyer, destination, amountForDestination);
182 }
183
184 /**
185  * @notice Check if a referral is valid
186  * @param signature The signature to check (signed referral)
187  * @param referrer The address of the referrer
188  * @param referee The address of the referee
189  * @param expiryTime The expiry time of the referral
190  * @param commissionRate The commissionRate of the referral
191  * @return True if the referral is valid
192  */
193 //@CTK NO_OVERFLOW
194 //@CTK NO_BUF_OVERFLOW
195 //@CTK NO_ASF
196 function isReferralValid(
197     bytes memory signature,
198     address referrer,
199     address referee,
200     uint256 expiryTime,
201     uint256 commissionRate
202 ) public view returns (
203     bool
204 ) {
205     if (commissionRate > _maxCommissionRate || referrer == referee || now > expiryTime) {
206
207         return false;
208     }
209
210     bytes32 hashedData = keccak256(
211         abi.encodePacked(
212             referrer,
213             referee,
214             expiryTime,
215             commissionRate
216         )
217     );
218
219     address signer = SigUtil.recover(
220         keccak256(
221             abi.encodePacked("\x19Ethereum Signed Message:\n32", hashedData)

```

```
222     signature
223 );
224
225 if (_previousSigningWallets[signer] >= now) {
226     return true;
227 }
228
229 return _signingWallet == signer;
230 }
231
232 function decodeReferral(
233     bytes memory referral
234 ) public pure returns (
235     bytes memory,
236     address,
237     address,
238     uint256,
239     uint256
240 ) {
241     (
242         bytes memory signature,
243         address referrer,
244         address referee,
245         uint256 expiryTime,
246         uint256 commissionRate
247     ) = abi.decode(referral, (bytes, address, address, uint256, uint256));
248
249     return (
250         signature,
251         referrer,
252         referee,
253         expiryTime,
254         commissionRate
255     );
256 }
257 }
```

