Mode wstETH mainnet

bytecode & storage contents verification

Cantina vCISO review by:
Lucas Goiriz, Junior Security Researcher

July 18, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

2 Verification Summary

Mode is the Ethereum L2 that rewards users for growing the network via new economic mechanisms. Built on the OP Stack L2, designed for growth that incentivises and directly rewards developers, users and protocols to grow Mode and the Superchain ecosystem.

On Jun 10th a verification of the on-chain bytecode and storage state corresponding to lido-l2-mode was conducted on commit hash 3c36d93c at blocks 8936437 in the Mode network, 15625677 in the Base network and 20062936 in Ethereum mainnet.

The goal of this verification is to confirm that the contract instances deployed on Mode network's Sepolia testnet (and its corresponding L1 messenger contract on the Sepolia testnet) have equivalent bytecode and storage contents to the contract instances deployed on the Base mainnet (and its corresponding L1 messenger contract on the Ethereum mainnet). The addresses of the contract instances are the following:

Contract	Mode Mainnet	Base Mainnet	Ethereum Mainnet
OptimismBridgeExecutor	0x2aCeC6...	0x0E3759...	
ERC20Bridged Proxy	0x98f96A...	0xc1CBa3...	
ERC20Bridged Impl	0xF27b1B...	0x69ce25...	
L2ERC20TokenBridge Proxy	0xb8161F...	0xac9D11...	
L2ERC20TokenBridge Impl	0x488cDB...	0x7063ef...	
L1ERC20TokenBridge Proxy (Mode)			0xD0DeA0...
L1ERC20TokenBridge Impl (Mode)			0xE6A4ED...
L1ERC20TokenBridge Proxy (Base)			0x9de443...
L1ERC20TokenBridge Impl (Base)			0x313819...

2.1 Conclusions

The bytecode verification confirmed that the logic found in the contracts destined to the wstETH deployment on Mode match the logic of the Base implementation, as existing diffs exclusively correspond to addresses used as parameters.

The storage retrieval and comparison showed that the contents of the contracts were mostly equivalent, except for differences regarding to implementation addresses and admin addresses. Furthermore, there were differences in two boolean variables in the ERC20Bridged Implementation contract, which were set to false for the Mode network and true for the Base network. This particular difference should not have any impact on the functionality of the contracts.

Regarding roles set in the contracts, all the roles are set correctly, as defined in the wstETH deployment on Mode Lido proposal.

3 Bytecode Verification Report

3.1 Perfect matches

The bytecodes were fetched from their respective networks, parsed and compared by pairs. The results were the following:

Contract	Mode Bytecode	Base Bytecode	Perfect match
OptimismBridgeExecutor	`0x608060405260...`	`0x608060405260...`	`True`
ERC20Bridged Proxy	`0x608060405260...`	`0x608060405260...`	`True`
ERC20Bridged Impl	`0x608060405234...`	`0x608060405234...`	`False`
L2ERC20TokenBridge Proxy	`0x608060405260...`	`0x608060405260...`	`True`
L2ERC20TokenBridge Impl	`0x608060405234...`	`0x608060405234...`	`False`
L1ERC20TokenBridge Proxy	`0x608060405260...`	`0x608060405260...`	`True`
L1ERC20TokenBridge Impl	`0x608060405234...`	`0x608060405234...`	`False`

The contracts that yielded a perfect match corresponded to the ossifiable proxies and the OptimismBridgeExecutor, these were not further analyzed. However, the contracts that did not yield a perfect nibble-by-nibble match were formatted into files with 60 characters per line and compared via `diff`.

3.2 Partial Matches

3.2.1 `ERC20Bridged` Implementation

The diff command was used to compare the two implementations:

```
diff mode_ERC20Bridged_Impl.bin base_ERC20Bridged_Impl.bin
```

Which yielded the following output:

```
  22c22
- 00 b8161f28a5a38ce58f155d9a96bdac0104985fac 81565b60405173ffff
  ---
+ 00 ac9d11cd4d7ef6e54f14643a393f68ca014287ab 81565b60405173ffff
  36c36
- 00000000 b8161f28a5a38ce58f155d9a96bdac0104985fac 161461046857
  ---
+ 00000000 ac9d11cd4d7ef6e54f14643a393f68ca014287ab 161461046857
  40c40
- 0000000000 b8161f28a5a38ce58f155d9a96bdac0104985fac 16146104e1
  ---
+ 0000000000 ac9d11cd4d7ef6e54f14643a393f68ca014287ab 16146104e1
```

From the above diff we can tell that the only difference between the two implementations is the bridge address, which is set as an immutable variable on construction. Indeed, one can see that Mode's implementation constructor arguments has address `0xb8161f28a5a38ce58f155d9a96bdac0104985fac` (which corresponds to the L2ERC20TokenBridge Proxy contract) as their bridge, while Base's implementation constructor arguments has address `0xac9D11cD4D7eF6e54F14643a393F68Ca014287AB` (also corresponding to the L2ERC20TokenBridge Proxy contract) as their bridge.

Hence, the bytecode logic is verified to be identical, with the only difference being the bridge address.

3.2.2 `L2ERC20TokenBridge` Implementation

The diff command was used to compare the two implementations:

```
diff mode_L2ERC20TokenBridge_Impl.bin base_L2ERC20TokenBridge_Impl.bin
```

