Unirep audit

Code: Unirep Github

Commit hash: `945ecf98967277a5799a23757d9d14a840360a5e`

Audit scope: Solidity contracts and Circom circuits.

Auditor: blockdev

Table of content

Low severity Issues

1. `Polysum.add()` allows any value for `vals[i]`

Context: Polysum.sol#L53

Description: The version of `Polysum.add()` which takes `vals` array as input doesn't check that the array elements are less than `SNARK_SCALAR_FIELD`. This check is done by the other `Polysum.add()` function.

However, the code is safe since this function isn't used currently.

Recommendation: Revert if `val[i] >= SNARK_SCALAR_FIELD`.

2. Use version `0x00` for signature verification

Context: VerifySignature.sol#L20-L25

Description: EIP-191 specifies that for data with intended validator, `0x19 <0x00> <intended validator address> <data to sign>` should be used for data format.

VerifySignature.sol#L20-L25 doesn't follow EIP-191 standard:

```
bytes32 messageHash = keccak256(
    abi.encodePacked(
        '\x19Ethereum Signed Message:\n32',
        keccak256(abi.encodePacked(signer, this))
    )
);
```

Recommendation: VerifySignature.sol#L20-L25 can be replaced with:

```
bytes32 messageHash = keccak256(
    abi.encodePacked(byte(0x19), byte(0), address(this), signer));
```

Further, if you want this signature to only be valid for one chain, you can include `chainid` in the message.

3. Fix `TODO` for signature verification

Context: Unirep.sol#L204

Description: Currently, an attestor be signed up for arbitrary epoch length if `attesterSignUpViaRelayer()` is called.

Recommendation: Include `epochLength` in data to be signed.

Constraint optimization

1. Bit length constraint on `nonce` can be avoided

Context: epochKeyLite.circom#L45-L49

Description:

```
component nonce_bits = Num2Bits(254);
nonce_bits.in <== nonce;
for (var x = 8; x < 254; x++) {
    nonce_bits.out[x] === 0;
}

component nonce_lt = LessThan(8);
nonce_lt.in[0] <== nonce;
nonce_lt.in[1] <== EPOCH_KEY_NONCE_PER_EPOCH;
nonce_lt.out === 1;
```

If `EPOCH_KEY_NONCE_PER_EPOCH` is able to fit in 8 bits, there is no need to explicitly constrain `nonce` to fit in 8 bits as `LessThan` circuit ensures that.

Recommendation: Explicitly assert that `EPOCH_KEY_NONCE_PER_EPOCH` fits in 8 bits, and remove the bit length check on `nonce`.

2. `epoch` is explicitly constrained to 64 bits

Context: userStateTransition.circom#L58-L66

Description: Epochs are constrained to 64 bits in the circuit. This works currently due to way epochs are calculated: `(block.timestamp - startTimestamp)/epochLength`. Assuming timestamps fit in 64 bits, epochs will fit in 64 bits too.

There is no need to explicitly contrain `epoch` to fit within 64 bits. As Unirep.sol#L404 verifies that `to_epoch` matches attestor's current epoch:

```
if (attester.currentEpoch != publicSignals[5]) revert EpochNotMatch();
```

Since `currentEpoch` fits in 64 bits, `to_epoch` cannot overflow 64 bits.

Recommendation: Consider removing the bit length check on `to_epoch` and leaving a comment mentioning that `to_epoch` is assumed to fit in 64 bits due to check in `Unirep.sol`.

Gas optimization

1. Use lateset `@zk-kit/incremental-merkle-tree.sol`

Context: Unirep.sol#L12

Description: `Unirep` is using an old version of `@zk-kit`.

Recommendation: Consider updating `@zk-kit` to a newer version.

2. `SafeMath` not required for solc>=v0.8

Context: Unirep.sol#L23

Description: Since version 8, Solidity implicitly reverts on arithmetic overflows and underflows. So there is no need to use `SafeMath`.

Recommendation: Remove the dependency on `SafeMath`.

3. Use `calldata` for all arguments

Context: Unirep.sol#L105

Description: It's better to keep the function arguments, which don't needed to modified, in `calldata` instead of `memory`. It saves gas on copying those arguments to `memory` each time the function is called.

Recommendation: Use `calldata` for every argument currently kept in `memory`.

4. Constant variables can be used in assembly directly

Context: Polysum.sol#L105-L121

Description: No need to store `SNARK_SCALAR_FIELD` in an explicit stack variable `_F`: `uint _F = SNARK_SCALAR_FIELD;`

It can directly be used in assembly blocks.

Recommendation: Apply this diff:

```
-uint _F = SNARK_SCALAR_FIELD;
...
-mstore(add(freemem, 0xA0), _F)
+mstore(add(freemem, 0xA0), SNARK_SCALAR_FIELD)
```

5. No need to save gas for execution after `staticcall` to `0x05`

Context: Polysum.sol#L123

Description: The highlighted code makes a call to `address(5)` which is a known precompile for modular exponentiation. Since it's a known precompile, you can send it all the available gas as it will always take a known amount of gas.

Recommendation: Apply this diff:

```
-sub(gas(), 2000),
+gas(),
```

6. `_updateEpochIfNeeded()` can be made `internal`

Context: Unirep.sol#L447

Description: There is another public function `updateEpochIfNeeded()` which takes a `uint160` type argument. hence `_updateEpochIfNeeded()` can be made an internal function.

Recommendation: Make `_updateEpochIfNeeded()` an `internal` function.

7. Check on `attestorId` not needed

Context: Unirep.sol#L633

Description: `verifyReputationProof()` checks that `attestorId` doesn't overflow 160 bits:

```
if (signals.attesterId >= type(uint160).max) revert AttesterInvalid();
```

However, the way `attestorId` is generated, it guarantees that it fits in 160 bits:

```
attesterId = (control >> 72) & ((1 << 160) - 1);
```

Recommendation Remove the check:

```
-if (signals.attesterId >= type(uint160).max) revert AttesterInvalid();
```

Informational issues

1. Use `address(this)` to read contract's address

Context: VerifySignature.sol#L23

Description: `address(this)` is a standard way to read contract's address. `this` refers to the current contract. However, in this case, it's being implicitly converted to the correct address.

Recommendation: Apply this diff:

```
-keccak256(abi.encodePacked(signer, this))
+keccak256(abi.encodePacked(signer, address(this)))
```

2. Remove useless files

Context: SnarkConstants.sol, Hash.sol

Description: These are not used and can be removed.

Recommendation: Remove these files.

6. `lt_comp` and `leaf_mux` signal arrays can be reduced by 1 element

Context: buildOrderedTree.circom#L82-L97

Description: Since the first element is never assigned or contrained in these arrays, the array size can be reduced by 1.

Recommendation: The code is fine as it is, however, the reader may expect that all the array elements to be constrained. To avoid this confusion, consider shortening the array lengths by 1, or commenting this issue in the code.

7. `hasher_index` can be removed and `i` can be used directly

Context: buildOrderedTree.circom#L179

Description: For elements in `hashers` array, `hasher_index` is used which is always equal to `i` which is the index variable for `values` array.

Recommendation: Consider removing `hasher_index` variable and using `i` directly in its place.

8. `state_tree_elements` can be made a 1d array

Context: userStateTransition.circom#L28

Description:

```
signal input state_tree_elements[STATE_TREE_DEPTH][1];
```

The second dimension is 1, so it can just be made a 1d array.

Recommendation: Consider making `state_tree_elements` a 1d array.