LUKSO LSPs Updates Audit Report

Apr 21, 2023

WATCHPUG

Table of Contents

Summary

This report has been prepared for LUKSO LSPs Updates Audit Report smart contract, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing 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.

Overview

Project Summary

Project Name	**LUKSO LSPs Updates Audit Report**
Codebase	**https://github.com/lukso-network/lsp-smart-contracts**
Commit	**dd235bf8fbef29cb714a923f775211221975a7fa**
Language	**Solidity**

Audit Summary

Delivery Date	**Apr 21, 2023**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**11**

[WP-L2] LSP6: `LSP6KeyManagerCore.lsp20VerifyCall()` Wrong value for the `signer` parameter in the `VerifiedCall` event

Low

Issue Description

- The current implementation: `VerifiedCall(signer: msg.sender, ...)` sets `_target` (the called contract) as the `signer` for the `VerifiedCall` event.
- The expected implementation: `VerifiedCall(signer: caller, ...)` sets the actual caller (presumably `msg.origin`) that initiates the transaction as the `signer` for the `VerifiedCall` event.

https://github.com/lukso-network/lsp-smart-contracts/blob/e1657de39dae3e31e764ef24ea3f5bd071e092f1/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L94-L116

```
94      function lsp20VerifyCall(
95          address caller,
96          uint256 msgValue,
97          bytes calldata data
98      ) external returns (bytes4) {
99          if (msg.sender != _target) revert CallerIsNotTheTarget(msg.sender);
100
101         bool isSetData = false;
102         if (bytes4(data) == SETDATA_SELECTOR || bytes4(data) ==
    SETDATA_ARRAY_SELECTOR) {
103             isSetData = true;
104         }
105
106         _nonReentrantBefore(isSetData, caller);
107
108         _verifyPermissions(caller, msgValue, data);
109         emit VerifiedCall(msg.sender, msgValue, bytes4(data));
110
111         // if it's a setData call, do not invoke the `lsp20VerifyCallResult(..)`
    function
112         return
113             isSetData
114                 ? bytes4(bytes.concat(bytes3(ILSP20.lsp20VerifyCall.selector),
    hex"00"))
```

```
115                 : bytes4(bytes.concat(bytes3(ILSP20.lsp20VerifyCall.selector),
    hex"01"));
116     }
```

See also: https://github.com/lukso-network/LIPs/blob/445b837f9fb8a93ae929f2fc4d8141a238481387/LSPs/LSP-6-KeyManager.md#L263-L271

```
263 ### Events
264
265 #### VerifiedCall
266
267 ```solidity
268 event VerifiedCall(address indexed signer, uint256 indexed value, bytes4 indexed
    selector);
269 ```
270
271 MUST be fired when the permissions of a call was successfully verified.
```

Other implementation of the same event: `VerifiedCall` :

https://github.com/lukso-network/lsp-smart-contracts/blob/e1657de39dae3e31e764ef24ea3f5bd071e092f1/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L277-L323

```
277     function _executeRelayCall(
278         bytes memory signature,
279         uint256 nonce,
280         uint256 msgValue,
281         bytes calldata payload
282     ) internal virtual returns (bytes memory) {
    @@ 283,293 @@
294
295         address signer =
    address(this).toDataWithIntendedValidator(encodedMessage).recover(
296             signature
297         );
298
```

```
@@ 299,311 @@
312
313         _verifyPermissions(signer, msgValue, payload);
314         emit VerifiedCall(signer, msgValue, bytes4(payload));
315
@@ 316,322 @@
323     }
```

Recommendation

- Clearly define all the fields of the `VerifiedCall` event in the standard.
- Update the implementation of `LSP6KeyManagerCore.lsp20VerifyCall()` according to the specifications in the standard. For example, it could be changed to `emit VerifiedCall(caller, msgValue, bytes4(data))` .