Which yielded the following output:

```
  21c21
- 00 d0dea0a3bd8e4d55170943129c025d3fe0493f2a 81565b60405173ffff
  ---
+ 00 9de443adc5a411e83f1878ef24c3f52c61571e72 81565b60405173ffff
  26c26
- 0000000000000000 98f96a4b34d03a2e6f225b28b8f8cb1279562d81 8156
  ---
+ 0000000000000000 c1cba3fcea344f92d9239c08c0568f6f2f0ee452 8156
  53c53
- 00 98f96a4b34d03a2e6f225b28b8f8cb1279562d81 73ffffffffffffffff
  ---
+ 00 c1cba3fcea344f92d9239c08c0568f6f2f0ee452 73ffffffffffffffff
  80,81c80,81
- 5180910390fd5b867f000000000000000000000000 98f96a4b34d03a2e6f
- 225b28b8f8cb1279562d81 73ffffffffffffffffffffffffffffffffffff
  ---
+ 5180910390fd5b867f000000000000000000000000 c1cba3fcea344f92d9
+ 239c08c0568f6f2f0ee452 73ffffffffffffffffffffffffffffffffffff
  85c85
- 00000000 d0dea0a3bd8e4d55170943129c025d3fe0493f2a 3373ffffffff
  ---
+ 00000000 9de443adc5a411e83f1878ef24c3f52c61571e72 3373ffffffff
  110,111c110,111
- 90fd5b857f000000000000000000000000 98f96a4b34d03a2e6f225b28b8
- f8cb1279562d81 73ffffffffffffffffffffffffffffffffffffffff1681
  ---
+ 90fd5b857f000000000000000000000000 c1cba3fcea344f92d9239c08c0
+ 568f6f2f0ee452 73ffffffffffffffffffffffffffffffffffffffff1681
  160,161c160,161
- 04830152602482018690527f000000000000000000000000 98f96a4b34d0
- 3a2e6f225b28b8f8cb1279562d81 16906374f4f547906044016000604051
  ---
+ 04830152602482018690527f000000000000000000000000 c1cba3fcea34
+ 4f92d9239c08c0568f6f2f0ee452 16906374f4f547906044016000604051
  165,166c165,166
- 0000000000000000000000 98f96a4b34d03a2e6f225b28b8f8cb1279562d
- 81 89898988886040516024016113709796959493929190611cc2565b6040
  ---
+ 0000000000000000000000 c1cba3fcea344f92d9239c08c0568f6f2f0ee4
+ 52 89898988886040516024016113709796959493929190611cc2565b6040
  171,174c171,174
- 909152905061141a7f000000000000000000000000 d0dea0a3bd8e4d5517
- 0943129c025d3fe0493f2a 8583611676565b8673ffffffffffffffffffff
- ffffffffffffffffffff167f000000000000000000000000 98f96a4b34d0
- 3a2e6f225b28b8f8cb1279562d81 73ffffffffffffffffffffffffffffff
  ---
+ 909152905061141a7f000000000000000000000000 9de443adc5a411e83f
+ 1878ef24c3f52c61571e72 8583611676565b8673ffffffffffffffffffff
+ ffffffffffffffffffff167f000000000000000000000000 c1cba3fcea34
+ 4f92d9239c08c0568f6f2f0ee452 73ffffffffffffffffffffffffffffff
```

Which essentially are the following differences in hex nibbles:

```
  21c21 | 85c85 | 171,174c171,174
- mode: d0dea0a3bd8e4d55170943129c025d3fe0493f2a
+ base: 9de443adc5a411e83f1878ef24c3f52c61571e72

  26c26 | 53c53 | 80,81c80,81 | 110,111c110,111 | 160,161c160,161 | 165,166c165,166 | 171,174c171,174
- mode: 98f96a4b34d03a2e6f225b28b8f8cb1279562d81
+ base: c1cba3fcea344f92d9239c08c0568f6f2f0ee452
```

Due to the length of the hex nibble diffs, one can infer that these refer to different immutable addresses:

1. The first diff corresponds to the `l1TokenBridge_` constructor parameter, which corresponds to address `0xd0dea0a3bd8e4d55170943129c025d3fe0493f2a` (the L1ERC20TokenBridge) for Mode and address `0x9de443AdC5A411E83F1878Ef24C3F52C61571e72` (the L1ERC20TokenBridge again) for Base.

2. The second diff corresponds to the `l2Token_` constructor parameter, which corresponds to address `0x98f96a4b34d03a2e6f225b28b8f8cb1279562d81` (the ERC20Bridged Proxy) for Mode and address `0xc1CBa3fCea344f92D9239c08C0568f6F2F0ee452` (the ERC20Bridged Proxy again) for Base.

Note that, in contrast with past bytecode and state verifications, the `l1token_` address does not differ (i.e. diffs `37,38c37,38`, `76c76`, `164c164` and `175,176c175,176` do not exist) as both the Mode and Base implementations use the same address for the `l1Token_` constructor parameter (the corresponding contract address in Ethereum mainnet).

Hence, the bytecode logic is verified to be identical, with the only differences being the `l1TokenBridge_`, and `l2Token_`.

3.2.3 `L1ERC20TokenBridge` Implementation

The diff command was used to compare the two implementations:

```
diff mode_L1ERC20TokenBridge_Impl.bin base_L1ERC20TokenBridge_Impl.bin
```

Which yielded the following output:

```
  21,22c21,22
- 60ff166101c6565b6102857f000000000000000000000000 95bdca6c8ede
- b69c98bd5bd17660bacef1298a6f 81565b60405173ffffffffffffffffff
  ---
+ 60ff166101c6565b6102857f000000000000000000000000 866e82a600a1
+ 414e583f7f13623f1ac5d58b0afa 81565b60405173ffffffffffffffffff
  24c24
- 00000000000000000000 98f96a4b34d03a2e6f225b28b8f8cb1279562d81
  ---
+ 00000000000000000000 c1cba3fcea344f92d9239c08c0568f6f2f0ee452
  32,33c32,33
- 2303b685683908857c81565b6102857f000000000000000000000000 b816
- 1f28a5a38ce58f155d9a96bdac0104985fac 81565b6101c66103e1366004
  ---
+ 2303b685683908857c81565b6102857f000000000000000000000000 ac9d
+ 11cd4d7ef6e54f14643a393f68ca014287ab 81565b6101c66103e1366004
  63,64c63,64
- 0390fd5b857f000000000000000000000000 98f96a4b34d03a2e6f225b28
- b8f8cb1279562d81 73ffffffffffffffffffffffffffffffffffffffff16
  ---
+ 0390fd5b857f000000000000000000000000 c1cba3fcea344f92d9239c08
+ c0568f6f2f0ee452 73ffffffffffffffffffffffffffffffffffffffff16
  89,90c89,90
- 5180910390fd5b877f000000000000000000000000 98f96a4b34d03a2e6f
- 225b28b8f8cb1279562d81 73ffffffffffffffffffffffffffffffffffff
  ---
+ 5180910390fd5b877f000000000000000000000000 c1cba3fcea344f92d9
+ 239c08c0568f6f2f0ee452 73ffffffffffffffffffffffffffffffffffff
  102,103c102,103
- 405180910390fd5b867f000000000000000000000000 98f96a4b34d03a2e
- 6f225b28b8f8cb1279562d81 73ffffffffffffffffffffffffffffffffff
  ---
+ 405180910390fd5b867f000000000000000000000000 c1cba3fcea344f92
+ d9239c08c0568f6f2f0ee452 73ffffffffffffffffffffffffffffffffff
  107c107
- 0000000000 b8161f28a5a38ce58f155d9a96bdac0104985fac 3373ffffff
  ---
+ 0000000000 ac9d11cd4d7ef6e54f14643a393f68ca014287ab 3373ffffff
  109c109
- 95bdca6c8edeb69c98bd5bd17660bacef1298a6f 1614610cf2576040517f
  ---
+ 866e82a600a1414e583f7f13623f1ac5d58b0afa 1614610cf2576040517f
  112,113c112,113
- ffffffffffffffff167f000000000000000000000000 95bdca6c8edeb69c
< 98bd5bd17660bacef1298a6f 73ffffffffffffffffffffffffffffffffff
  ---
+ ffffffffffffffff167f000000000000000000000000 866e82a600a1414e
+ 583f7f13623f1ac5d58b0afa 73ffffffffffffffffffffffffffffffffff
  178c178
- 000000000000 98f96a4b34d03a2e6f225b28b8f8cb1279562d81 89898988
  ---
+ 000000000000 c1cba3fcea344f92d9239c08c0568f6f2f0ee452 89898988
  184,187c184,187
```

```
- 61159b7f000000000000000000000000 b8161f28a5a38ce58f155d9a96bd
- ac0104985fac 8583611878565b8673ffffffffffffffffffffffffffffff
- ffffffffff167f000000000000000000000000 98f96a4b34d03a2e6f225b
- 28b8f8cb1279562d81 73ffffffffffffffffffffffffffffffffffffffff
  ---
+ 61159b7f000000000000000000000000 ac9d11cd4d7ef6e54f14643a393f
+ 68ca014287ab 8583611878565b8673ffffffffffffffffffffffffffffff
+ ffffffffff167f000000000000000000000000 c1cba3fcea344f92d9239c
+ 08c0568f6f2f0ee452 73ffffffffffffffffffffffffffffffffffffffff
  212c212
- 0000000000000000 95bdca6c8edeb69c98bd5bd17660bacef1298a6f 1690
  ---
+ 0000000000000000 866e82a600a1414e583f7f13623f1ac5d58b0afa 1690
```

