L1-Lottery
Security Review Report

October 7, 2024

Version 2.0

Email	Telephone
audits@extropy.io	+44 1865338228

Data Classification	Client Confidential
Client Name	ZkNoid
Document Title	L1-Lottery Security Review Report
Document Version	2.0
Author	Extropy Audit Team

Contents

1 | Executive Summary

Extropy was contracted to conduct an initial code review and vulnerability assessment of the L1-Lottery protocol.

The protocol, deployed on the Mina blockchain, aims to allow users to play a lottery game: users are able to buy tickets for a certain lottery round choosing a combinations of six numbers from 1 to 9. Each round winning combination will be generated using a seed retrieved from a component who interacts with an external oracle.

Users' tickets that match at least one number from the winning combination are eligible to claim the reward: a share of the treasury, which holds the amount paid by users for tickets of that round, can be withdrawn. If for any reason, the result is not finalized from the oracle in 2 days after the round ended, ticket owners can ask for refunds.

In the initial report 21 security issues were discovered and explained, Section 4 details them. Where possible, we have given recommendations for their resolution.
For the final report we have indicated the status of the issues.

Some issues such as 4.4 are caused more by the current state of the Mina ecosystem than being specific to this project, for example there are trust assumptions being made that need to be pointed out.

A general point is that of the centralisation of design, which is be a problem faced by many applications on Mina. The fact of off chain computation is introducing a potential single point of failure. Users of the game are dependent on the continued existence of the data structures, and functioning of the application.
Furthermore, given the current features of zk oracles, users are required to trust the supplier of the random seeds provided, without being able to check the correctness of the VRF.

Some issues rely for their resolution on code or processes outside the scope of this audit, and as such we are unable to say whether these issues are resolved or not.

2 | Audit summary

This audit, conducted from August 19 2024 to September 9 2024, employed a comprehensive approach using a manual review method. Our examination aimed to ensure the robustness and security of the L1-Lottery codebase.

- The code is taken from the L1-lottery repository.
- The audit was performed on commit e5523e8521c47b389e0457e0eadd42e347ac95af.

2.1 | Audit scope

The following contracts were audited:

Contract	LoC
PLottery.ts	802
Ticket.ts	88
util.ts	52
TicketReduceProof.ts	277
DistributionProof.ts	92
Total	**1311**

It's worth mentioning that the current audit scope does not include several crucial components of the system. Those were treated as black box modules and their internal aspects were not analyzed. Among them:

- RandomManager.ts, which is responsible of the communication with the ZkOn oracle and should ensure that the provided seed is random and hidden until being consumed for generating the current round winning combination by the system. **We are putting complete trust in the integrity of ZkOn oracle not to manipulate the seed in order to favour certain numbers**;
- CustomMerkleMap.ts, which is a custom reimplementation of the native Merkle Map data structure from 'o1js' with some changes, like a different merkle height.

2.2 | Issues Summary

ID	Finding	Status
4.1	[HIGH] No zero check on amounts of tickets bought	Partially resolved
4.2	[MEDIUM] ‘generateNumbersSeed()‘ output can be manipulated by ‘seed‘ input length	Partially resolved
4.3	[MEDIUM] Deprecated ‘computeRootAndKey()‘ is used instead of ‘computeRootAndKeyV2()‘	Resolved
4.4	[MEDIUM] Current design heavily relies on a centralized backend	Acknowledged
4.5	[MEDIUM] Compromised ‘treasury‘ address	Resolved
4.6	[LOW] ‘checkCurrentRound()‘ validates multiple rounds if called during specific slots	Resolved
4.7	[LOW] ‘Ticket.check()‘ does not check combination numbers to be greater than or equal to ‘1‘	Resolved
4.8	[LOW] Outdated copypaste function ‘prefixToField‘ from o1js	Resolved
4.9	[LOW] Missing check on ‘initialBankRoot‘ in ‘reduceTickets()‘	Resolved
4.10	[LOW] A round lasts less than documented	Resolved
4.11	[LOW] Possible hash collision may causes different tickets to have the same ‘ticketId‘	Unresolved
4.12	[LOW] Incorrect check for ticket owner	Resolved
4.13	[INFO] Typos	Resolved
4.14	[INFO] Non standard reimplementation of ‘MerkleMap‘ and ‘MerkleMapWitness‘	Acknowledged
4.15	[INFO] Having 0 among ticket numbers is possible when using ‘random()‘ function	Resolved
4.16	[INFO] Lack of explicit permissions declaration	Unresolved
4.17	[INFO] Dust tokens left in contract when computing ‘payAmount‘	Acknowledged
4.18	[INFO] Rewards may be lower than transaction fees when 6 numbers are guessed correctly in a lottery round	Acknowledged
4.19	[INFO] Unnecessary optional parameters	Resolved
4.20	[INFO] Dead code	Resolved

2.3 | Methodology

2.3.1 | Risk Rating

The risk rating given for issues follows the standard approach of the OWASP Foundation. We combine two factors :

- Likelihood of exploit
- Impact of Exploit

The Categories we use are *Critical*, *High*, *Medium*, *Low* and *Informational* These categories may not align with the categories used by other companies.
The informational category is used to contain suggestions for optimisation (where this is not seen as causing significant impact), or for alternative design or best practices.

2.4 | Approach

The project was assessed mainly by code inspection, with auditors working independently or together, looking for possible exploits. Tests were written were possible to validate the issues found and summary results are attached to this report.

3 | System Overview

As written in the first chapter of this document, the protocol consists of a lottery game: users are able to buy tickets for a certain lottery round choosing a combinations of six numbers from 1 to 9.
Ticket price is fixed at 10 MINA (10e9). That amount is sent from the ticket buyer to the main contract, named 'PLottery'. Only when the winning combination is generated for that specific round, 3% of the ticket price is taken as a fee for the developers and sent to the 'treasury' address.
Users can buy unlimited tickets for each round using multiple transactions, they can also buy more than 1 ticket in a single transaction but with the same combination: a way for them to bet more on a specific combination.

As stated in the project documentation, each round should last around 1 day (480 slots) but this is not reflected in the actual code and mentioned in a finding of the current report.
The winning combination of each round will be generated using a seed retrieved from a component who interacts with the ZKON oracle.
If for any reason, after 2 days (960 slots) the round is not finalized the user can claim the full refund of the tickets he bought (including commissions because they were not yet moved to the 'treasury').

Users' tickets that match at least one number up to six of the winning combination are eligible to claim the reward: a share of the treasury, which holds the amount paid by users for tickets of that round, can then be withdrawn.
To acquire the reward, the user must provide a proof that proves the user is eligible for a reward. This proof includes information that the ticket was not already claimed and that the ticket is winning ticket for the specific round.
The amount that a user can claim as a reward for a winning ticket is calculated as

```
bank[round] * ticketScore / totalScore.
```

To understand how 'ticketScore' is computed please refer to the explanation of **getScore()** in paragraph 3.1.2 . TotalScore is simply the sum of all the ticketScores.