Status

✓ Fixed

[WP-L3] LSP20: Inconsistent encoding of `callResult` in `lsp20VerifyCallResult()`

Low

Issue Description

https://github.com/lukso-network/LIPs/blob/dd235bf8fbef29cb714a923f775211221975a7fa/LSPs/LSP-20-CallVerification.md#L57-L70

```
57  #### lsp20VerifyCallResult
58
59  ```solidity
60  function lsp20VerifyCallResult(bytes32 callHash, bytes memory callResult) external
    returns (bytes4 magicValue);
61  ```
62
63  MUST return the `lsp20VerifyCallResult(..)` function selector if the call to the
    function is allowed.
64
65  _Parameters:_
66
67  - `callHash`: The keccak256 of the parameters of `lsp20VerifyCall(..)` parameters
    packed-encoded (concatened).
68  - `callResult`: The result of the function being called on the contract delegating
    the verification mechanism.
69
70  _Returns:_ `magicValue` , the magic value determining if the verification
    succeeded or not.
```

The standard has not specified a clear encoding format for the `callResult` .

Additionally, in the current implementation of `LSP0ERC725AccountCore` , the encoding of `callResult` differs across various functions.

For example, the `_verifyCallResult()` function in `execute(uint256,address,uint256,bytes)` is passing the result without using `abi.encode()` . However, the `_verifyCallResult()` function in `execute(uint256[],address[],uint256[],bytes[])` is passing the `callResult` as `abi.encode(results)` .

https://github.com/lukso-network/lsp-smart-contracts/blob/e1657de39dae3e31e764ef24ea3f5bd071e092f1/contracts/LSP0ERC725Account/LSP0ERC725AccountCore.sol#L153-L171

```
153  function execute(
154      uint256 operationType,
155      address target,
156      uint256 value,
157      bytes memory data
158  ) public payable virtual override returns (bytes memory) {
159      if (msg.value != 0) emit ValueReceived(msg.sender, msg.value);
160
161      address _owner = owner();
162
163      if (msg.sender == _owner) return _execute(operationType, target, value, data);
164
165      // perform reverse verification if the caller is not the owner
166      bool verifyAfter = _verifyCall(_owner);
167      bytes memory result = _execute(operationType, target, value, data);
168      if (verifyAfter) _verifyCallResult(_owner, result);
169
170      return result;
171  }
```

https://github.com/lukso-network/lsp-smart-contracts/blob/e1657de39dae3e31e764ef24ea3f5bd071e092f1/contracts/LSP0ERC725Account/LSP0ERC725AccountCore.sol#L181-L199

```
181  function execute(
182      uint256[] memory operationsType,
183      address[] memory targets,
184      uint256[] memory values,
185      bytes[] memory datas
186  ) public payable virtual override returns (bytes[] memory) {
187      if (msg.value != 0) emit ValueReceived(msg.sender, msg.value);
188
189      address _owner = owner();
190
191      if (msg.sender == _owner) return _execute(operationsType, targets, values,
     datas);
192
```

```
193     // perform reverse verification if the caller is not the owner
194     bool verifyAfter = _verifyCall(_owner);
195     bytes[] memory results = _execute(operationsType, targets, values, datas);
196     if (verifyAfter) _verifyCallResult(_owner, abi.encode(results));
197
198     return results;
199 }
```

Recommendation

To address this issue, we recommend the following:

- Update the standard to clearly specify the format of the `callResult`. For example, it should be the entire returndata of the call and be encoded with `abi.encode()`. When there is no result for the call, use an empty byte array (`0x`) as `callResult`.
- Update the implementation to adhere to the standard's specified format for `callResult`. Modify `execute(uint256, address, uint256, bytes)` to pass `abi.encode(result)` rather than `result` as `callResult`.

