PUBLIC

Deployment Verification

of the Spark Lend Smart Contracts

April 18, 2023

Produced for

by

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help MakerDAO with the validation of the Spark Lend deployment. Our executive summary provides an overview of subjects covered in our validation of the Spark Lend deployment according to Scope to support you in forming an opinion on deviations from the Aave V3 Ethereum mainnet deployment.

Spark Lend is a fork of the Aave V3 codebase with two additional contracts not present in Aave.

The subjects covered in this report are validation of the deployed bytecode against the Aave V3 Ethereum mainnet deployment and validation of the configuration.

To summarize, our findings indicate that the codebase closely matches that of Aave V3, but not completely, and that there are some partial differences in the configurations. In Findings Overview, you can find a summary of the main differences found.

It is important to note that deployment validations are time-boxed and cannot uncover all inconsistencies. Further, it is important to note that it does not include a security assessment of the code base or the parameters selected. It is complementary but does not replace other vital measures to secure a project.

The following sections will give an overview of our methodology and our findings. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Findings Overview

This section summarizes the differences found between the Aave V3 (Ethereum mainnet) and the Spark Lend deployments. For further details see Deployment Validation.

Generally, all the protocol addresses are set as state variables or immutables mismatch so that contracts interact correctly with each other. The only address that has received privileged roles (besides the ones that are given to protocol addresses) is MakerDAO's DSPauseProxy. Some special roles are not given out which are given out by Aave (e.g. flash borrower, bridge).

However, note that there is a mismatch between Aave and Spark Lend in terms of role assignments and contract deployment. Namely,, Aave deployed the RewardsController through the pool address provider's `setAddressAsProxy` function while Spark Lend deployed it independently. However, `setAddress` was not used for Spark Lend and hence the address provider of Spark Lend does not store the address of the Rewards Controller. Further, the owner of the contract for Aave was set to the address provider while the owner in Spark Lend was set to MakerDAO's DSPauseProxy. Additionally, distinct proxy contracts are used. Aave uses the InitializableAdminUpgradeabilityProxy while Spark Lend uses the InitializableAdminUpgradeabilityProxy.

The IDs of the address providers in the corresponding address provider registry mismatch. Namely, Spark Lend uses ID 1 while Aave uses 30. Additionally, the names set in the address provider mismatch. More specifically, Spark Lend's market is named "Spark Protocol" while Aave's is "Aave Ethereum Market".

The pool contracts have differences in the reserve configurations. See the corresponding Pool section. The flashloan fees are set to 0 for Spark Lend (while Aave takes fees). While both have only one EMode category for the same assets, the categories are named differently.

In the Spark Lend tokens, the incentives controllers are set to the zero address while in Aave they are set to the address of Aave's rewards controller. Additionally, a separate DAI treasury is used on Spark Lend.

The interest rate strategies match the Aave repository and some of the deployed Aave's interest rate strategies in bytecode. See Interest Rate Strategies for differences in the parameters chosen. Additionally, there is a special strategy for DAI (DaiInterestRateStrategy) that is not present in Aave. However, it mismatches the Spark Lend repository.

The price sources match mostly with the Aave ones except for the USDC where Aave uses ChainLink and Spark Lend uses a MockAggregator returning 1. Additionally, there is a special oracle used as a price source for sDAI that matches the bytecode compiled from the repository.

The treasury manager and the treasury implementation mismatch the Aave ones. Distinct contracts are used. However, Spark Lend uses the contracts present in the repository while Aave has special contracts. Further, the proxy contracts of the treasury contracts mismatch with Aave (potentially due to differences in the compiler version). Additionally, Spark Lend has two treasury addresses, one for DAI and one for the remaining tokens.

Spark Lend's UI Pool Data Provider and UI Incentive Data Provider mismatch the Aave deployment in bytecode but match the repository.

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the deployed contracts documented inside the Spark Lend repository and is based on the documentation of Aave v3's deployed contracts on Ethereum Mainnet. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	03 April 2023	d105126e34c7453cf3820ebfbda2db057a1a580e	Spark Lend Deployment

For the solidity smart contracts, the compiler version `0.8.10` was chosen and hence was used for validating the deployments of code not present in Aave.