Which essentially are the following differences in hex nibbles:

```
  21,22c21,22 | 109c109 | 112,113c112,113 | 212c212
- mode: 95bdca6c8edeb69c98bd5bd17660bacef1298a6f
+ base: 866e82a600a1414e583f7f13623f1ac5d58b0afa

  24c24 | 63,64c63,64 | 89,90c89,90 | 102,103c102,103 | 178c178 | 184,187c184,187
- mode: 98f96a4b34d03a2e6f225b28b8f8cb1279562d81
+ base: c1cba3fcea344f92d9239c08c0568f6f2f0ee452

  32,33c32,33 | 107c107 | 184,187c184,187
- mode: b8161f28a5a38ce58f155d9a96bdac0104985fac
+ base: ac9d11cd4d7ef6e54f14643a393f68ca014287ab
```

Due to the length of the hex nibble diffs, one can infer that these refer to different immutable addresses:

1. The first diff corresponds to the `messenger` constructor parameter, which corresponds to address `0x95bdca6c8edeb69c98bd5bd17660bacef1298a6f` for Mode and address `0x866e82a600a1414e583f7f13623f1ac5d58b0afa` for Base.
2. The second diff corresponds to the `l2Token_` constructor parameter, which corresponds to address `0x98f96a4b34d03a2e6f225b28b8f8cb1279562d81` (the ERC20Bridged Proxy) for Mode and address `0xc1CBa3fCea344f92D9239c08C0568f6F2F0ee452` (the ERC20Bridged Proxy again) for Base.
3. The third diff corresponds to the `l2TokenBridge_` constructor parameter, which corresponds to address `0xb8161f28a5a38ce58f155d9a96bdac0104985fac` (the L2ERC20TokenBridge Proxy contract) for Mode and address `0xac9D11cD4D7eF6e54F14643a393F68Ca014287AB` (the L2ERC20TokenBridge Proxy contract again) for Base, as also seen in the `ERC20Bridged` case.

Hence, the bytecode logic is verified to be identical, with the only differences being the `messenger`, `l2Token_` and `l2TokenBridge_`.

Note that, in contrast with past bytecode and state verifications, the `l1token_` address does not differ (i.e. diffs `37,38c37,38`, `59c59`, `85c85`, `98c98`, `175,177c175,177` and `188,189c188,189` do not exist) as both the Mode and Base implementations use the same address for the `l1Token_` constructor parameter (the corresponding contract address in Ethereum mainnet).

4 Storage Layout Report

4.1 OptimismBridgeExecutor

The storage layouts of the OptimismBridgeExecutor contract instance in the Mode and Base networks was fetched and compared:

Slot	Description	Mode	Base	Perfect match
0x0	_delay	0x0	0x0	True
0x1	_gracePeriod	0x015180	0x015180	True
0x2	_minimumDelay	0x0	0x0	True
0x3	_maximumDelay	0x1	0x1	True
0x4	_guardian	0x0	0x0	True
0x5	_actionsSetCounter	0x0	0x0	True
0x6	_actionsSets	0x0	0x0	True
0x7	_queuedActions	0x0	0x0	True
0x8	_ethereumGovernanceExecutor	0x3e40d73eb977dc6a537af587d48316fee66e9c8c	0x3e40d73eb977dc6a537af587d48316fee66e9c8c	True

No mismatch was found between the storage layouts of the OptimismBridgeExecutor contract instance in the Mode and Base networks. Note that address 0x3e40d73eb977dc6a537af587d48316fee66e9c8c corresponds to the Lido DAO Agent address on Ethereum mainnet, which is shared between the Mode and Base networks implementations.

4.2 ERC20Bridged Proxy

The storage layouts of the ERC20Bridged proxy contract instance in the Mode and Base networks was fetched and compared. Given that it is an ERC1967 compliant proxy, there are slots devoted to proxy-specific data:

Slot	Description	Mode	Base	Perfect match
0x0	totalSupply	0x0	0x030d20bfe1457e04c030	False
0x1	balanceOf	0x0	0x0	True
0x2	allowance	0x0	0x0	True
0xad3228b676f7d3cd4284a5443f17f1962b36e491b30a40b2405849e597ba5fb5	Default admin slot	0x0	0x0	True
0x4910fdfa16fed3260ed0e7147f7cc6da11a60208b5b9406d12a635614ffd9143	Rollback slot	0x0	0x0	True
0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc	Implementation slot	0xf27b1b121e55a13047d66dc4aaa8c17ba72c762a	0x69ce2505ce515c0203160450157366f927243309	False
0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103	Admin slot	0x2acec6d8aba90685927b61968d84cfff6192b32c	0x0e37599436974a25ddeedf795c848d30af46eacf	False