3.1 | Data Structures

L1-Lottery protocol makes large use of data structures like merkle maps, merkle trees and so on. Here is a short overview:

- **MerkleTree**: A Merkle Tree is a binary tree where each leaf node contains a hash of a data block, and each non-leaf node contains a hash of its two child nodes. The root of the tree is a single hash that represents the entire data set.

```
Root = Hash(Hash(H1, H2), Hash(H3, H4))
            /                   \
       Hash(H1, H2)          Hash(H3, H4)
         /      \                /      \
       H1        H2            H3        H4
```

- **MerkleList**: A Merkle List is a specialized type of Merkle Tree where the structure is linear. Each node in the list is linked to the next node, and the hashes are computed sequentially.

```
H1 -> H2 -> H3
```

```
Root = Hash(Hash(H1, H2), H3)
```

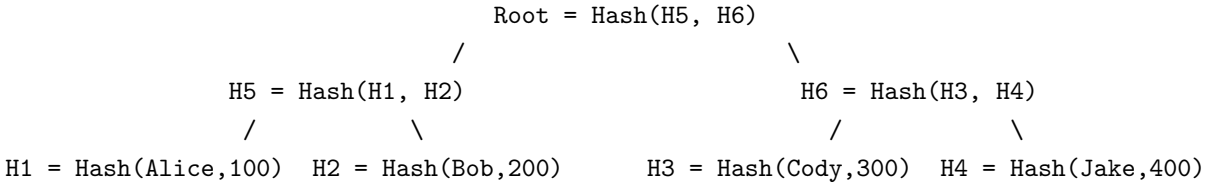
- **MerkleList of MerkleLists**: A Merkle List of Merkle Lists is a hierarchical structure where each element in a primary Merkle List is itself a Merkle List. This allows for organizing and verifying multiple sequences of data within a single structure.

```
First MerkleList:          H1 -> H2 -> H3            Root1 = Hash(Hash(H1, H2), H3)
Second MerkleList:         H4 -> H5 -> H6            Root2 = Hash(Hash(H4, H5), H6)
Third MerkleList:          H7 -> H8 -> H9            Root3 = Hash(Hash(H7, H8), H9)

MerkleList of MerkleLists:
                Root1 -> Root2 -> Root3    FinalRoot = Hash(Hash(Root1, Root2), Root3)
```

- **MerkleMap**: A Merkle Map is a key-value data structure where each entry is associated with a unique hash, and the entire structure can be verified with a Merkle Root. It is like a Merkle Tree but optimized for associative arrays or dictionaries. Each node is the hash of the two child nodes.

```
With the following data:
{ "Alice": 100,
  "Bob": 200,
  "Cody": 300,
  "Jake": 400}
```



Thus a MerkleTree has in the leaves the hashes of generic data, instead MerkleMap has the hash of key-value pairs.

- **MerkleMapWitness**: A Merkle Map Witness is a proof structure used to show that a particular key-value pair is included in a specific Merkle Map without revealing the entire map, by presenting the necessary hashes to compute the root.
  Considering the previous and simple MerkleMap example, generating the MerkleMapWitness for H3, proving the existence of the key-value pair "Charlie: 300", needs only the sibling H4 and H5 without revealing any other data.

```
Witness = [H4 , H5]

Recomputing the root it's simple: Root = Hash(  Hash(H3, H4), H5)
```

- **Zero Hashes Array (Zeroes)**: This is an array of values: considering an empty MerkleTree or MerkleMap, each element of the array is the hash of a generic node of a specific level. Since we are considering an empty structure, it's fine to pick a generic node for each level because they have the same hash:

- zeroes[0] : is the hash of a lead node
- zeroes[1] : is the hash of a node on the level above the leaf level
- ...
- zeroes[merkleMapHeight - 1] : is the root of the empty MerkleMap

The Zero hashes Array can be used to fill in gaps in a Merkle proof to verify the presence or absence of a key-value pair in the map, namely it's used to fill an invalid MerkleTree or MerkleMap.
The following is an invalid MerkleMap because the tree is not complete:

```
                    Root = Hash(H4, H3)
                         /          \
               H4 = Hash(H1, H2)       H3 = Hash(Charlie, 300)
                 /           \
  H1 = Hash(Alice, 100)     H2 = Hash(Bob, 200)
```

Using the zeroes array it will be a valid one:

```
                              Root = Hash(H4, H5)
                            /                       \
                 H4 = Hash(H1, H2)                  H5 = Hash(H3, zeros[0])
                /           \                              /             \
H1 = Hash(Alice, 100)  H2 = Hash(Bob, 200)     H3 = Hash(Charlie, 300)     zeroes[0]
```

3.1.1 | On-chain Merkle roots

In the main protocol contract, 'PLottery.ts' several Merkle Map roots are saved on-chain:

- **ticketRoot**: The root of a two layer Merkle Map where the first contains in each leaf the key-value pair made by the lottery round number and the root of another Merkle Map storing all the tickets sold for that round. The second layer will contain in each leaf the key-value pair made by the 'ticketId' and the 'ticket.hash()'

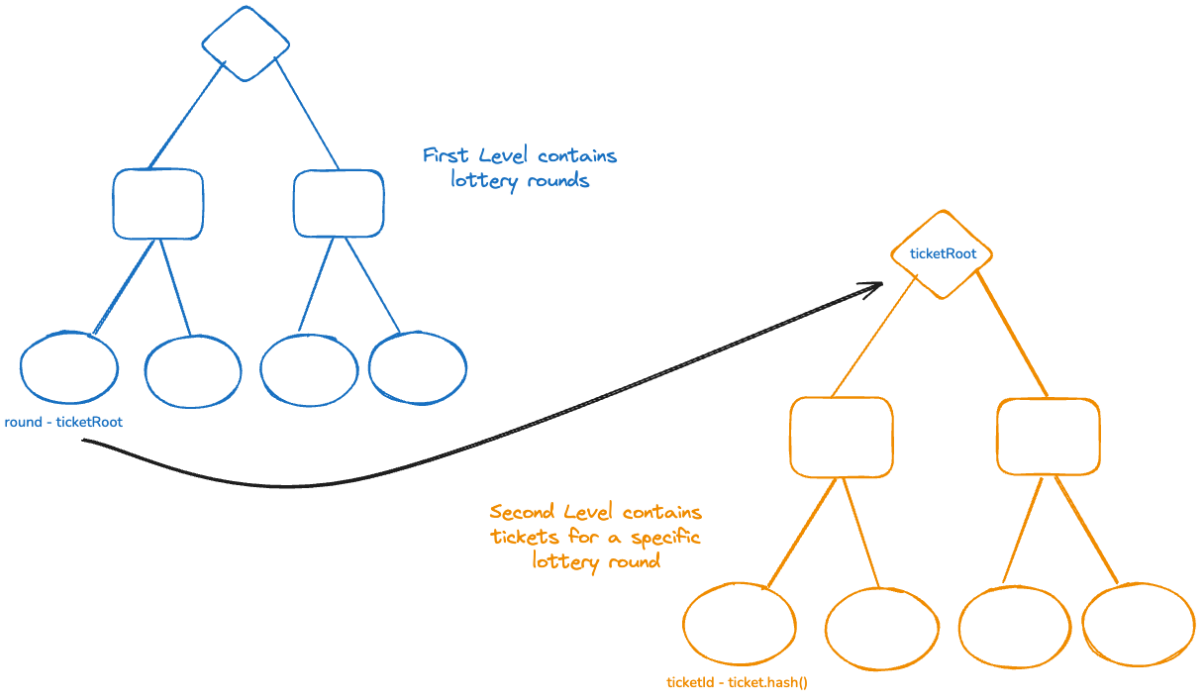


Figure 3.1: 2d Merkle Map graphical representation.

- **ticketNullifier**: The root of a Merkle Map that tracks all used tickets, so one ticket can be redeemed only once. It stores in the leaves the key-value pair made by the nullifierId (computed from round and ticketId values) and a Field element (0 or 1) that marks a ticket as nullified after the reward is claimed for it.
- **bankRoot**: The root of a Merkle Map that stores in the leaves the key-value pair made by the round and the sum of MINA available for that round, namely the sum of purchased ticket prices.
- **roundResultRoot**: The root of a Merkle Map that stores in the leaves the key-value pair made by the round and the corresponding winning combination packed into a single Field element.

3.1.2 | Ticket Struct

Within 'Ticket.ts' the 'Ticket' is defined, containing three fields:

- **numbers**: a Provable Array of six Uint32 values corresponding to the ticket combination;
- **owner**: the PublicKey of the user that bought the current ticket;
- **amount**: a single Uint64 value representing how many tickets, with the same combination, the owner bought in a single buy transaction.