The values values retrieved on-chain were retrieved from Aave and Spark Lend in the time between 03.04.2023 and 16.03.2023.

The contracts in scope are explicitly listed throughout section Deployment Validation in the according subsections. Appendix: Token-Specific Addresses lists all token-specific addresses. The addresses validated in scope of the Spark Lend deployment are the ATokens, the debt tokens and interest rate strategies.

2.1.1 Excluded from scope

The security and the selection of economic parameters are out of scope. Additionally, Aave is expected to be properly set up. Further, we assume that Aave's documentation reflects the deployed contracts correctly and that MakerDAO's Chainlog returns the addresses correctly.

3 Methodology

The validation of Spark Lend's deployment included validating the bytecode and the protocol's setup.

For bytecode validation, we validated whether the bytecodes fully match, match with differences in immutables, or mismatch. To achieve this, we locally compiled the source file and compared it to both creation codes. If the creation code matched the compiler-generated creation code, a match is detected and otherwise a mismatch. To distinguish a full match and a match with differences in immutables, the source code was manually analyzed and constructor arguments were extracted and compared.

The validation of the setup included comparing the end-state of the configurations and investigating the traces of the Spark Lend contracts. Investigating the traces included manual verification of the functions called. Additionally, we compared whether the deployment methods matched how the Aave v3 contracts had been deployed. Validating the end state included extracting the configuration parameters and comparing them.

4 Limitations and use of report

Deployment assessments cannot uncover all existing mismatches; even an assessment in which no mismatches are found is not a guarantee of a correct setup. However, deployment assessments enable the discovery of unintended mismatches that were overlooked during development and areas where additional security measures are necessary. This is why we carry out a deployment validation aimed at determining all locations that mismatch the intended deployment. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many inconsistencies as possible.

The focus of our assessment was limited to the contracts defined in the engagement letter. We assessed whether the project follows a similar deployment procedure as Aave V3 on Ethereum Mainnet. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification. Further, they should be aware that no security assessment was performed.

5 Deployment Validation

5.1 Pool Address Provider Registry

The pool address provider registry (PoolAddressProviderRegistry) addresses are:

- Aave: `0xbaA999AC55EAce41CcAE355c77809e68Bb345170`
- Spark Lend: `0x03cFa0C4622FF84E50E75062683F44c9587e6Cc1`

The pool address provider registry bytecodes match.

The only constructor parameter owner sets state variables in the constructor. Initially set to `0xb90594ea5128a8178e132286dc2b7fbac7d7266c` (EOA) for Aave. Initially, it is set to `0xd1236a6a111879d9862f8374ba15344b6b233fbd` (EOA) for Spark Lend.

For Spark Lend, the EOA has registered an address provider at address `0x02C3eA4e34C0cBd694D2adFa2c690EECbC1793eE` (see Pool Address Provider) with ID `1`. Note that for Aave, the id of the market is `30`. Afterward, ownership has been transferred to MakerDAO's DSPauseProxy.

5.2 Pool Address Provider

The pool address provider (PoolAddressProvider) addresses are:

- Aave: `0x2f39d218133AFaB8F2B819B1066c7E434Ad94E9e`
- Spark Lend: `0x02C3eA4e34C0cBd694D2adFa2c690EECbC1793eE`

The pool address provider bytecodes match.

The following parameters set state variables in the constructor:

- `marketId`: Initially set to "0" for Aave, then set to "Aave Ethereum Market". Initially set to "Spark Protocol" for Spark Lend.
- `owner`: Initially set to `0xb90594ea5128a8178e132286dc2b7fbac7d7266c` (EOA) for Aave. Initially set to `0xd1236a6a111879d9862f8374ba15344b6b233fbd` (EOA) for Spark Lend.

The traces show that MakerDAO's DSPauseProxy is the final owner and final ACL admin. The remaining calls set and deploy the data provider, the configurator, the ACL manager, and the price oracle.