Slot	Description	Mode	Base	Perfect match
0xa3f0ad74e5423aebfd80d3ef4346578335a9a72aeaee59ff6cb3582b35133d50	Beacon slot	0x0	0x0	True

The following mismatches were found:

- The totalSupply slot is set to 0x0 in the Mode network and 0x030d20bfe1457e04c030 in the Base network. This mismatch is expected, as the Base implementation is an actively used contract, while the Mode implementation has been deployed recently and has not been annouced yet.
- The implementation slot is set to 0xf27b1b121e55a13047d66dc4aaa8c17ba72c762a in the Mode network and 0x69ce2505ce515c0203160450157366f927243309 in the Base network. Again, this mismatch is expected as these proxy contracts have each their own implementation contracts. In particular, the Mode network uses the ERC20Bridged implementation contract at 0xf27b1b121e55a13047d66dc4aaa8c17ba72c762a, while the Base network uses the ERC20Bridged implementation contract at 0x69ce2505ce515c0203160450157366f927243309, which have been previously shown to be equivalent.
- The admin slot is set to 0x2acec6d8aba90685927b61968d84cfff6192b32c in the Mode network and 0x0e37599436974a25ddeedf795c848d30af46eacf in the Base network. This mismatch is expected, as the admin slot is set to the OptimismBridgeExecutor contract on the Mode network and the OptimismBridgeExecutor contract on the Base network. Hence, logic-wise, the storage layout of the ERC20Bridged proxy contract instance is the same in both networks.

4.3 ERC20Bridged Implementation

For the sake of completeness, the storage layouts of the ERC20Bridged implementation contract instance in the Mode and Base networks was fetched and compared. Given that it is an implementation contract, it is expected for it to have empty storage slots:

Slot	Description	Mode	Base	Perfect match
0x0	totalSupply	0x0	0x0	True
0x1	balanceOf	0x0	0x0	True
0x2	allowance	0x0	0x0	True

As expected, the storage slots are empty in both networks, which additionally means that the storage layout of the ERC20Bridged implementation contract instance is the same in both networks.

4.4 L2ERC20TokenBridge Proxy

The storage layouts of the L2ERC20TokenBridge proxy contract instance in the Mode and Base networks was fetched and compared. Again, given that it is an ERC1967 compliant proxy, there are slots devoted to proxy-specific data:

Slot	Description	Mode	Base	Perfect match
0x0	_roles	0x0	0x0	True
0x013e929b381f2fbbac854bd18fb8231dc73c4a2eab0d4cbb4db9436b6ff9b2ba	State slot	0x010101	0x010101	True

Slot	Description	Mode	Base	Perfect match
0xad3228b676f7d3cd4284a5443f17f1962b36e491b30a40b2405849e597ba5fb5	Default admin slot	0x0	0x0	True
0x4910fdfa16fed3260ed0e7147f7cc6da11a60208b5b9406d12a635614ffd9143	Rollback slot	0x0	0x0	True
0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc	Implementation slot	0x488cdb57e9a1006ab77730fc8b19e1bb76e1cb97	0x7063ef4f2887586e96096d3e94c9b6961c50a9a2	False
0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103	Admin slot	0x2acec6d8aba90685927b61968d84cfff6192b32c	0x0e37599436974a25ddeedf795c848d30af46eacf	False
0xa3f0ad74e5423aebfd80d3ef4346578335a9a72aeaee59ff6cb3582b35133d50	Beacon slot	0x0	0x0	True

Note that the State slot corresponds to a struct of the form:

```
struct State {
    bool isInitialized;
    bool isDepositsEnabled;
    bool isWithdrawalsEnabled;
}
```

Which are all set to `true` in both networks.

The following mismatches were found:

- The implementation slot is set to `0x488cdb57e9a1006ab77730fc8b19e1bb76e1cb97` in the Mode network and `0x7063ef4f2887586e96096d3e94c9b6961c50a9a2` in the Base network. This mismatch is expected, as these proxy contracts have each their own implementation contracts. In particular, the Mode network uses the `L2ERC20TokenBridge` implementation contract at `0x488cdb57e9a1006ab77730fc8b19e1bb76e1cb97`, while the Base network uses the `L2ERC20TokenBridge` implementation contract at `0x7063ef4f2887586e96096d3e94c9b6961c50a9a2`, which have been previously shown to be equivalent.

- The admin slot is set to `0x2acec6d8aba90685927b61968d84cfff6192b32c` in the Mode network and `0x0e37599436974a25ddeedf795c848d30af46eacf` in the Base network. This mismatch is expected, as the admin slot is set to the `OptimismBridgeExecutor` contract on the Mode network and the `OptimismBridgeExecutor` contract on the Base network, which have been previously shown to be equivalent.

Hence, logic-wise, the storage layout of the `L2ERC20TokenBridge` proxy contract instance is the same in both networks.

4.5 L2ERC20TokenBridge Implementation

For the sake of completeness, the storage layouts of the `L2ERC20TokenBridge` implementation contract instance in the Mode and Base networks was fetched and compared. Given that it is an implementation contract, it is expected for it to have empty storage slots:

Slot	Description	Mode	Base	Perfect match
0x0	_roles	0x0	0x0	True
0x013e929b381f2fbbac854bd18fb8231dc73c4a2eab0d4cbb4db9436b6ff9b2ba	State slot	0x000000	0x000001	False

The only mismatch found was the `state` slot, which is set to `0x000000` in the Mode network and `0x000001` in the Base network. The state slot contains a struct of the form:

```
struct State {
    bool isInitialized;
    bool isDepositsEnabled;
    bool isWithdrawalsEnabled;
}
```

which are all set to `false` in the Mode network. However, on the Base network, the `isInitialized` field is set to `true`. Given that it is an implementation contract, this mismatch should not have any impact on the functionality of the proxy.