The Ticket struct comes with proper methods and, among them, it's possible to highlight the main ones:

- **check()**: ensures that each of the number in numbers is less than 10;
- **hash()**: calculates the 'ticket.hash()', corresponding to the Poseidon hash of the ticket field concatenation.
- **getScore()**: computes the score that the Ticket gained, after the winning combination was generated for that round. It does so by firstly checking how many numbers of ticket combination match the winning combination (the order matters) generating a value called 'result' from 0 to 6. Then the

value is used for creating the 'condition' array which is an array of Bool containing only one value 'true' in the index specified by 'result'.
For example if result is 3 (meaning 3 numbers matched), the conditions array might look like this:

```
conditions = [false, false, false, true, false, false, false]
```

Finally 'values' array it's calculated using a constant array of scores named 'SCORE_COEFFICIENTS' multiplied by 'ticket.amount'. 'SCORE_COEFFICIENTS' is hardcoded in the codebase and it was calculated by the dev team using a probabilistic approach.

```
SCORE_COEFFICIENTS = [0, 90, 324, 2187, 26244, 590490, 31886460]
ticket.amount = 10
values = [0, 900, 3240, 21870, 262440, 5904900, 318864600]
```

At the end, the element of values with the same index where true is placed in conditions will be returned. In this example the score will be '21870'.

3.2 | Actions & Reducers

Actions & Reducers are a crucial component of the Mina protocol. Actions are public arbitrary information passed within a transaction and stored in archive nodes. This looks similar to 'events' definitiion on the Mina protocol but there a difference: events are a way to log information and notify other components that something happened, with actions it's possible to do complex operation on them using reducers.
Reducers are objects that take a list of actions and allows you to call on them:

- **dispatch()**: dispatching actions is like emitting events, it simply pushes one new action to the account's actionState, which works like the history of the dispatched actions for that account;
- **reduce()**: allows you to write more complex code in order to read and do operations on the actions of the actionState . Below an example of using reduce() to find an action = Field(1000) in a list of actions.

The main use case of Actions & Reducers is for writing zkApps that process concurrent state updates by multiple users.

3.3 | PLottery

The 'Plottery.ts' file contains the smart contract and functions that allow users to interact with the system. The contract uses all the eight available slots of the contract state, including:

- **ticketRoot**, **ticketNullifier**, **bankRoot** and **roundResultRoot** which were already analyzed in paragraph 3.1.1;
- **startBlock**: stores the slot number of contract deployment;
- **lastProcessedState**: stores the last action state that was processed by the reducer;
- **lastReduceInRound**: stores the last round in which reduce was called.
- **lastProcessedTicketId**: stores the id of the last ticket processed;

The contract is made up of several methods to ensure that the protocol works properly. Among the main ones we can highlight:

- **buyTicket()**: allows users to purchase new tickets by sending the amount in MINA to this contract. It does not actually update any Merkle Map, but simply the reducer dispatches the action. An event is emitted at the end.
- **reduceTickets()**: verifies the proof (TicketReduceProof) in input and ensures that the contract's state matches the state described in the proof. It then updates the tickets Merkle Map, populating it with new tickets. An event is emitted at the end.
- **produceResult()**: generates the winning combination for the current round triggering 'generateNumberSeed()' function explained in sections below. After that the 3% fee on the entire bank value (for the round) is moved to the 'treasury' address. An event is emitted at the end.
- **refund()**: allows users to ask for ticket refunds, if after 2 days (960 slots) no winning combination has been generated yet. This function ensures that the ticket owner is the one requesting the refund, verifies the ticket's validity in the Merkle maps, checks that the actually no winning combination has been generated and processes the refund. The refund operations overwrite the 'ticket.hash()' in the 2d Merkle map with 0.
- **getReward()**: verifies the proof (DistributionProof) in input and ensures that the contract's state matches the state described in the proof. Function allows to claim the reward for winning lottery tickets. It calculates ticket score, the totalScore obtained from DistibutionProof and then sends the appropriate portion of bank to winners, as described at the beginning of this chapter. Finally it nullifies the ticket.

3.4 | Proofs

Functions 'reduceTickets()' and 'getReward()' need respectively 'TicketReduceProof' and 'DistributionProof' as input to make their computations.
Mentioned proofs make use of 'ZKPrograms' used to define the steps of a recursive program. ZkProgram() methods execute off-chain. After performing the desired recursive steps, you can settle the interaction on Mina's blockchain by embedding ZkProgram within a SmartContract method that verifies the underlying proof of execution and extracts the output that can be used elsewhere in the method. Inputs to ZkProgram are private by default and are never seen by the Mina network.

3.4.1 | TicketReduceProof

TicketReduceProof contains a ZKProgram with two methods: addTicket(), which is responsible for processing a new action (ticket) and adding it to the actionList, and cutActions(), which is responsible for updating the finalState and emptying the actionList. For each ticket added, addTicket() is invoked recursively by taking the output proof of the previous ticket. Finally cutActions() generates the proof to be used in reduceTicket().

Two main structs are declared within TicketReduceProof.ts and are used for the ZkProgram input and output. Specifically, 'TicketReduceProofPublicInput' which contains the action to add to the ActionList (namely the ticket bought and the related round) and different witnesses useful to compute the root of the needed MerkleMaps.

Instead, 'TicketReduceProofPublicOutput' contains several fields , like the initial and final actionState, old and new roots of MerkleMaps that changed their state after new ticket were reduced (tickets and bank), and other fields to pass the round id of last processed ticket and the latest ticketId processed.

3.4.2 | DistributionProof

DistributionProof contains a ZKProgram with just one method called, again, addTicket() which recursively takes care of processing tickets and calculating the totalScore of the current lottery round.
Two structs are declared within DistributionProof.ts and are used for the ZkProgram input and output. Specifically, 'DistributionProofPublicInput' which takes in the winning combination packed as a single Field element, the ticket and the witness used to compute the new root of the winning ticket processed by using the ticket.hash() as the value.
On the other hand 'DistributionProofPublicOutput' just outputs the new calculated root and the current value of the totalScore

3.5 | Generating seed and winning combination

As mentioned in paragraph 2.1, the RandomManager contract, responsible for the interaction with the ZkOn oracle, is out of scope of the current audit, thus it have been treated as black box. We considered the seed in output to be a valid random one and kept hidden until being consumed for generating the current round winning combination.

Specifically, the random 'seed' is a Field element and it's the input of 'generateNumberSeed()' function in the 'PLottery' contract. The function is supposed to work that way:

1. Converts the 'seed' to an array of 254 Bool, representing the little endian binary representation of this Field element (false corresponds to 0 and true to 1)
2. Creates 6 array slices of 32 Bool elements each; so it uses only the first 192 element of the Bool array (6 * 32 = 192), ignoring the rest
3. For each array slice it computes back a Field element
4. Each of the 6 computed Field elements is constrained to the range [1; 9] using the modulo 9 operation and adding 1 to the result
5. The output of 'generateNumberSeed()' will be an array of Uint32 elements containing the winning combination

4 | Findings

4.1 | [HIGH] No zero check on amounts of tickets bought

- **Location(s):** PLottery.ts#168 , Ticket.ts#49-53
- **Description:** Within `Ticket.ts` the function `check()` is used to ensure ticket numbers validity.

```
check(): Bool {
return this.numbers.reduce(
  (acc, val) => acc.and(val.lessThan(UInt32.from(10))),
  Bool(true)
);
}
```

  However, since no assertion is made on `Ticket.amount` to check the amount of tickets to buy, one can specify it to be zero and call `buyTicket()` function. It allows a malicious actor to dispatch an arbitrary amount of actions under `buyTicket()` without paying the ticket fee, just transaction fee.

```
// Take ticket price from user
let senderUpdate = AccountUpdate.createSigned(
this.sender.getAndRequireSignature()
);
senderUpdate.send({ to: this, amount: TICKET_PRICE.mul(ticket.amount) });
```

  Although we are aware that `Ticket.amount = 0` should be allowed for you to buy the `marker` ticket at the beginning of a new round, a malicious user or an opponent company can leverage this and 'DoS'ing the protocol by flooding the reducer of actions, making it hard to reduce real tickets and causing the lottery process to not work properly.