For Aave, the pool address provider deploys the RewardsController (`0x8164Cc65827dcFe994AB23944CBC90e0aa80bFcb`) through the function `setAddressAsProxy`. The admin of the RewardsController (being able to upgrade the implementation) is the pool address provider. The pool address provider stores the address of the RewardsController in the internal `_addresses` mapping at location `0x703c2c8634bed68d98c029c18f310e7f7ec0e5d6342c590190b3cb8b3ba54532` which is `keccak256("INCENTIVES_CONTROLLER")`. For Spark Lend, the RewardsController (`0x4370D3b6C9588E02ce9D22e684387859c7Ff5b34`) is deployed independently of the pool address provider. On Spark Lend `setAddress` is not called to set the address of the reward manager for id `0x703c2c8634bed68d98c029c18f310e7f7ec0e5d6342c590190b3cb8b3ba54532`. The pool address provider therefore cannot be queried to obtain the RewardsController address on Spark Lend.

5.3 Pool

The Pool (Pool) contracts are upgradeable contracts (InitializableImmutableAdminUpgradeabilityProxy). The addresses of the proxies are:

- Aave: `0x87870Bca3F3fD6335C3F4ce8392D69350B4fA4E2`

- Spark Lend: `0xC13e21B648A5Ee794902342038FF3aDAB66BE987`

The pool proxy is deployed as in Aave using the `PoolAddressProvider.setImpl` function, setting the implementation contract to `0x62da45546a0f87b23941ffe5ca22f9d2a8fa7df3` (see below). That additionally initializes the proxy contract. The only constructor parameter `admin` sets the `admin` contract as immutable (owner). Hence, the bytecodes match with differences in the immutables. The difference in immutables is due to the protocols setting the corresponding address provider contract.

The implementation addresses are:

- Aave: `0xfCc00A1e250644d89AF0df661bC6f04891E21585`
- Spark Lend: `0x62DA45546A0F87b23941FFE5CA22f9D2A8fa7DF3`

The bytecodes match with differences in the immutables. The only constructor parameter addressProvider sets the addressProvider contract as an immutable. They are distinct since both set the corresponding pool address provider contract. As in Aave, the implementation contract has been initialized.

The post-construction (call-only) traces are a sequence of configuration calls by the configurator (`initReserve`, `setConfiguration`, `configEModeCategory`. `updateFlashloanPremiums`) followed by regular pool operations (`supply`, `withdraw`, `borrow`, `liquidationCall`). Note that these occurred after the pool configurator had been set up. Further, note that the implementation contract has a call to its initializer as is common for Aave contracts.

The reserves (lending pools) are initialized on Spark Lend for the following assets:

- WETH (`0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2`)
- WBTC (`0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599`)
- DAI (`0x6B175474E89094C44Da98b954EedeAC495271d0F`)
- WstETH (`0x7f39C581F595B53c5cb19bD0b3f8dA6c935E2Ca0`)
- USDC (`0x7f39C581F595B53c5cb19bD0b3f8dA6c935E2Ca0`)
- sDAI (`0x83F20F44975D03b1b09e64809B757c47f942BEeA`)

The configuration parameters for the above assets are compared with the corresponding reserves on Aave (except sDAI which is unique to Spark Lend). The table in Appendix: Reserve Configuration shows the difference in parameters. Notably, USDC is not active in Spark Lend, as shown by the loan-to-value of 0 and borrowing being disabled. Other differences in parameters exist, which should be verified by the protocol administrators to mirror the desired specification.

The two global parameters of the pool `FLASHLOAN_PREMIUM_TOTAL` and `FLASHLOAN_PREMIUM_TO_PROTOCOL` differ between the Aave and Spark Lend deployment. Namely, Aave has set `FLASHLOAN_PREMIUM_TOTAL` and `FLASHLOAN_PREMIUM_TO_PROTOCOL` to 5 and 4 respectively while Spark Lend has set both to 0 (no flash loan fees)

The EMode parameters, which are configured for ETH-correlated assets (WETH and wstETH), match between Aave and Sparknet deployments with the only difference being the category name (ETH for Spark Lend and ETH correlated for Aave).