4.6 L1ERC20TokenBridge Proxy

The storage layouts of the `L1ERC20TokenBridge` proxy contract instance for the Mode and Base networks was fetched and compared (note that both instances of `L1ERC20TokenBridge` exist in Ethereum mainnet but each one of them is devoted to a different network). Again, given that it is an ERC1967 compliant proxy, there are slots devoted to proxy-specific data:

Slot	Description	Mainnet (for Mode)	Mainnet (for Base)	Perfect match
0x0	_roles	0x0	0x0	True
0x013e929b381f2fbbac854bd18fb8231dc73c4a2eab0d4cbb4db9436b6ff9b2ba	State slot	0x010101	0x010101	True
0xad3228b676f7d3cd4284a5443f17f1962b36e491b30a40b2405849e597ba5fb5	Default admin slot	0x0	0x0	True
0x4910fdfa16fed3260ed0e7147f7cc6da11a60208b5b9406d12a635614ffd9143	Rollback slot	0x0	0x0	True
0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc	Implementation slot	0xe6a4ed59ec73ed78ae3a10294c99f0ee18a6bf76	0x313819736457910ac1dd21a712a37f3d7595645a	False
0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103	Admin slot	0x3e40d73eb977dc6a537af587d48316fee66e9c8c	0x3e40d73eb977dc6a537af587d48316fee66e9c8c	True

Slot	Description	Mainnet (for Mode)	Mainnet (for Base)	Perfect match
0xa3f0ad74e5423aebfd80d3ef4346578335a9a72aeaee59ff6cb3582b35133d50	Beacon slot	0x0	0x0	True

Again, the State slot corresponds to a struct of the form:

```
struct State {
    bool isInitialized;
    bool isDepositsEnabled;
    bool isWithdrawalsEnabled;
}
```

Which are all set to true in both networks.

The only mismatch found corresponds to the implementation slot, which is set to 0xe6a4ed59ec73ed78ae3a10294c99f0ee18a6bf76 for the Mode implementation and 0x313819736457910ac1dd21a712a37f3d7595645a for the Base implementation, both in the Ethereum mainnet. This mismatch is expected, as these proxy contracts have each their own implementation contracts. In particular, the Mode implementation uses the L1ERC20TokenBridge implementation contract at 0xe6a4ed59ec73ed78ae3a10294c99f0ee18a6bf76, while the Base implementation uses the L1ERC20TokenBridge implementation contract at 0x313819736457910ac1dd21a712a37f3d7595645a, which have been previously shown to be equivalent.

Hence, logic-wise, the storage layout of the L1ERC20TokenBridge proxy contract instance is the same in both networks. Note that address 0x3e40d73eb977dc6a537af587d48316fee66e9c8c corresponds to the Lido DAO Agent address on Ethereum mainnet, which is shared between the Mode and Base networks implementations.

4.7 L1ERC20TokenBridge Implementation

For the sake of completeness, the storage layouts of the L1ERC20TokenBridge implementation contract instance in the Mode and Base networks was fetched and compared. Given that it is an implementation contract, it is expected for it to have empty storage slots:

Slot	Description	Mainnet (for Mode)	Mainnet (for Base)	Perfect match
0x0	_roles	0x0	0x0	True
0x013e929b381f2fbbac854bd18fb8231dc73c4a2eab0d4cbb4db9436b6ff9b2ba	State slot	0x000000	0x000001	False

The only mismatch found was the state slot, which is set to 0x000000 for the Mode implementation and 0x000001 for the Base implementation. The state slot contains a struct of the form:

```
struct State {
    bool isInitialized;
    bool isDepositsEnabled;
    bool isWithdrawalsEnabled;
}
```

which are all set to false in the Mode implementation. However, for the Base implementation, the isInitialized field is set to true. Given that it is an implementation contract, this mismatch should not have any impact on the functionality of the proxy.

4.8 Roles and Permissions

4.9 L1ERC20TokenBridge Proxy

The `L1ERC20TokenBridge` contract has the following roles and permissions in the Mode network:

Role	Address	Description
Proxy admin	`0x3e40d73eb977dc6a537af587d48316fee66e9c8c`	Lido DAO Agent
`DEFAULT_ADMIN_ROLE`	`0x3e40d73eb977dc6a537af587d48316fee66e9c8c`	Lido DAO Agent
`WITHDRAWALS_DISABLER_ROLE`	`0x3e40d73eb977dc6a537af587d48316fee66e9c8c`	Lido DAO Agent
`WITHDRAWALS_DISABLER_ROLE`	`0x73b047fe6337183A454c5217241D780a932777bD`	MultiSig for emergency breaks
`WITHDRAWALS_ENABLER_ROLE`	`0x3e40d73eb977dc6a537af587d48316fee66e9c8c`	Lido DAO Agent
`DEPOSITS_DISABLER_ROLE`	`0x3e40d73eb977dc6a537af587d48316fee66e9c8c`	Lido DAO Agent
`DEPOSITS_DISABLER_ROLE`	`0x73b047fe6337183A454c5217241D780a932777bD`	MultiSig for emergency breaks
`DEPOSITS_ENABLER_ROLE`	`0x3e40d73eb977dc6a537af587d48316fee66e9c8c`	Lido DAO Agent

Each role was queried for their respective admin candidates (as stated in the wstETH deployment on Mode Lido proposal) and they were found to **match** the described specification.

4.10 L2ERC20TokenBridge

The `L2ERC20TokenBridge` contract has the following roles and permissions in the Mode network:

Role	Address	Description
Proxy admin	`0x2aCeC6D8ABA90685927b61968D84CfFf6192B32C`	`OptimismBridgeExecutor` contract on the Mode network
`DEFAULT_ADMIN_ROLE`	`0x2aCeC6D8ABA90685927b61968D84CfFf6192B32C`	`OptimismBridgeExecutor` contract on the Mode network
`WITHDRAWALS_DISABLER_ROLE`	`0x2aCeC6D8ABA90685927b61968D84CfFf6192B32C`	`OptimismBridgeExecutor` contract on the Mode network
`WITHDRAWALS_DISABLER_ROLE`	`0x244912352A639001ceCFa208cDaa7CB474c9eadE`	MultiSig for emergency breaks
`WITHDRAWALS_ENABLER_ROLE`	`0x2aCeC6D8ABA90685927b61968D84CfFf6192B32C`	`OptimismBridgeExecutor` contract on the Mode network
`DEPOSITS_DISABLER_ROLE`	`0x2aCeC6D8ABA90685927b61968D84CfFf6192B32C`	`OptimismBridgeExecutor` contract on the Mode network
`DEPOSITS_DISABLER_ROLE`	`0x244912352A639001ceCFa208cDaa7CB474c9eadE`	MultiSig for emergency breaks
`DEPOSITS_ENABLER_ROLE`	`0x2aCeC6D8ABA90685927b61968D84CfFf6192B32C`	`OptimismBridgeExecutor` contract on the Mode network

Each role was queried for their respective admin candidates (as stated in the wstETH deployment on Mode Lido proposal) and they were found to **match** the described specification.