Status

✓ Fixed

[WP-I5] LSP6: `LSP6KeyManagerCore.lsp20VerifyCall()` of `execute(uint256[],address[],uint256[],bytes[])` is not supported

Informational

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/e1657de39dae3e31e764ef24ea3f5bd071e092f1/contracts/LSP0ERC725Account/LSP0ERC725AccountCore.sol#L181-L199

```
181   function execute(
182           uint256[] memory operationsType,
183           address[] memory targets,
184           uint256[] memory values,
185           bytes[] memory datas
186       ) public payable virtual override returns (bytes[] memory) {
187           if (msg.value != 0) emit ValueReceived(msg.sender, msg.value);
188
189           address _owner = owner();
190
191           if (msg.sender == _owner) return _execute(operationsType, targets, values,
      datas);
192
193           // perform reverse verification if the caller is not the owner
194           bool verifyAfter = _verifyCall(_owner);
195           bytes[] memory results = _execute(operationsType, targets, values, datas);
196           if (verifyAfter) _verifyCallResult(_owner, abi.encode(results));
197
198           return results;
199       }
```

`_verifyPermissions()` does not handle `EXECUTE_ARRAY_SELECTOR`, it only handles `EXECUTE_SELECTOR`.

https://github.com/lukso-network/lsp-smart-contracts/blob/e1657de39dae3e31e764ef24ea3f5bd071e092f1/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L367-L405

```
367  function _verifyPermissions(
368      address from,
369      uint256 msgValue,
370      bytes calldata payload
371  ) internal view virtual {
372      bytes32 permissions = ERC725Y(_target).getPermissionsFor(from);
373      if (permissions == bytes32(0)) revert NoPermissionsSet(from);
374
375      bytes4 erc725Function = bytes4(payload);
376
377      // ERC725Y.setData(bytes32,bytes)
378      if (erc725Function == SETDATA_SELECTOR) {
379          if (msgValue != 0) revert CannotSendValueToSetData();
380          (bytes32 inputKey, bytes memory inputValue) = abi.decode(payload[4:],
     (bytes32, bytes));
381
382          LSP6SetDataModule._verifyCanSetData(_target, from, permissions, inputKey,
     inputValue);
383
384          // ERC725Y.setData(bytes32[],bytes[])
385      } else if (erc725Function == SETDATA_ARRAY_SELECTOR) {
386          if (msgValue != 0) revert CannotSendValueToSetData();
387          (bytes32[] memory inputKeys, bytes[] memory inputValues) = abi.decode(
388              payload[4:],
389              (bytes32[], bytes[])
390          );
391
392          LSP6SetDataModule._verifyCanSetData(_target, from, permissions, inputKeys,
     inputValues);
393
394          // ERC725X.execute(uint256,address,uint256,bytes)
395      } else if (erc725Function == EXECUTE_SELECTOR) {
396          LSP6ExecuteModule._verifyCanExecute(_target, from, permissions, payload);
397      } else if (
398          erc725Function == ILSP14Ownable2Step.transferOwnership.selector ||
399          erc725Function == ILSP14Ownable2Step.acceptOwnership.selector
400      ) {
401          LSP6OwnershipModule._verifyOwnershipPermissions(from, permissions);
402      } else {
403          revert InvalidERC725Function(erc725Function);
404      }
405  }
```

Recommendation

If it was intentionally not supported, consider documenting it. Otherwise, consider adding the support.

Status

✓ Fixed

[WP-D6] LSP20: Missing detailed documentation and example on how the `bytes32 callHash` should be used in `lsp20VerifyCallResult()`

Issue Description

https://github.com/lukso-network/LIPs/blob/dd235bf8fbef29cb714a923f775211221975a7fa/LSPs/LSP-20-CallVerification.md#L57-L70

```
57  #### lsp20VerifyCallResult
58
59  ```solidity
60  function lsp20VerifyCallResult(bytes32 callHash, bytes memory callResult) external
    returns (bytes4 magicValue);
61  ```
62
63  MUST return the `lsp20VerifyCallResult(..)` function selector if the call to the
    function is allowed.
64
65  _Parameters:_
66
67  - `callHash`: The keccak256 of the parameters of `lsp20VerifyCall(..)` parameters
    packed-encoded (concatened).
68  - `callResult`: The result of the function being called on the contract delegating
    the verification mechanism.
69
70  _Returns:_ `magicValue` , the magic value determining if the verification
    succeeded or not.
```

When it comes to the case where the call requirements should be checked before and after the execution of the function being called on another contract, `lsp20VerifyCallResult()` is expected to be called with two parameters: `(bytes32 callHash, bytes memory callResult)` .