5.4 Pool Configurator

The Pool Configurator (PoolConfigurator) contracts are upgradeable contracts (InitializableImmutableAdminUpgradeabilityProxy). The addresses of the proxies are:

- Aave: `0x64b761D848206f447Fe2dd461b0c635Ec39EbB27`
- Spark Lend: `0x542DBa469bdE58FAeE189ffB60C6b49CE60E0738`

The bytecodes match with differences in the immutables. The only constructor parameter admin sets the admin contract as an immutable (owner). They are distinct since both set the corresponding address provider contract.

Note that the pool configurator proxy is deployed as in Aave using the `PoolAddressProvider. setPoolConfiguratorImpl` function, setting the implementation contract to `0xF7b656C95420194b79687fc86D965FB51DA4799F` (see below). That additionally initializes the proxy contract.

The implementation addresses are:

- Aave: `0xFDA7ffA872bDc906D43079EA134ebC9a511db0c2`
- Spark Lend: `0xF7b656C95420194b79687fc86D965FB51DA4799F`

The bytecodes match. As in Aave, the implementation contract has been initialized.

The post-construction (call-only) traces are a sequence of configuration operations (`initReserves`, `updateFlashloanPremiumTotal`, `setEModeCategory`, `setReserveBorrowing`, `configureReserveAsCollateral`, `setReserveFactor`, `setAssetEModeCategory`, `setReserveFlashLoaning`, `setLiquidationProtocolFee`, `setBorrowCap`, `setSupplyCap`) that interact with the pool. Note that these occurred after the pool had been set up and that the effects on the configuration setup of the operations are further elaborated on in Pool.

5.5 Tokens

The AToken and debt token contracts are upgradeable contracts behind the InitializableImmutableAdminUpgradeabilityProxy. See Appendix: Token-Specific Addresses for the addresses of the protocol-specific tokens (ATokens and debt tokens) supported by Spark Lend and Aave (excluding assets only present in Aave).

The deployment of tokens (ATokens, variableDebtTokens, and stableDebtTokens) is performed in a call by the function `PoolConfigurator.initReserves`. The proxy's constructor takes an admin argument, which the configurator sets to itself as the immutable admin in the proxy. Thus, the bytecodes match with a difference in the immutable variable. After the tokens are deployed, the pool configurator initializes the parameters of the tokens through the `initialize` function.

Overall deployment of tokens (and initialization of reserves) differ between Aave and Spark Lend. In Aave, every reserve is initialized by a dedicated transaction, which both deploys the tokens and sets the pool parameters for the reserve. In Spark Lend, the six reserves are initialized in a single transaction, which deploys the tokens. Individual transactions are then used to configure the reserve parameters in the Pool.

The tokens are registered as tokens for a reserve through `Pool.initReserve()`, which only the PoolConfigurator can call (called with `initReserves`). For Spark Lend, the address of the tokens configured in the pool match the upgradable tokens deployed by the PoolConfigurator.

The implementation addresses are:

- AToken:
  - Aave: `0x7EfFD7b47Bfd17e52fB7559d3f924201b9DbfF3d`
  - Spark Lend: `0x6175ddEc3B9b38c88157C10A01ed4A3fa8639cC6`
- VariableDebtTokens:
  - Aave: `0xaC725CB59D16C81061BDeA61041a8A5e73DA9EC6`
  - Spark Lend: `0x86C71796CcDB31c3997F8Ec5C2E3dB3e9e40b985`
- StableDebtTokens:

- Aave: `0x15C5620dfFaC7c7366EED66C20Ad222DDbB1eD57`
- Spark Lend: `0x026a5B6114431d8F3eF2fA0E1B2EDdDccA9c540E`

The bytecodes match with differences in the immutables. The constructor takes the corresponding pool's address as an argument which is set as an immutable in all implementations.

The token parameters are correctly configured. As a difference between Aave and Spark Lend, in the Spark Lend tokens the incentives controllers are set to the zero address, in Aave it is set to the address of the RewardsController. Additionally, a separate DAI treasury is used on Spark Lend, see Treasury & DAI Treasury. The name of the tokens differ between Aave and Spark Lend to show that they belong to Spark Lend or Aave.