- **Recommendation:** Consider adding a mechanism that allows only admins to buy ticket with `ticket.amount = 0`.
- **Status:** Partially resolved.
- **Updates:**
  - `[zkNoid, 11/09/2024]`: The client acknowledged the issue and stated: "Most probably we'll add the constraints for the zero ticket buying" .
  - `[zkNoid, 26/09/2024]`: The client stated: "We have solved this error with introduction of minimal pay for ticket(0.03). Later we will add dust release, and this value also will be released. We didn't want to create onlyOwner function, as 1) It will hugely increase contract complexity 2) It will make contract a little bit more centralized"
  - `[Extropy / Lauri, 30/09/2024]`: The fixes in commits 61310477fd342ed9405f474bd99c36e37599be45[1] and e985f1f51346a35c9218274ed9ea86b1e66f8c24[2] address only the DoS problem, making it unfeasible. However this solution forces the backend to have to pay 0.3 MINA for each marker ticket purchase at the beginning of each round.

[1] https://github.com/ZkNoid/L1-lottery/commit/61310477fd342ed9405f474bd99c36e37599be45
[2] https://github.com/ZkNoid/L1-lottery/commit/e985f1f51346a35c9218274ed9ea86b1e66f8c24

4.2 | [MEDIUM] 'generateNumbersSeed()' output can be manipulated by 'seed' input length

- **Location(s):** PLottery.ts#52-61
- **Description:** Within `PLottery.ts` the function `generateNumbersSeed()` is supposed to work that way:

  1. Takes in input a `seed` of type `Field`
  2. Converts the `seed` to an array of 254 Bool, representing the little endian binary representation of this Field element (false corresponds to 0 and true to 1)
  3. For each 32 Bool elements of the array it computes a winning number constraining it to the range [1;9] with `res.mod(9).add(1)`

```
export const generateNumbersSeed = (seed: Field): UInt32[] => {
  let bits = seed.toBits();
  const numbers = [...Array(6)].map((_, i) => {
    let res = UInt32.from(0);
    res.value = Field.fromBits(bits.slice(32 * i, 32 * (i + 1)));
    res = res.mod(9).add(1);
    return res;
  });
  return numbers;
};
```

  However there is no check made on `seed` length and depending on its value the winning combination can be manipulated. Indeed only the first 192 elements of the `bits` array are ever used (32 * 6) while the elements from 193 to 254 are completely ignored. Also if the `seed` in input is a low value then all the last elements of the winning combination will be the same.

  Using an example of the worst case scenario:

  - `seed` is `Field(7)` (in binary `111`)
  - `bits` array is `[true, true, true, false, false, false ....... false ]` with 254 elements in the array
  - the first slice made with the first 32 elements generates a value x (in this example it will be 8 but let's keep it generic)
  - all the other slices of 32 elements will be all full of `false`. This means that the same number, `1` will be generated.
  - the winning combination will be [x, 1, 1, 1, 1 ,1]
  - having `1` for the last 5 numbers of the winning combination happens for all values of `seed` up to `2^32 = 4294967295`.

- **Recommendation:** Consider ensuring the following:

  - If the incoming `seed` is too small, hash it first.
  - Use more bits: take 42 instead of 32, so that bits are not wasted

- **Status:** Partially resolved.
- **Updates:**

- [zkNoid, 30/09/2024]: The client acknowledged the issue and stated: "Seed that is used for this function call is always seed from random manager. And this seed is hash by itself, so no need to hash it additionaly i would say. Don't get how seed can be manipulated, as, it got form RandomManager, however using full Field still will make sense to decrease rounding difference for different numbers."
- [Extropy / Lauri, 30/09/2024]: The incoming seed is out of scope. Also, anyone can call the earlier function `produceResult` with whatever seed, so there are no guarantees about seed quality. Hashing it in `generateNumbersSeed` would guarantee that it's hashed at least once.
- [Extropy / Giuseppe, 01/10/2024]: The commit a8cacfacf6b2e705ef216c1fff402edd464c1c33[3] only fixes one of the issues.

4.3 | [MEDIUM] Deprecated 'computeRootAndKey()' is used instead of 'computeRootAndKeyV2()'

- **Location(s):** CustomMerkleMap.ts#28-49
- **Description:** Although the file is out-of-scope it has been noticed that within `CustomMerkleMap.ts` the function `computeRootAndKey()` is pasted from the official o1js library with the only difference in the for loop: the current implementation iterates until the condition `i < LENGTH20 - 1` is valid while on the original code the condition was `i < 255`. This is fine since `CustomMerkleMap.ts` is a merkle map of height equals to 20.

```
computeRootAndKey(value: Field) {
  let hash = value;

  const isLeft = this.isLefts;
  const siblings = this.siblings;

  let key = Field(0);

  for (let i = 0; i < LENGTH20 - 1; i++) {
    const left = Provable.if(isLeft[i], hash, siblings[i]);
    const right = Provable.if(isLeft[i], siblings[i], hash);
    hash = Poseidon.hash([left, right]);

    const bit = Provable.if(isLeft[i], Field(0), Field(1));

    key = key.mul(2).add(bit);
  }

  return [hash, key];
}
}
```

However `computeRootAndKey()` is a deprecated function and it's replaced in the official lib by computeRootAndKeyV2()[4] since the former had collision problems with merkle map indices as

[3] https://github.com/ZkNoid/L1-lottery/commit/a8cacfacf6b2e705ef216c1fff402edd464c1c33
[4] https://github.com/o1-labs/o1js/blob/4994b449e350a6ed17c0299879381aa40d448fb1/src/lib/provable/merkle-map.ts#L145

discussed here[5].

- **Recommendation:** Consider replacing `computeRootAndKey()` with `computeRootAndKeyV2()` as it is used everywhere in the codebase.
- **Status:** Resolved.
- **Updates:**
  - `[Extropy / Lauri, 30/09/2024]`: Fixed in commit 2d846158e2ec1ab7c8c866bd882eae437b88b62e[6], but still have also the old version (unused)

4.4 | [MEDIUM] Current design heavily relies on a centralized backend

- **Location(s):** -
- **Description:** Current project design works in a way that all the data structures (Merkle Maps) containing ticket informations, round datas, treasury accountings, nullifiers and so on are stored in a centralized back end, while the roots are stored on chain. When there is the need for a user to exhibit the presence of a certain `value` in the related Merkle Map he should input the Merkle Map Witness in order to calculate the `key` and the `root` correctly.

  The project relies on this way of functioning and every operation that should be performed on-chain requires users to be able to get the Merkle Map contents from the backend and calculate the related witness.