However, there is no detailed explanation or demonstration on how `callHash` and `callResult` should be used in the current documentation and examples (`contracts/Mocks/LSP20Owners`), as well as in LSP6:

https://github.com/lukso-network/lsp-smart-contracts/blob/

43315f03e77436d4874bf735af85429ed49dedae/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L121-L128

```
121  function lsp20VerifyCallResult(
122      bytes32, /*callHash*/
123      bytes memory /*result*/
124  ) external returns (bytes4) {
125      if (msg.sender != _target) revert CallerIsNotTheTarget(msg.sender);
126      _nonReentrantAfter();
127      return ILSP20.lsp20VerifyCallResult.selector;
128  }
```

We assume that the detailed call data (`msg.sender` , `msg.value` , `msg.data`) could be stored in a storage mapping when `lsp20VerifyCall(..)` is called. This data can later be retrieved with the `callHash` that is received in `lsp20VerifyCallResult(..)` .

Recommendation

Consider adding detailed documentation and example on how the `bytes32 callHash` should be used in `lsp20VerifyCallResult()` .

[WP-D7] LSP20: The standard should provide a clearer guide for handling verification failures.

Issue Description

While the LSP-20-CallVerification specifies a `magicValue` to be returned upon successful verification, it lacks clarity regarding how to handle verification failures.

This ambiguity could lead to inconsistent implementation or misuse.

We believe it's important to establish a clear protocol for handling verification failures by specifying whether to revert or return a specific error code.

LSP-20-CallVerification.md

```
37  #### lsp20VerifyCall
38
39  ```solidity
40  function lsp20VerifyCall(address caller, uint256 value, bytes memory
    receivedCalldata) external returns (bytes4 magicValue);
41  ```
42
43  MUST return the first 3 bytes of `lsp20VerifyCall(..)` function selector if the
    call to the function is allowed, concatenated with a byte that determines if the
    `lsp20VerifyCallResult(..)` function should be called after the original function
    call.
44
45  The byte that invokes the `lsp20VerifyCallResult(..)` function is strictly `0x01`.
46
47  _Parameters:_
48
49  - `caller`: The address who called the function on the contract delegating the
    verification mechanism.
50  - `value`:  The value sent by the caller to the function called on the contract
    delegating the verification mechanism.
51  - `receivedCalldata`: The calldata sent by the caller to the contract delegating
    the verification mechanism.
52
53  _Returns:_ `magicValue` , the magic value determining if the verification
    succeeded or not.
```

[WP-D8] LSP6: Missing `}` at the end of the ”Interface Cheat Sheet” section of the LSP6 standard document

Issue Description

https://github.com/lukso-network/LIPs/blob/445b837f9fb8a93ae929f2fc4d8141a238481387/LSPs/LSP-6-KeyManager.md#L809-L840

```
809  ## Interface Cheat Sheet
810
811  ```solidity
812
813  interface ILSP6  /* is ERC165 */ {
     @@ 814,836 @@
837      function executeRelayCall(bytes[] calldata signatures, uint256[] calldata
         nonces, uint256[] calldata values, bytes[] calldata payloads) external payable
         returns (bytes[] memory);
838
839
840  ```
```

Recommendation

Consider changing to:

```
809  ## Interface Cheat Sheet
810
811  ```solidity
812
813  interface ILSP6  /* is ERC165 */ {
     @@ 814,836 @@
837      function executeRelayCall(bytes[] calldata signatures, uint256[] calldata
         nonces, uint256[] calldata values, bytes[] calldata payloads) external payable
         returns (bytes[] memory);
838
839  }
840  ```
```

[WP-D9] LSP6: The format in the comments is inconsistent with the `bytes[CompactBytesArray]` format specified by the LSP-2-ERC725YJSONSchema standard