5 Appendix

5.1 Bytecode fetching script

The following script was employed to fetch the on-chain bytecode of the contracts. The script uses the foundry's `anvil` to fork the network and raw `eth` calls via `python`'s `requests` module.

```python
import pandas as pd
import json
import time
import requests
import subprocess

# Constants
# L1 Proxy admin: 0x3e40D73EB977Dc6a537aF587D48316feE66E9C8c
# L2 Proxy admin: 0x2aCeC6D8ABA90685927b61968D84CfFf6192B32C
# L1 Emergency Brakes MSIG: 0x73b047fe6337183A454c5217241D780a932777bD
# L2 Emergency Brakes MSIG: 0x244912352A639001ceCFa208cDaa7CB474c9eadE

MODE_ADDRESSES = {
    "OptimismBridgeExecutor": "0x2aCeC6D8ABA90685927b61968D84CfFf6192B32C",
    "ERC20Bridged Proxy": "0x98f96A4B34D03a2E6f225B28b8f8Cb1279562d81",
    "ERC20Bridged Impl": "0xF27b1B121e55A13047d66dC4AAA8c17BA72c762A",
    "L2ERC20TokenBridge Proxy": "0xb8161F28a5a38cE58f155D9A96bDAc0104985FAc",
    "L2ERC20TokenBridge Impl": "0x488cDB57E9a1006ab77730fC8b19e1BB76e1cB97"
}
BASE_ADDRESSES = {
    "OptimismBridgeExecutor": "0x0E37599436974a25dDeEdF795C848d30Af46eaCF",
    "ERC20Bridged Proxy": "0xc1CBa3fCea344f92D9239c08C0568f6F2F0ee452",
    "ERC20Bridged Impl": "0x69ce2505ce515c0203160450157366f927243309",
    "L2ERC20TokenBridge Proxy": "0xac9D11cD4D7eF6e54F14643a393F68Ca014287AB",
    "L2ERC20TokenBridge Impl": "0x7063ef4f2887586e96096d3e94c9b6961c50a9a2"
}
MAINNET2_ADDRESSES = {
    "L1ERC20TokenBridge Proxy": "0xD0DeA0a3bd8E4D55170943129c025d3fe0493F2A",
    "L1ERC20TokenBridge Impl": "0xE6A4ED59Ec73eD78aE3A10294c99F0EE18A6bF76"
}
MAINNET_ADDRESSES = {
    "L1ERC20TokenBridge Proxy": "0x9de443AdC5A411E83F1878Ef24C3F52C61571e72",
    "L1ERC20TokenBridge Impl": "0x313819736457910aC1Dd21a712a37f3d7595645A"
}

MODE_NETWORK_RPC = ""
BASE_NETWORK_RPC = ""
MAINNET_RPC = ""
HEADERS = {"Content-Type": "application/json"}


# Helper functions
def _anvil_fork(rpc_url):
    return subprocess.Popen(
        ["anvil", "--fork-url", rpc_url],
        stdin=subprocess.PIPE,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE
    )

def _node_call(rpc_url, method, params):
    return requests.post(
        rpc_url,
        headers=HEADERS,
        json={
            "jsonrpc": "2.0",
            "method": method,
            "params": params,
            "id": 1
        }
    )

def get_bytecode(rpc_url, contract_address):
    response = _node_call(rpc_url, "eth_getCode", [contract_address, "latest"])

    if response.status_code != 200:
        raise Exception(f"Failed to get bytecode: {response.text}")

    return response.json()["result"]

def get_contract_bytecodes(rpc_url, addresses):
    bytecode = {}

    for contract, address in addresses.items():
        bytecode[contract] = get_bytecode(rpc_url, address)
        time.sleep(0.5)

    return bytecode

def get_mode_contracts_bytecode():
    bytecode = get_contract_bytecodes(MODE_NETWORK_RPC, MODE_ADDRESSES)
    bytecode.update(get_contract_bytecodes(MAINNET_RPC, MAINNET2_ADDRESSES))
    return bytecode

def get_base_contracts_bytecode():
    bytecode = get_contract_bytecodes(BASE_NETWORK_RPC, BASE_ADDRESSES)
    bytecode.update(get_contract_bytecodes(MAINNET_RPC, MAINNET_ADDRESSES))
    return bytecode

def bytecode_to_file(bytecode, filename):
    with open(f"{filename}.bin", "w") as file:
        file.write(
            "\n"
            .join(
                bytecode[i:i + 60]
                for i in range(0, len(bytecode), 60)
            )
        )
def mode_bytecode_to_file(df):
    (
        df.loc[df["perfect_match"] == False, ["Contract", "Mode Bytecode"]]
        .apply(
            lambda row: bytecode_to_file(
                row["Mode Bytecode"],
                f"mode_{row['Contract'].replace(' ', '_')}"
            ),
            axis=1
        )
    )

def base_bytecode_to_file(df):
    (
        df.loc[df["perfect_match"] == False, ["Contract", "Base Bytecode"]]
        .apply(
            lambda row: bytecode_to_file(
                row["Base Bytecode"],
                f"base_{row['Contract'].replace(' ', '_')}"
            ),
            axis=1
        )
    )

def main():

    with open("mode_contracts_bytecodes.json", "w") as file:
        json.dump(get_mode_contracts_bytecode(), file, indent=2)
        print("Bytecodes saved to mode_contracts_bytecodes.json")

    with open("base_contracts_bytecodes.json", "w") as file:
        json.dump(get_base_contracts_bytecode(), file, indent=2)
        print("Bytecodes saved to base_contracts_bytecodes.json")

    with (
        open("mode_contracts_bytecodes.json") as file1,
        open("base_contracts_bytecodes.json") as file2
    ):
        d1 = json.load(file1)

        df = pd.DataFrame({
            "Contract": list(d1.keys()),
            "Mode Bytecode": list(d1.values()),
            "Base Bytecode": list(json.load(file2).values())
        })

    df.to_parquet("bytecode_comparison.parquet", index=False)

    df = pd.read_parquet("bytecode_comparison.parquet")

    df["perfect_match"] = df["Mode Bytecode"] == df["Base Bytecode"]

    mode_bytecode_to_file(df)
    base_bytecode_to_file(df)


if __name__ == "__main__":
    main()
```