  However if the back-end goes down for any reason, not being capable to continue operating properly in an hypothetical failure event (e.g. data lost), users will not able to interact with the protocol. For example they can't claim rewards or ask for refunds and, in the worst case scenario where lottery round can't be finalized properly, users funds will be stuck in the contract.
- **Recommendation:** Consider adding a concrete possibility for users to calculate the trees/witnesses themselves. Be it from past events or something else.
- **Status:** Acknowledged.
- **Updates:**
  - `[zkNoid, 11/09/2024]`: The client acknowledged the issue and stated:"It's not actually. Our backend is opensource and it restores all the data from public archieve nodes. Players can host their own backend that will restore the same data as ours:
    - Backend repo: link[7]
    - Feeding state manager with events: link[8]
    - Mongodb sync with events: link[9] After feeding the updated state manager is used to reconstruct merkle trees: link[10]"

[5] https://github.com/o1-labs/o1js/pull/1694
[6] https://github.com/ZkNoid/L1-lottery/commit/2d846158e2ec1ab7c8c866bd882eae437b88b62e
[7] https://github.com/ZkNoid/lottery-backend
[8] https://github.com/ZkNoid/lottery-backend/blob/e4e5716d094f910c9d210c401b72ad3dfa7e38f3/src/workers/sync-events/sync-events.service.ts#L80
[9] https://github.com/ZkNoid/lottery-backend/blob/main/src/workers/sync-events/sync-events.service.ts#L176
[10] https://github.com/ZkNoid/lottery-backend/blob/e4e5716d094f910c9d210c401b72ad3dfa7e38f3/src/workers/produce-result/produce-result.service.ts

- [zkNoid, 26/09/2024]: The client stated: "PLottery contract is not heavily centralized, as all the data required for tree reconstruction lays on events, so everyone with access to archive node could do it. The only heavy centralization is on random generation, as we are one part of commit reveal process, but there is no solution for this right now, as VRF is still under development"
- [Extropy / Lauri, 30/09/2024]: All of the mentioned backend files are out of scope, no way to estimate in reality.

4.5 | [MEDIUM] Compromised 'treasury' address

- **Location(s):** constants.ts#16-18
- **Description:** Although `constants.ts` is out of scope, it should be mentioned tat the hardcoded `treasury` address, which is supposed to collect lottery ticket fees, has a comment stating that the current address in use is compromised.

```
/// xxQ was compomised previously. Use it for test purpose only
export const treasury = PublicKey.fromBase58(
  'B62qj3DYVUCaTrDnFXkJW34xHUBr9zUorg72pYN3BJTGB4KFdpYjxxQ'
);
```

- **Recommendation:** Consider changing the hardcoded treasury address to a different one.
- **Status:** Resolved.
- **Updates:**
  - [Extropy / Lauri, 30/09/2024]: Fixed in commit 476788c007a0e520cc967d1c920adb469b86de61[11]

4.6 | [LOW] 'checkCurrentRound()' validates multiple rounds if called during specific slots

- **Location(s):** PLottery.ts#559-563
- **Description:** Within `PLottery.ts` the function `checkCurrentRound()` is used within `buyTicket()` to ensure that the ticket being bought belongs to the current lottery `round`.

```
public checkCurrentRound(round: UInt32) {
  const startBlock = this.startBlock.getAndRequireEquals();
  this.network.globalSlotSinceGenesis.requireBetween(
    startBlock.add(round.mul(BLOCK_PER_ROUND)),
    startBlock.add(round.add(1).mul(BLOCK_PER_ROUND))
  );
}
```

However, as from the docs[12], the method `requireBetween()` used in the validation process is an assertion that checks if `globalSlotSinceGenesis` is "between lower and upper (inclusive)." Having the upper value inclusive means that the same slot number can belong to different rounds.

As an example let's suppose the following scenario:

- during the `init()` function the state `startBlock` is set to 604500 using 'globalSlotSinceGenesis

[11] https://github.com/ZkNoid/L1-lottery/commit/476788c007a0e520cc967d1c920adb469b86de61

[12] https://docs.minaprotocol.com/zkapps/writing-a-zkapp/feature-overview/on-chain-values#:~:text=The-,requireBetween,-()%20method%20has

- the first round starts and a user calls `buyTicket()` specifying round = 0
- `BLOCK_PER_ROUND` is a constant equals to 60
- `checkCurrentRound(0)` checks wether the current `globalSlotSinceGenesis` is in the range

  `[604500 + 0*60 ; 604500 + (0+1)*60 ] -> [604500 ; 604560]`

  and if so the execution proceeds
- when the next round starts `checkCurrentRound(1)` validates that now the current `globalSlotSinceGenesis` should be within the range

  `[604500 + 1*60 ; 604500 + (1+1)*60 ] -> [604560 ; 604620]`

Knowing that from the docs "a slot is the Mina-native time unit of 3 minutes", during `slot = 604560`, `checkCurrentRound()` passes for both `round = 0` and `round = 1`. During that time window one can execute `buyTicket()`specifying both rounds, which should not be allowed.

- **Recommendation:** Consider ensuring that when buying tickets, `checkCurrentRound()` will not validate multiple rounds during the time window of a single slot.
- **Status:** Resolved.
- **Updates:**
  - `[Extropy / Lauri, 30/09/2024]`: Fixed in commit dc140f1ce319519f7e40670c1dbdf178cbeec307[13]

4.7 | [LOW] 'Ticket.check()' does not check combination numbers to be greater than or equal to '1'

- **Location(s):** Ticket.ts#49-53
- **Description:** Within `Ticket.ts` the function `check()` ensures that the values of the `numbers` array are all below 10 otherwise `Bool(false)` is returned.

```
check(): Bool {
  return this.numbers.reduce(
    (acc, val) => acc.and(val.lessThan(UInt32.from(10))),
    Bool(true)
  );
}
```

However it does not check that they are also greater than or equal to 1. In the scenario that a user calls `PLottery.buyTicket()` passing values for the ticket numbers below 1, `ticket.check().assertTrue();` will not revert in `buyTicket()`. The user then can buy a ticket with numbers that can never match the ones in the winning combination.

- **Recommendation:** Consider changing the `check()` function in order to ensure that all the ticket numbers are greater than or equal to 1 as well.
- **Status:** Resolved.
- **Updates:**
  - `[Extropy / Lauri, 30/09/2024]`: Fixed in commit a412bab2e6631676e0d2a3d968bdcfd345e89771[14]

[13] https://github.com/ZkNoid/L1-lottery/commit/dc140f1ce319519f7e40670c1dbdf178cbeec307
[14] https://github.com/ZkNoid/L1-lottery/commit/a412bab2e6631676e0d2a3d968bdcfd345e89771

4.8 | [LOW] Outdated copypaste function 'prefixToField' from o1js

- **Location(s):** TicketReduceProof#16-29
- **Description:** Within `TicketReduceProof.ts` the function `prefixToField()` is a copypaste function from an outdated version of the o1js implementation and potentially contains bugs. One obvious bug is that the old implementation assumes the field size is 255 bits, when it's less.
- **Recommendation:** Consider upgrading `prefixToField()` to to the newest version. At the time of writing, new version can be found here[15].
- **Status:** Resolved.
- **Updates:**
  - `[ZkNoid, 30/08/2024]`: The client acknowledged the issue and stated: "we definitely should change it to the code from link, so it would be more clear"
  - `[Extropy / Lauri, 30/09/2024]`: Fixed in commit 64bd74a854e7c173ff16e49b74d88afc3a3529af[16]

4.9 | [LOW] Missing check on 'initialBankRoot' in 'reduceTickets()'

- **Location(s):** PLottery.ts#211
- **Description:** Within `PLottery.ts` the function `reduceTickets()` lacks in checking that the `reduceProof.publicOutput.initialBankRoot` should be equal to the `bankRoot` saved on-chain before updating the contract state to `reduceProof.publicOutput.newTicketRoot`.
- **Recommendation:** Consider adding the following check in `reduceTickets()` before updating the contract state:

```
reduceProof.publicOutput.initialBankRoot.assertEquals(
  bankRoot,
  'Initial bank root does not match on chain bank root'
);
```

- **Status:** Resolved.
- **Updates:**
  - `[Extropy / Lauri, 30/09/2024]`: Fixed in commit c433160f481268461df261ddd86a9b535fb3ef3d[17]

4.10 | [LOW] A round lasts less than documented

- **Location(s):** constants.ts#6
- **Description:** The available project documentation states that a round lasts 480 slots (about 24 hours). More can be found here[18] and here[19].