Issue Description

According to the LSP-2-ERC725YJSONSchema standard for the `bytes[CompactBytesArray]` ValueType, each element is prefixed with **2 bytes** to specify its length:

https://github.com/lukso-network/LIPs/blob/445b837f9fb8a93ae929f2fc4d8141a238481387/LSPs/LSP-2-ERC725YJSONSchema.md#L410-L439

```
410  ## ValueType
411
412  ### bytes[CompactBytesArray]
413
414  A `bytes[CompactBytesArray]` represents an array of `bytes` values _encoded in a
     compact way_. The elements contained in the array are `bytes` values with
     different dynamic lengths.
415
416  In a compact bytes array of `bytes`, each element is prefixed with 2 bytes to
     specify its length.
417
418  For instance, `0xaabbccdd` in a `bytes[CompactBytesArray]` is encoded as
     `0x0004aabbccdd`, where:
419  - `0x0004` = `4` represents the total number of `bytes` in `0xaabbccdd`.
420  - `0xaabbccdd` is the actual value of the element.
421
422  > **Note:** the maximum length of each element is 65535, because two bytes
     (equivalent to a `uint16`) are used to store the length of each element and the
     maximum value of a `uint16` is 65535.
423
424
425  _example_
426
427  If we want to have the following bytes as elements in the compacted bytes array:
428
429  ```
430  [
431      0xaabbccdd,                    // element 1 length is 4 in hex:    0x04
```

```
    0xcafecafecafecafecafecafecafe, // element 2 length is 14 in hex:   0x0E
    0xff                           // element 3 length is 1 in hex:    0x01
]
```

The representation of these dynamic elements in a compacted bytes array would be:

`0x0004 aabbccdd 000e cafecafecafecafecafecafecafe 0001 ff` > `0x0004aabbccdd000ecafecafecafecafecafecafecafe0001ff`

However, the comments in `LSP6SetDataModule._verifyAllowedERC725YSingleKey()` still indicates that the prefix length is 1 byte, which can lead to misunderstandings.

https://github.com/lukso-network/lsp-smart-contracts/blob/e1657de39dae3e31e764ef24ea3f5bd071e092f1/contracts/LSP6KeyManager/LSP6Modules/LSP6SetDataModule.sol#L374-L476

```
    function _verifyAllowedERC725YSingleKey(
        address controllerAddress,
        bytes32 inputDataKey,
        bytes memory allowedERC725YDataKeysCompacted
    ) internal pure virtual {
        if (allowedERC725YDataKeysCompacted.length == 0)
            revert NoERC725YDataKeysAllowed(controllerAddress);

        /**
         * The pointer will always land on the length of each bytes value:
         *
         * ↓↓
         * 03 a00000
         * 05 fff83a0011
         * 20 aa0000000000000000000000000000000000000000000000000000000000cafe
         * 12 bb00000000000000000000000000beef
         * 19 cc0000000000000000000000000000000000000000deed
         * ↑↑
         *
         */
        uint256 pointer;

        // information extracted from each Allowed ERC725Y Data Key.
```

```
397         uint256 length;
398         bytes32 allowedKey;
399         bytes32 mask;
400
401         /**
402          * iterate over each data key and update the `pointer` variable with the index where to find the length of each data key.
403          *
404          * 0x 03 a00000 03 fff83a 20 aa00...00cafe
405          *    ↑↑        ↑↑        ↑↑
406          *  first  |  second  |  third
407          *  length |  length  |  length
408          */
@@ 409,475 @@
476     }
```