5.2 State fetching script

The following script was employed to fetch the on-chain state of the contracts. The script uses the foundry's `anvil` to fork the network and raw `eth` calls via `python`'s `requests` module.

```
import pandas as pd
import json
import time
import requests
import subprocess
```

```
# Constants
# L1 Proxy admin: 0x3e40D73EB977Dc6a537aF587D48316feE66E9C8c
# L2 Proxy admin: 0x2aCeC6D8ABA90685927b61968D84CfFf6192B32C
# L1 Emergency Brakes MSIG: 0x73b047fe6337183A454c5217241D780a932777bD
# L2 Emergency Brakes MSIG: 0x244912352A639001ceCFa208cDaa7CB474c9eadE
MODE_ADDRESSES = {
    "OptimismBridgeExecutor": "0x2aCeC6D8ABA90685927b61968D84CfFf6192B32C",
    "ERC20Bridged Proxy": "0x98f96A4B34D03a2E6f225B28b8f8Cb1279562d81",
    "ERC20Bridged Impl": "0xF27b1B121e55A13047d66dC4AAA8c17BA72c762A",
    "L2ERC20TokenBridge Proxy": "0xb8161F28a5a38cE58f155D9A96bDAc0104985FAc",
    "L2ERC20TokenBridge Impl": "0x488cDB57E9a1006ab77730fC8b19e1BB76e1cB97"
}
BASE_ADDRESSES = {
    "OptimismBridgeExecutor": "0x0E37599436974a25dDeEdF795C848d30Af46eaCF",
    "ERC20Bridged Proxy": "0xc1CBa3fCea344f92D9239c08C0568f6F2F0ee452",
    "ERC20Bridged Impl": "0x69ce2505ce515c0203160450157366f927243309",
    "L2ERC20TokenBridge Proxy": "0xac9D11cD4D7eF6e54F14643a393F68Ca014287AB",
    "L2ERC20TokenBridge Impl": "0x7063ef4f2887586e96096d3e94c9b6961c50a9a2"
}
MAINNET2_ADDRESSES = {
    "L1ERC20TokenBridge Proxy": "0xD0DeA0a3bd8E4D55170943129c025d3fe0493F2A",
    "L1ERC20TokenBridge Impl": "0xE6A4ED59Ec73eD78aE3A10294c99F0EE18A6bF76"
}
MAINNET_ADDRESSES = {
    "L1ERC20TokenBridge Proxy": "0x9de443AdC5A411E83F1878Ef24C3F52C61571e72",
    "L1ERC20TokenBridge Impl": "0x313819736457910aC1Dd21a712a37f3d7595645A"
}

MODE_NETWORK_RPC = ""
BASE_NETWORK_RPC = ""
MAINNET_RPC = ""
HEADERS = {"Content-Type": "application/json"}


# Helper functions
def _anvil_fork(rpc_url):
    return subprocess.Popen(
        ["anvil", "--fork-url", rpc_url],
        stdin=subprocess.PIPE,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE
    )

def _node_call(rpc_url, method, params):
    return requests.post(
        rpc_url,
        headers=HEADERS,
        json={
            "jsonrpc": "2.0",
            "method": method,
            "params": params,
            "id": 1
        }
    )

def get_storage_slot(rpc_url, contract_address, pos):
    response = _node_call(
        rpc_url,
        "eth_getStorageAt",
        [contract_address, pos, "latest"]
    )

    if response.status_code != 200:
        raise Exception(
            f"Failed to get storage slot {pos}: {response.text}"
        )

    r = response.json()

    try:
        return r["result"]
    except KeyError:
        raise Exception(
            f"Failed to get storage slot {pos}. Error: {r}. Addresses {contract_address}, {pos}"
        )

def get_storage_slots(rpc_url, contract_address, slots):
    _slots = {}

    for slot in slots:
        _slots[slot] = get_storage_slot(rpc_url, contract_address, slot)
        time.sleep(1)

    return _slots

def get_contract_storages(rpc_url, contracts):
    storage = {}

    for contract, content in contracts.items():
        storage[contract] = get_storage_slots(
            rpc_url,
            content["address"],
            content["slots"]
        )
        time.sleep(0.5)

    return storage

def main():

    contracts_mode = {
        "OptimismBridgeExecutor": {
            "address": MODE_ADDRESSES["OptimismBridgeExecutor"],
            "slots": list(map(lambda x: hex(x), range(9)))
        },
        "ERC20Bridged Proxy": {
            "address": MODE_ADDRESSES["ERC20Bridged Proxy"],
            "slots": [
                *list(map(lambda x: hex(x), range(3))),
                *[
                    "0xad3228b676f7d3cd4284a5443f17f1962b36e491b30a40b2405849e597ba5fb5", # Default admin slot
                    "0x4910fdfa16fed3260ed0e7147f7cc6da11a60208b5b9406d12a635614ffd9143", # Rollback slot
                    "0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc", # Implementation slot
                    "0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103", # Admin slot
                    "0xa3f0ad74e5423aebfd80d3ef4346578335a9a72aeaee59ff6cb3582b35133d50" # Beacon slot
                ]
            ]
        },
        "ERC20Bridged Impl": {
            "address": MODE_ADDRESSES["ERC20Bridged Impl"],
            "slots": list(map(lambda x: hex(x), range(3)))
        },
        "L2ERC20TokenBridge Proxy": {
            "address": MODE_ADDRESSES["L2ERC20TokenBridge Proxy"],
            "slots": [
                "0x0", # The access control roles mapping
                "0x013e929b381f2fbbac854bd18fb8231dc73c4a2eab0d4cbb4db9436b6ff9b2ba", # State slot
                "0xad3228b676f7d3cd4284a5443f17f1962b36e491b30a40b2405849e597ba5fb5", # Default admin slot
                "0x4910fdfa16fed3260ed0e7147f7cc6da11a60208b5b9406d12a635614ffd9143", # Rollback slot
                "0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc", # Implementation slot
                "0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103", # Admin slot
                "0xa3f0ad74e5423aebfd80d3ef4346578335a9a72aeaee59ff6cb3582b35133d50" # Beacon slot
            ]
        },
        "L2ERC20TokenBridge Impl": {
            "address": MODE_ADDRESSES["L2ERC20TokenBridge Impl"],
            "slots": [
                "0x0", # The access control roles mapping
                "0x013e929b381f2fbbac854bd18fb8231dc73c4a2eab0d4cbb4db9436b6ff9b2ba", # State slot
            ]
        }
    }

    contracts_base = {
        "OptimismBridgeExecutor": {
            "address": BASE_ADDRESSES["OptimismBridgeExecutor"],
            "slots": list(map(lambda x: hex(x), range(9)))
        },
        "ERC20Bridged Proxy": {
            "address": BASE_ADDRESSES["ERC20Bridged Proxy"],
            "slots": [
                *list(map(lambda x: hex(x), range(3))),
                *[
                    "0xad3228b676f7d3cd4284a5443f17f1962b36e491b30a40b2405849e597ba5fb5", # Default admin slot