  However, the used constant `BLOCK_PER_ROUND` states 60 slots, which is about 3 hours. This contradicts the documented value.

[15] https://github.com/o1-labs/o1js-bindings/blob/ff3588199eaf8f423c4474862f049bf1d8ed4c77/lib/binable.ts#L317
[16] https://github.com/ZkNoid/L1-lottery/commit/64bd74a854e7c173ff16e49b74d88afc3a3529af
[17] https://github.com/ZkNoid/L1-lottery/commit/c433160f481268461df261ddd86a9b535fb3ef3d
[18] https://zknoid.medium.com/mina-navigators-l1-lottery-ab2df88c985c
[19] https://forums.minaprotocol.com/t/zknoid-l1-lottery/6269/2

```
export const BLOCK_PER_ROUND = 60; // Aproximate blocks per 3 hours
```

Note that `constants.ts` is out of scope.

- **Recommendation:** Consider either documenting the new round length or fixing the code.
- **Status:** Resolved.
- **Updates:**
  - `[Extropy / Lauri, 30/09/2024]`: Fixed in commit 3854b2c1c2194be372a04c7a29028a47f5dc2cdb[20]

4.11 | [LOW] Possible hash collision may causes different tickets to have the same 'ticketId'

- **Location(s):** Ticket.ts#56-63, util.ts#33-38
- **Description:** Within `Ticket.ts` the function `hash()` calculates the Poseidon hash using only ticket numbers, owner public key and amount.

```
hash(): Field {
  return Poseidon.hash(
    this.numbers
      .map((number) => number.value)
      .concat(this.owner.toFields())
      .concat(this.amount.value)
  );
}
```

However if a single user, through two different `buyTicket()` calls in the same round, buys two tickets using the same numbers and same `ticket.amount` an hash collision occurs since they will have the same `ticket.hash()`.

Example:

- In round 3, userA buys a ticket with amount=1, numbers = [1,2,3,4,5,6] and owner = userAPrvtKey
- then after 1 hour (still during round 3) userA buys again a ticket with amount=1, numbers = [1,2,3,4,5,6] and owner = userAPrvtKey
- both tickets will have the same `ticket.hash()`

Having the same `ticket.hash()` may causes an important consequence: the `ticketId` retrieved from `checkTicket()` in `PLottery.ts` can be the same for both tickets under a specific assumption: the witnesses used to compute 2D Merkle Map roots and keys are the same.

This scenario is possible as we don't know how witnesses are generated in the back-end: we cannot exclude that the back end will return the same witness of the first occurrence of `ticket.hash()` for both ticket.

For two tickets having the same `ticketId` can be really problematic. If the tickets are winning tickets, the owner will be only able to claim the reward for the first one, since after the first call to `getReward()` that `ticketId` will be nullified through `this.checkAndUpdateNullifier()`.

- **Recommendation:** Consider ensuring that tickets bought in a single round by the same user will always have a different `ticket.hash()` even if the same `ticket.amount` and `ticket.numbers` are used.

[20] https://github.com/ZkNoid/L1-lottery/commit/3854b2c1c2194be372a04c7a29028a47f5dc2cdb

- **Status:** Unresolved.
- **Updates:**
  - `[zkNoid, 11/09/2024]`: The client acknowledged the issue and stated: "ticketHash do not used for calculating ticketId. TicketId is combination of round and index of action within this round, that is was reduced. So two equal ticket will have different ticketId, and tho different witness, so there should be no problem with that. "
  - `[zkNoid, 26/09/2024]`: The client stated: "hash collision is not a problem, as hash is not used for ticketId generation. So two similar ticket would have different ticketId and different merkle proof for each of them"
  - `[Extropy / Lauri, 30/09/2024]`: I disagree with the client. Yes, `ticketId` is not calculated based on the `ticket.hash()`, but In `checkTicket` function the `ticketId` is retrieved from trees based on the hash. Even if the trees have two different `ticketIds`, they are stored by the hash and retrieval by that (same) hash twice returns the same ticketId both times.
  - `[Extropy / Giuseppe, 03/10/2024]`: The problem is not the hash collision itself. Also i agree that with two equal `ticket.hash()` but different witnesses you end up with different `ticketId`. The problem stated here is that, depending on backend computations, which are out of scope, we cannot guarantee that for the same `ticket.hash()` the same witness is calculated and then used in PLottery.

4.12 | [LOW] Incorrect check for ticket owner

- **Location(s):** PLottery.ts#167
- **Description:** Within `buyTicket()` the assertion made to check that the transaction sender is the same user that bought the ticket uses the method `equals()`.

```
ticket.owner.equals(this.sender.getAndRequireSignature());
```

  However, `equals()` returns a `Bool` type and specifically `Bool(false)` if the ticket owner is not the sender instead of reverting the execution.
- **Recommendation:** Consider changing `equals()` with `assertEquals()`.
- **Status:** Resolved.
- **Updates:**
  - `[Extropy / Lauri, 30/09/2024]`: Fixed in commit d89f78153a83a6ec058d4c2e456087b27402f487[21]

4.13 | [INFO] Typos

- **Location(s):** constants.ts#10, constants.ts#11, PLottery.ts#97, PLottery.ts#101, PLottery.ts#289, PLottery.ts#535
- **Description:** The codebase contains several typos at the mentioned locations. Some of them belong to out-of-scope files but are mentioned for completeness. Also please that here only typos in variable names are mentioned, but typos in the comments are frequent as well in the codebase.

```
export const PRESICION = 1000; // instead of PRECISION
```

[21] https://github.com/ZkNoid/L1-lottery/commit/d89f78153a83a6ec058d4c2e456087b27402f487

```
export const COMMISION = 30 // instead of COMMISSION

coordiantorAddress: PublicKey = ZkOnCoordinatorAddress // instead of coordinatorAddress

coordiantorAddress // instead of coordinatorAddress

resultWiness: MerkleMap20Witness, // instead of resultWitness

roundResulValue: Field, // instead of roundResultValue
```

- **Recommendation:** Consider fixing the aforementioned typos in order to improve code readability.
- **Status:** Resolved.
- **Updates:**
  - `[Extropy / Lauri, 30/09/2024]`: Fixed in commit 2690b67c42f65272ec791263ee91393bf38cbf98[22]

4.14 | [INFO] Non standard reimplementation of 'MerkleMap' and 'MerkleMapWitness'

- **Location(s):** CustomMerkleMap.ts
- **Description:** Although out of scope, the file `CustomMerkleMap.ts` it's an own implementation of the regular `MerkleMap` and `MerkleMapWitness` classes from `o1js`. Original implementation can be found here[23]. It's clear that the main motivation behind that is the different MerkleMap height, but other small details seems to be different than the original version.

  Since this file is not in scope, no further analysis was performed. No exact exploitation is researched, nor are the exact problems caused.
- **Recommendation:** Consider reusing original implementations. For the tree height consider just inheriting `MerkleMap` and changing the tree height in the constructor.
- **Status:** Acknowledged.
- **Updates:**
  - `[zkNoid, 26/09/2024]`: The client acknowledged the issue and stated: "We cannot reuse MerkleMap code properly as there is some hardcoded values there(https://github.com/o1-labs/o1js/blob/74121af790d7a0978b93afa8914a3f0b7bed72a7/src/lib/provable/merkle-map.ts#L127). So we have to fully copy their code" .
  - `[Extropy / Lauri, 30/09/2024]`: Ok, not worth considering or analyzing further, since this is out of scope

4.15 | [INFO] Having 0 among ticket numbers is possible when using 'random()' function

- **Location(s):** Ticket.ts#42
- **Description:** Within `Ticket.ts` the function `random()` of the `Ticket` class generates 6 random numbers for testing purposes using the code:

```
 return new Ticket({
numbers: [...Array(NUMBERS_IN_TICKET)].map(() =>
  UInt32.from(getRandomInt(10))
),
```

[22] https://github.com/ZkNoid/L1-lottery/commit/2690b67c42f65272ec791263ee91393bf38cbf98
[23] https://github.com/o1-labs/o1js/blob/72b8e2c2b9aaba4b5c12257bdd63dfe2676a3d99/src/lib/provable/merkle-map.ts

However the function `getRandomInt()` called with the input `10` returns values in the range [0;9] which is incorrect since the allowed numbers for a ticket should be in the range [1;9].