Recommendation

Consider changing to:

```
374     function _verifyAllowedERC725YSingleKey(
375         address controllerAddress,
376         bytes32 inputDataKey,
377         bytes memory allowedERC725YDataKeysCompacted
378     ) internal pure virtual {
379         if (allowedERC725YDataKeysCompacted.length == 0)
380             revert NoERC725YDataKeysAllowed(controllerAddress);
381
382         /**
383          * The pointer will always land on the length of each bytes value:
384          *
385          * ↓↓↓↓
386          * 0003 a00000
387          * 0005 fff83a0011
388          * 0020 aa000000000000000000000000000000000000000000000000000000000000cafe
389          * 0012 bb00000000000000000000000000beef
390          * 0019 cc0000000000000000000000000000000000deed
391          * ↑↑↑↑
392          *
393          */
```

```
394         uint256 pointer;
395
396         // information extracted from each Allowed ERC725Y Data Key.
397         uint256 length;
398         bytes32 allowedKey;
399         bytes32 mask;
400
401         /**
402          * iterate over each data key and update the `pointer` variable with the index where to find the length of each data key.
403          *
404          * 0x 0003 a00000 0003 fff83a 0020 aa00...00cafe
405          *    ↑↑↑↑        ↑↑↑↑        ↑↑↑↑
406          *   first    |  second   |  third
407          *   length   |  length   |  length
408          */
@@ 409,475 @@
476     }
```

[WP-D10] Lack of detailed documentation on the use case of `msg.sig == bytes4(0)`

Issue Description

There is currently a special handling of `msg.sig == 0x00000000` in both the standard and implementation.

However, the documentation on the intended use case for `msg.sig == 0x00000000` is not detailed enough.

https://github.com/lukso-network/lsp-smart-contracts/blob/e1657de39dae3e31e764ef24ea3f5bd071e092f1/contracts/LSP0ERC725Account/LSP0ERC725AccountCore.sol#L489-L542

```
489     /**
490      * @dev Forwards the call to an extension mapped to a function selector. If no extension address
491      * is mapped to the function selector (address(0)), then revert.
492      *
493      * The bytes4(0) msg.sig is an exception, the function won't revert if there is no extension found
494      * mapped to bytes4(0), but will execute the call to the extension in case it existed.
495      *
496      * The call to the extension is appended with bytes20 (msg.sender) and bytes32 (msg.value).
497      * Returns the return value on success and revert in case of failure.
498      *
499      * Because the function uses assembly {return()/revert()} to terminate the call, it cannot be
500      * called before other codes in fallback().
501      *
502      * Otherwise, the codes after _fallbackLSP17Extendable() may never be reached.
503      */
504     function _fallbackLSP17Extendable() internal virtual override {
505         // If there is a function selector
506         address extension = _getExtension(msg.sig);
507
508         // if no extension was found for bytes4(0) return don't revert
509         if (msg.sig == bytes4(0) && extension == address(0)) return;
```

```
510
511         // if no extension was found, revert
512         if (extension == address(0)) revert
    NoExtensionFoundForFunctionSelector(msg.sig);
513
    @@ 514,541 @@
542     }
```

https://github.com/lukso-network/LIPs/blob/dd235bf8fbef29cb714a923f775211221975a7fa/LSPs/LSP-0-ERC725Account.md#L65-L90

```
65  #### fallback
66
67  ```solidity
68  fallback() external payable;
69  ```
70
71  This function is part of the [LSP17] specification, with additional requirements
    as follows:
72
73  - MUST be payable.
74  - MUST emit a [`ValueReceived`] event if value was sent alongside some calldata.
75  - MUST return if the data sent to the contract is less than 4 bytes in length.
76  - MUST check for address of the extension under the following ERC725Y Data Key,
    and call the extension and behave according to [LSP17-ContractExtension]
    specification.
77
78  ```json
79  {
80      "name": "LSP17Extension:<bytes4>",
81      "key": "0xcee78b4094da860110960000<bytes4>",
82      "keyType": "Mapping",
83      "valueType": "address",
84      "valueContent": "Address"
85  }
86  ```
87
88  > <bytes4\> is the `functionSelector` called on the account contract. Check
    [LSP2-ERC725YJSONSchema] to learn how to encode the key.
89
```

```
90  - MUST NOT revert when there is no extension set for `0x00000000`.
```

We reckon that one possible use case of `msg.sig == bytes4(0)` is to support sending a message to the contract in the format of `0x00000000 + "some message"` .