                    "0x4910fdfa16fed3260ed0e7147f7cc6da11a60208b5b9406d12a635614ffd9143", # Rollback slot
                    "0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc", # Implementation slot
                    "0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103", # Admin slot
                    "0xa3f0ad74e5423aebfd80d3ef4346578335a9a72aeaee59ff6cb3582b35133d50" # Beacon slot
                ]
            ]
        },
        "ERC20Bridged Impl": {
            "address": BASE_ADDRESSES["ERC20Bridged Impl"],
            "slots": list(map(lambda x: hex(x), range(3)))
        },
        "L2ERC20TokenBridge Proxy": {
            "address": BASE_ADDRESSES["L2ERC20TokenBridge Proxy"],
            "slots": [
                "0x0", # The access control roles mapping
                "0x013e929b381f2fbbac854bd18fb8231dc73c4a2eab0d4cbb4db9436b6ff9b2ba", # State slot
                "0xad3228b676f7d3cd4284a5443f17f1962b36e491b30a40b2405849e597ba5fb5", # Default admin slot
                "0x4910fdfa16fed3260ed0e7147f7cc6da11a60208b5b9406d12a635614ffd9143", # Rollback slot
                "0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc", # Implementation slot
                "0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103", # Admin slot
                "0xa3f0ad74e5423aebfd80d3ef4346578335a9a72aeaee59ff6cb3582b35133d50" # Beacon slot
            ]
        },
        "L2ERC20TokenBridge Impl": {
            "address": BASE_ADDRESSES["L2ERC20TokenBridge Impl"],
            "slots": [
                "0x0", # The access control roles mapping
                "0x013e929b381f2fbbac854bd18fb8231dc73c4a2eab0d4cbb4db9436b6ff9b2ba", # State slot
        ]
    }
}

contracts_mainnet2 = {
    "L1ERC20TokenBridge Proxy": {
        "address": MAINNET2_ADDRESSES["L1ERC20TokenBridge Proxy"],
        "slots": [
            "0x0", # The access control roles mapping
            "0x013e929b381f2fbbac854bd18fb8231dc73c4a2eab0d4cbb4db9436b6ff9b2ba", # State slot
            "0xad3228b676f7d3cd4284a5443f17f1962b36e491b30a40b2405849e597ba5fb5", # Default admin slot
            "0x4910fdfa16fed3260ed0e7147f7cc6da11a60208b5b9406d12a635614ffd9143", # Rollback slot
            "0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc", # Implementation slot
            "0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103", # Admin slot
            "0xa3f0ad74e5423aebfd80d3ef4346578335a9a72aeaee59ff6cb3582b35133d50" # Beacon slot
        ]
    },
    "L1ERC20TokenBridge Impl": {
        "address": MAINNET2_ADDRESSES["L1ERC20TokenBridge Impl"],
        "slots": [
            "0x0", # The access control roles mapping
            "0x013e929b381f2fbbac854bd18fb8231dc73c4a2eab0d4cbb4db9436b6ff9b2ba", # State slot
        ]
    }
}

contracts_mainnet = {
    "L1ERC20TokenBridge Proxy": {
        "address": MAINNET_ADDRESSES["L1ERC20TokenBridge Proxy"],
        "slots": [
            "0x0", # The access control roles mapping
            "0x013e929b381f2fbbac854bd18fb8231dc73c4a2eab0d4cbb4db9436b6ff9b2ba", # State slot
            "0xad3228b676f7d3cd4284a5443f17f1962b36e491b30a40b2405849e597ba5fb5", # Default admin slot
            "0x4910fdfa16fed3260ed0e7147f7cc6da11a60208b5b9406d12a635614ffd9143", # Rollback slot
            "0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc", # Implementation slot
            "0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103", # Admin slot
            "0xa3f0ad74e5423aebfd80d3ef4346578335a9a72aeaee59ff6cb3582b35133d50" # Beacon slot
        ]
    },
    "L1ERC20TokenBridge Impl": {
        "address": MAINNET_ADDRESSES["L1ERC20TokenBridge Impl"],
        "slots": [
            "0x0", # The access control roles mapping
            "0x013e929b381f2fbbac854bd18fb8231dc73c4a2eab0d4cbb4db9436b6ff9b2ba", # State slot
        ]
    }
}

slot_results = {
    "mode": get_contract_storages(MODE_NETWORK_RPC, contracts_mode),
    "base": get_contract_storages(BASE_NETWORK_RPC, contracts_base),
    "mainnet2": get_contract_storages(MAINNET_RPC, contracts_mainnet2),
    "mainnet": get_contract_storages(MAINNET_RPC, contracts_mainnet)
}

with open("slot_results.json", "w") as file:
    json.dump(slot_results, file, indent=4)

with open("slot_results.json") as file:
    slot_results = json.load(file)

# Create a dictionary of dataframes for each contract, that is present on the same networks
dfs = {
    **{
        contract: pd.DataFrame({
            "Slot": list(slot_results["mode"][contract].keys()),
            "Mode": list(slot_results["mode"][contract].values()),
            "Base": list(slot_results["base"][contract].values()),
            "Mainnet2": [""]*len(slot_results["mode"][contract].values()),
            "Mainnet": [""]*len(slot_results["mode"][contract].values())
        })
        for contract in contracts_mode.keys()
    },
    **{
        contract: pd.DataFrame({
            "Slot": list(slot_results["mainnet"][contract].keys()),
            "Mode": [""]*len(slot_results["mainnet"][contract].values()),
            "Base": [""]*len(slot_results["mainnet"][contract].values()),
            "Mainnet2": list(slot_results["mainnet2"][contract].values()),
            "Mainnet": list(slot_results["mainnet"][contract].values())
        })
        for contract in contracts_mainnet.keys()
    }
}

with open("tables.md", "w") as file:

    for k,df in dfs.items():
        df["Slot"] = df["Slot"].apply(lambda x: f"`{x}`")
        if "".join(df["Mode"]) != "":
            df["Mode"] = df["Mode"].apply(lambda x: f"`{x}`")
```

```python
                df["Base"] = df["Base"].apply(lambda x: f"`{x}`")
                df["Perfect match"] = df["Mode"] == df["Base"]
            else:
                df["Mainnet2"] = df["Mainnet2"].apply(lambda x: f"`{x}`")
                df["Mainnet"] = df["Mainnet"].apply(lambda x: f"`{x}`")
                df["Perfect match"] = df["Mainnet2"] == df["Mainnet"]

            file.write(f"\n\n{df.to_markdown(index=False)}\n\n")

if __name__ == "__main__":
    main()
```