Finding severity is set to `INFO` because `random()` function is used for testing only.

- **Recommendation:** Consider ensuring that if `getRandomInt(10)` returns 0, that value will not be used as one of the 6 ticket numbers.
- **Status:** Resolved.
- **Updates:**
  - `[Extropy / Lauri, 30/09/2024]`: Fixed in commit b1d41eea10eb52aa35312ae2ac8a1a45401e4e2f[24]

4.16 | [INFO] Lack of explicit permissions declaration

- **Location(s):** PLottery.ts.sol#138
- **Description:** Within the `init()` function of `PLottery.ts` there are no explicit definitions of permissions. By default, the following permissions are being set:

```
access: none
editActionState: proof
editState: proof
incrementNonce: signature
receive: none
send: proof
setDelegate: signature
setPermissions: signature
setTiming: signature
setTokenSymbol: signature
setVerificationKey: signature
setVotingFor: signature
setZkAppUri: signature
```

  This means that, for example, the contract is upgradable using a signature to set the verification key and above permissions may change in the future since `setPermissions` is set to `signature`.
- **Recommendation:** Consider making permissions explicit in the `init()` function.
- **Status:** Unresolved.
- **Updates:**
  - `[Extropy / Lauri, 30/09/2024]`: The latest code version fixed `setVerificationKey` issue, but still leaves `setPermissions` to default, rendering the fix rather useless. Latest version[25]

4.17 | [INFO] Dust tokens left in contract when computing 'payAmount'

- **Location(s):** PLottery.ts#494
- **Description:** Within `PLottery` contract, when calculating `payAmount` in `getReward()`, rounding issues calculations may eventually result in dust tokens left in the contract and, currently, there is no way to withdraw / reassign this kind of value.

[24] https://github.com/ZkNoid/L1-lottery/commit/b1d41eea10eb52aa35312ae2ac8a1a45401e4e2f
[25] https://github.com/ZkNoid/L1-lottery/blob/e985f1f51346a35c9218274ed9ea86b1e66f8c24/src/PLottery.ts#L161

```
const payAmount = convertToUInt64(bankValue).mul(score).div(totalScore);
```

Example:

- 100 tickets sold -¿ `bankValue = 100 * 10e9`
- one ticket (ticket1) won the highest score , which is `31886460 * ticket1.amount`
- another ticket (ticket2) won the lowest score, which is `90 * ticket2.amount`
- for simplicity let's suppose both `ticket1.amount` and `ticket2.amount` are `1`

When the user who bought ticket2 wants to claim reward, It will be

```
payAmount = 100 *10e9 * 90 / (31886460+90) = 2822506,6681...
// approximated to 2822506
```

When the user who bought ticket1 wants to claim reward, it will be

```
payAmount = 100 *10e9 * 31886460 / (31886460+90) = 999997177493,3318...
// approximated to 999997177493
```

Thus the bank were 100e9 but only `999997177493 + 2822506 = 999_999_999_999` were payed out.

- **Recommendation:** Consider adding an admin-only function to withdraw dust. However, this should only be callable when ending a round and all rewards have been distributed - probably within the same transaction.
- **Status:** Acknowledged.
- **Updates:**
  - `[zkNoid, 26/09/2024]`: The client acknowledged the issue and stated: "We will think about dust release for next version of contract, however not it is not trivial" .

4.18 | [INFO] Rewards may be lower than transaction fees when 6 numbers are guessed correctly in a lottery round

- **Location(s):** PLottery.ts#494
- **Description:** Within `PLottery.ts` the function `getReward()` computes the amount to pay to winning users

```
const payAmount = convertToUInt64(bankValue).mul(score).div(totalScore);
```

However in the scenario that a user guesses correctly all the 6 numbers of the winning combination, all the other users that score lower (namely 1, 2 or 3 numbers) will not be incentivized to ask for the reward because what they will get will be lower than transaction fees. This will cause an amount of MINAs left in the contract and never been withdrawn.

As an example:

- 100 tickets sold -¿ `bankValue = 100 * 10e9` = 1000 MINA
- one ticket (let's call it `ticket_high_score`) won the highest score having guessed correctly 6 numbers out of 6. The score will be `31886460 * ticket_high_score.amount`
- 6 tickets (`ticket_medium_score`) guessed correctly only 3 numbers of of 6. The score will be for each of them `2187 * ticket_low_score.amount`

- 45 tickets (`ticket_low_score`) won the lowest score having guessed correctly 1 number out of 6. The score will be for each of them `90 * ticket_low_score.amount`
- the remaining 48 tickets scored 0.
- for simplicity let's suppose that all the 100 tickets have `ticket.amount` equal to 1
- totalScore for this round is `(1 * 31886460) + (6 * 2187) + (45 * 90) + (48 * 0) = 31903632`

When it comes to calling `getReward()` the following scenario will happen:

1. `ticket_high_score` will have a `payAmount` of:

```
payAmount = 100 *10e9 * 31886460 / 31903632 = 999_461_754_072 | 999.46e9 | 999.46 MINA
```

2. Each `ticket_medium_score` will have a `payAmount` of:

```
payAmount = 100 *10e9 * 2187 / 31903632 = 68_550_188 | 0.068e9 | 0.068 MINA
```

3. Each `ticket_low_score` will have a `payAmount` of:

```
payAmount = 100 *10e9 * 90 / 31903632 = 2_820_995 | 0.002e9 | 0.002 MINA
```

Considering that on average MINA transaction fees are in the range `0.01 - 0.1 MINA` some users may be not incentivized to call `getReward()` since what they get is less than what they should pay in terms of fees. In the above example, if we suppose that transaction fee is 0.1 MINA, the reward for the 45 tickets (`ticket_low_score`) and 6 tickets (`ticket_medium_score`) will very likely not be claimed, leaving some few MINA in the contract for each round.

- **Recommendation:** Consider adding an admin-only function to withdraw dust. However, this should only be callable when ending a round and only if some time passed and winners didn't claim their rewards.
- **Status:** Acknowledged.
- **Updates:**
  - `[zkNoid, 26/09/2024]`: The client acknowledged the issue and stated: "We will think about dust release for next version of contract, however not it is not trivial" .

4.19 | [INFO] Unnecessary optional parameters

- **Location(s):** PLottery.ts#599, PLottery.ts#701, PLottery.ts#726
- **Description:** Within `PLottery.ts` several functions accept for one of the inputs both a Field element or null, which is unnecessary:
  - the function `checkMap()` takes `key` at line 726, but it's called at lines 602 and 705 within `checkAndUpdateMap()` and `checkResult()` passing non null values (see below);
  - the function `checkAndUpdateMap()` takes `key` at line 701, but it's called at lines 653 and 678 always with non null values;
  - the function `checkResult()` takes `round` at line 599, , but it's called at line 488 with the output of `checkTicket()` which is non null and at line 384 with the output of `checkAndUpdateTicket()` which is non null
- **Recommendation:** Consider changing mentioned function signatures in order to accept only Field input types.