The basis of this guess is the tests in LSP17Extendable.behaviour.ts

https://github.com/lukso-network/lsp-smart-contracts/blob/e1657de39dae3e31e764ef24ea3f5bd071e092f1/tests/LSP17ContractExtension/LSP17Extendable.behaviour.ts#L749-L792

```
749           describe("when the payload is `0x00000000` + some random data
    ('graffiti')", () => {
750             describe("with sending value", () => {
751               it("should pass and emit ValueReceived value", async () => {
752                 const amountSent = 2;
753                 const graffiti =
754                   "0x00000000" +
755                   ethers.utils
756                     .hexlify(
757                       ethers.utils.toUtf8Bytes("This is a small tip for you!")
758                     )
759                     .substring(2);
760
761                 await expect(
762                   context.accounts[0].sendTransaction({
763                     to: context.contract.address,
764                     data: graffiti,
765                     value: amountSent,
766                   })
767                 )
768                   .to.emit(context.contract, "ValueReceived")
769                   .withArgs(context.accounts[0].address, amountSent);
770               });
771             });
772             describe("without sending value", () => {
773               it("should pass", async () => {
774                 const graffiti =
775                   "0x00000000" +
776                   ethers.utils
```

```
777                     .hexlify(
778                       ethers.utils.toUtf8Bytes(
779                         "Sending a decentralized message"
780                       )
781                     )
782                     .substring(2);
783
784                   await expect(
785                     context.accounts[0].sendTransaction({
786                       to: context.contract.address,
787                       data: graffiti,
788                     })
789                   ).to.not.be.reverted;
790                 });
791               });
792             });
```

If `0x00000000` is reserved for this purpose, should setting an extension for `0x00000000` be prohibited?

Otherwise,

- The person sending the message may mistakenly execute the `0x00000000` extension
- The `0x00000000` extension could become a trap for the person sending the message

Recommendation

Consider:

- Clearly describing the purpose of `msg.sig` `0x00000000` in the standard
- Adjusting related designs accordingly (e.g. whether to allow setting an extension for `0x00000000`)

[WP-G11] Using bitwise operations instead of modulus for better compiler optimization

Gas

Issue Description

https://github.com/lukso-network/lsp-smart-contracts/blob/e1657de39dae3e31e764ef24ea3f5bd071e092f1/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L347-L360

```
@@ 347,354 @@
355     function _isValidNonce(address from, uint256 idx) internal view virtual
    returns (bool) {
356         // idx % (1 << 128) = nonce
357         // (idx >> 128) = channel
358         // equivalent to: return (nonce == _nonceStore[_from][channel]
359         return (idx % (1 << 128)) == (_nonceStore[from][idx >> 128]);
360     }
```

Recommendation

```
function _isValidNonce(address from, uint256 idx) internal view virtual returns (bool) {
    uint256 mask = ~uint128(0);
    // Alternatively:
    // uint256 mask = (1<<128)-1;
    // uint256 mask = 0xffffffffffffffffffffffffffffffff;
    return (idx & mask) == (_nonceStore[from][idx >> 128]);
}
```

Status

✓ Fixed

[WP-N12] Consistancy: explicit `false`

Issue Description

Across the codebase, there are cases where bool variables are explicitly assigned the default value `false` , and cases where they are not.

https://github.com/lukso-network/lsp-smart-contracts/blob/e1657de39dae3e31e764ef24ea3f5bd071e092f1/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L94-L116

```
94  function lsp20VerifyCall(
95      address caller,
96      uint256 msgValue,
97      bytes calldata data
98  ) external returns (bytes4) {
99      if (msg.sender != _target) revert CallerIsNotTheTarget(msg.sender);
100
101     bool isSetData = false;
102     if (bytes4(data) == SETDATA_SELECTOR || bytes4(data) ==
    SETDATA_ARRAY_SELECTOR) {
103         isSetData = true;
104     }
105

    @@ 106,115 @@

116 }
```

https://github.com/lukso-network/lsp-smart-contracts/blob/e1657de39dae3e31e764ef24ea3f5bd071e092f1/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L249-L276

```
249     function _execute(uint256 msgValue, bytes calldata payload)
250         internal
251         virtual
252         returns (bytes memory)
253     {

    @@ 254,256 @@
```

```
257
258         bool isSetData;
259         if (bytes4(payload) == SETDATA_SELECTOR || bytes4(payload) ==
    SETDATA_ARRAY_SELECTOR) {
260             isSetData = true;
261         }
262
    @@ 263,274 @@
275     }
276
```