- **Status:** Resolved.
- **Updates:**
  - `[Extropy / Lauri, 30/09/2024]`: Fixed in commit e242f6588ddac613f26309d11ac324055fde3f05[26]

4.20 | [INFO] Dead code

- **Location(s):** PLottery.ts#397-403, PLottery.ts#48-50, PLottery.ts#582-586
- **Description:** Within `PLottery.ts` the function `refund()` contains commented out code that seems to be a leftover code from an old version of the codebase.

```
// Check and update nullifier
// this.checkAndUpdateNullifier(
//   nullifierWitness,
//   getNullifierId(round, ticketId),
//   Field(0),
//   Field.from(1)
// );
```

  Indeed the `refund()` function currently works by changing the `ticket.hash()` of the refunded ticket to 0 and then recomputing the `ticketRoot`. Instead, the leftover code was supposed to set the ticket nullifier to 1 during the refund process as it happens when a user claims the round rewards.

  Furthermore, also `mockResult` and `getWiningNumbersForRound()` are unused code and can be removed.

```
export const mockResult = NumberPacked.pack(
  mockWinningCombination.map((v) => UInt32.from(v))
);

// public getWiningNumbersForRound(): UInt32[] {
//   return mockWinningCombination.map((val) => UInt32.from(val));
//   // // Temporary function implementation. Later will be switch with oracle call.
//   // return generateNumbersSeed(Field(12345));
// }
```

- **Recommendation:** Consider removing old code in order to improve readability and maintainability.
- **Status:** Partially resolved.
- **Updates:**
  - `[Extropy / Lauri, 30/09/2024]`: `mockResult` still remains in the code, but other points have been fixed in commit e10620a1705858a5f891b33fd15236803b2535b6[27].

[26] https://github.com/ZkNoid/L1-lottery/commit/e242f6588ddac613f26309d11ac324055fde3f05
[27] https://github.com/ZkNoid/L1-lottery/commit/e10620a1705858a5f891b33fd15236803b2535b6

5 | Test Coverage

5.1 | Npm tests

```
> l1-lottery-contracts@0.7.14 test
> node --experimental-vm-modules node_modules/jest/bin/jest.js

(node:683) ExperimentalWarning: VM Modules is an experimental feature and
    might change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
(node:767) ExperimentalWarning: VM Modules is an experimental feature and
    might change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
(node:766) ExperimentalWarning: VM Modules is an experimental feature and
    might change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
ts-jest[versions] (WARN) Version 5.5.4 of typescript installed has not been
    tested with ts-jest. If you're experiencing issues, consider using a
    supported version (>=4.3.0 <5.0.0-0). Please do not report issues in
    ts-jest if you are using unsupported versions.
ts-jest[versions] (WARN) Version 5.5.4 of typescript installed has not been
    tested with ts-jest. If you're experiencing issues, consider using a
    supported version (>=4.3.0 <5.0.0-0). Please do not report issues in
    ts-jest if you are using unsupported versions.
(node:767) [DEP0040] DeprecationWarning: The `punycode` module is deprecated.
    Please use a userland alternative instead.
(node:766) [DEP0040] DeprecationWarning: The `punycode` module is deprecated.
    Please use a userland alternative instead.
 PASS  src/Tests/Random.test.ts (6.453 s)
  Add
     JSON works (4008 ms)
     skipped Sould produce random value

  console.log
    Process ticket: <0> <0>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <1> <0>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <0> <0>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <0> <1>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <3> <0>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Bank: 9700000000
      at Object.<anonymous> (src/Tests/PLottery.test.ts:428:13)
```

```
console.log
  undefined
    at Object.<anonymous> (src/Tests/PLottery.test.ts:429:13)

console.log
  Process: 0 round
    at Object.<anonymous> (src/Tests/PLottery.test.ts:439:15)

console.log
  Process ticket: <0> <0>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <0> <1>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <0> <2>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <0> <3>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <0> <4>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <0> <5>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <0> <6>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <0> <7>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <0> <8>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <0> <9>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <1> <0>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process: 1 round
    at Object.<anonymous> (src/Tests/PLottery.test.ts:439:15)

console.log
```

```
    Process ticket: <1> <1>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <1> <2>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <1> <3>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <1> <4>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <1> <5>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <1> <6>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <1> <7>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <1> <8>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <1> <9>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <1> <10>

      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <2> <0>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process: 2 round
      at Object.<anonymous> (src/Tests/PLottery.test.ts:439:15)

  console.log
    Process ticket: <2> <1>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <2> <2>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <2> <3>
```

```
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <2> <4>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <2> <5>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <2> <6>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <2> <7>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <2> <8>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <2> <9>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <2> <10>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <3> <0>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process: 3 round
    at Object.<anonymous> (src/Tests/PLottery.test.ts:439:15)

console.log
  Process ticket: <3> <1>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <3> <2>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <3> <3>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <3> <4>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <3> <5>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)
```

```
console.log
  Process ticket: <3> <6>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <3> <7>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <3> <8>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <3> <9>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <3> <10>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <4> <0>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process: 4 round
    at Object.<anonymous> (src/Tests/PLottery.test.ts:439:15)

console.log
  Process ticket: <4> <1>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <4> <2>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <4> <3>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <4> <4>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <4> <5>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <4> <6>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <4> <7>
    at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

console.log
  Process ticket: <4> <8>
```

```
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <4> <9>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <4> <10>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

  console.log
    Process ticket: <5> <0>
      at PStateManager.reduceTickets (src/StateManager/PStateManager.ts:162:17)

 PASS  src/Tests/PLottery.test.ts (109.694 s)
  Add
     one user case (9086 ms)
     Refund check (6620 ms)
     Multiple round test (91515 ms)

Test Suites: 2 passed, 2 total
Tests:       1 skipped, 4 passed, 5 total
Snapshots:   0 total
Time:        110.055 s
Ran all test suites.
```

5.2 | Coverage

```
File                       | % Stmts | % Branch | % Funcs | % Lines | Uncovered Lines
---------------------------|---------|----------|---------|---------|----------------
All files                  |   85.53 |    37.93 |   83.18 |   85.15 |
 scripts                   |   27.86 |        0 |      20 |   27.86 |
  utils.ts                 |   27.86 |        0 |      20 |   27.86 | 21-34,38-58,
                                                                      65-82,91-105,
                                                                      112-125,129-135,
                                                                      143-148,174-177
 src                       |   98.19 |       40 |     100 |   98.08 |
  PLottery.ts              |     100 |       50 |     100 |     100 | 97
  constants.ts             |     100 |      100 |     100 |     100 |
  util.ts                  |   86.36 |    33.33 |     100 |   85.71 | 11,16-17
 src/Proofs                |   85.48 |        0 |   63.15 |   86.44 |
  DistributionProof.ts     |   86.66 |      100 |      50 |   86.66 | 74-86
  TicketReduceProof.ts     |    85.1 |        0 |   66.66 |   86.36 | 39-40,73,
                                                                      240-271
 src/Random                |   82.53 |        0 |   85.71 |   81.35 |
  RandomManager.ts         |   82.53 |        0 |   85.71 |   81.35 | 173-193,202-210
 src/StateManager          |   91.87 |    52.63 |    92.3 |   91.77 |
  BaseStateManager.ts      |   94.68 |    59.09 |      90 |   94.56 | 107,187,235,
                                                                      243,304
  PStateManager.ts         |   83.78 |       50 |      80 |   83.78 | 94-103
  RandomManagerManager.ts  |    93.1 |        0 |     100 |    93.1 | 36,56
 src/Structs               |   97.91 |        0 |     100 |   97.77 |
  CustomMerkleMap.ts       |     100 |      100 |     100 |     100 |
  Ticket.ts                |      95 |        0 |     100 |   94.73 | 28
---------------------------|---------|----------|---------|---------|----------------
```

A | Disclaimers

The audit makes no statements or warranty about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purpose

A.1 | Client Confidentiality

This document contains Client Confidential information and may not be copied without written permission.

A.2 | Proprietary Information

The content of this document should be considered proprietary information. Extropy gives permission to copy this report for the purposes of disseminating information within your organisation or any regulatory agency.

extropy