https://github.com/lukso-network/lsp-smart-contracts/blob/e1657de39dae3e31e764ef24ea3f5bd071e092f1/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L277-L323

```
277     function _executeRelayCall(
278         bytes memory signature,
279         uint256 nonce,
280         uint256 msgValue,
281         bytes calldata payload
282     ) internal virtual returns (bytes memory) {
    @@ 283,297 @@
298
299         bool isSetData;
300         if (bytes4(payload) == SETDATA_SELECTOR || bytes4(payload) ==
    SETDATA_ARRAY_SELECTOR) {
301             isSetData = true;
302         }
303
    @@ 304,322 @@
323     }
```

Recommendation

For the sake of consistency, consider always assigning a default value:

https://github.com/lukso-network/lsp-smart-contracts/blob/e1657de39dae3e31e764ef24ea3f5bd071e092f1/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L249-L276

```
249     function _execute(uint256 msgValue, bytes calldata payload)
250         internal
251         virtual
252         returns (bytes memory)
253     {
    @@ 254,256 @@
257
258         bool isSetData = false;
    @@ 259,274 @@
275     }
276
```

https://github.com/lukso-network/lsp-smart-contracts/blob/e1657de39dae3e31e764ef24ea3f5bd071e092f1/contracts/LSP6KeyManager/LSP6KeyManagerCore.sol#L277-L323

```
277     function _executeRelayCall(
278         bytes memory signature,
279         uint256 nonce,
280         uint256 msgValue,
281         bytes calldata payload
282     ) internal virtual returns (bytes memory) {
    @@ 283,297 @@
298
299         bool isSetData = false;
    @@ 300,322 @@
323     }
```

[WP-N14] Unnecessary type casting

Issue Description

It is not necessary to cast constants of type `uint256` to `uint256` .

https://github.com/ERC725Alliance/ERC725/blob/9258ec676f1cb0154c6dd3efa56aa04f6181a8bd/implementations/contracts/ERC725XCore.sol#L70-L121

```
70  /**
71   * @dev check the `operationType` provided and perform the associated low-level opcode.
72   * see `IERC725X.execute(uint256,address,uint256,bytes)`.
73   */
74  function _execute(
75      uint256 operationType,
76      address target,
77      uint256 value,
78      bytes memory data
79  ) internal virtual returns (bytes memory) {
80      // CALL
81      if (operationType == OPERATION_0_CALL) {
    @@ 82,83 @@
84
85      // Deploy with CREATE
86      if (operationType == uint256(OPERATION_1_CREATE)) {
    @@ 87,89 @@
90
91      // Deploy with CREATE2
92      if (operationType == uint256(OPERATION_2_CREATE2)) {
    @@ 93,95 @@
96
97      // STATICCALL
98      if (operationType == uint256(OPERATION_3_STATICCALL)) {
    @@ 99,101 @@
102
103     // DELEGATECALL
```

```
@@ 104,114 @@
115     if (operationType == uint256(OPERATION_4_DELEGATECALL)) {
@@ 116,118 @@
119
120     revert ERC725X_UnknownOperationType(operationType);
121 }
```

https://github.com/ERC725Alliance/ERC725/blob/9258ec676f1cb0154c6dd3efa56aa04f6181a8bd/implementations/contracts/constants.sol#L12-L17

```
12  // ERC725X OPERATION TYPES
13  uint256 constant OPERATION_0_CALL = 0;
14  uint256 constant OPERATION_1_CREATE = 1;
15  uint256 constant OPERATION_2_CREATE2 = 2;
16  uint256 constant OPERATION_3_STATICCALL = 3;
17  uint256 constant OPERATION_4_DELEGATECALL = 4;
```

Appendix

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by WatchPug; however, WatchPug does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Smart Contract technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.