5.6 Interest Rate Strategies

Please see Appendix: Token-Specific Addresses for the addresses of the interest rate strategies for each token supported by Spark Lend and Aave (excluding assets only present in Aave).

All Aave Interest Rate Strategies are DefaultReserveInterestRateStrategy. Similarly, this holds for all Spark Lend Interest Rate Strategies except for the DAI interest rate strategy which is a custom interest rate strategy DaiInterestRateStrategy for integration with D3Ms.

All Spark Lend interest rate strategies match the bytecode derived from the repository at the commit. Similarly, this holds for the Aave ones with the exceptions of WETH and wstETH ones. A potential reason for the mismatch in the Aave code is that a different compiler version is used (`0.8.17` instead of `0.8.10`).

Hence, all the non-exceptional interest rate strategies match with differences in immutables. Namely, all parameters are set as immutables. The provider `provider` corresponds to the Aave's and Spark Lend's pool address provider contracts, see Pool Address Provider.

See Appendix: Default Interest Rate Strategy Configuration for details about the economic parameters of the interest rate strategies for Aave and Spark Lend. Most notably, parameters related to stable borrowing are set to 0 for Spark Lend which is different from Aave's setup. Another example of a difference is that the reserve rate of Spark Lend for DAI is set to 100%.

The DaiInterestRateStrategy in the repository for DAI mismatches the bytecode compiled locally, due to some differences in the source file. The constructor arguments set the immutable variables of the interest rate strategy. The following parameters are set in the constructor:

- `_vat`: MakerDAO's Vat address
- `_pot`: MakerDAO's Pot address
- `_ilk`: `bytes32` representation of `DIRECT-SPARK-DAI` (hence, right padded with zeros)
- `_baseRateConversion`: `1111111111111111111111111111`
- `_borrowSpread`: `0`
- `_supplySpread`: `0`
- `_maxRate`: `750000000000000000000000000`
- `_performanceBonus`: `1000000000000000000000000000`

5.7 ACLManager

The ACL Manager contract (ACLManager) addresses are:

- Aave: `0xc2aaCf6553D20d1e9d78E365AAba8032af9c85b0`
- Spark Lend: `0xdA135Cd78A086025BcdC87B038a1C462032b510C`

The bytecodes match with differences in immutables. The only constructor parameter `provider` sets the `provider` contract as an immutable, see Pool Address Provider. They are distinct since both set the corresponding pool address provider contract.

Note that the constructor sets up the default role admin role as the ACL admin set in the pool address provider.

The traces of the contract indicate that the initial role admin set is an EOA that then

1. gives pool, emergency, and role admin rights to MakerDAO's DSPauseProxy,
2. and then revokes all of its permissions.

Ultimately, the result is that MakerDAO is given all permissions.

Note that Aave has given roles to other parties and has also assigned roles that Spark Lend has not assigned yet (e.g. flash borrower, bridge).

5.8 Oracle

The oracle contract (AaveOracle) addresses are:

- Aave: `0x54586bE62E3c3580375aE3723C145253060Ca0C2`
- Spark Lend: `0x8105f69D9C41644c6A0803fDA7D03Aa70996cFD9`

The bytecodes match with differences in immutables. The following constructor parameters set immutables:

- `provider`: Set to the corresponding pool address provider, see Pool Address Provider.
- `baseCurrency`: Set to `0x0000000000000000000000000000000000000000` for both.
- `baseCurrencyUnit`: Set to `100000000` for both.

The `fallbackOracle` constructor parameter sets the fallback oracle. Both set this initially to `0x0000000000000000000000000000000000000000`. Note that the fallback oracle for both has not been set afterward.

The assets and sources array constructor parameters set asset-to-source mappings. The addresses are presented in the table in Appendix: Token-Specific Addresses. For the assets present in both Aave and Spark Lend, the price feeds used are the same except for USDC price source. Spark Lend uses a MockAggregator returning 1 while Aave uses a Chainlink price feed.

The oracle for sDAI is a contract that has no equivalent in Aave. However, the bytecode matches the repository. The constructor arguments set the DAI price feed to the price feed to which DAI is mapped (`0xAed0c38402a5d19df6E4c03F4E2DceD6e29c1ee9`) and to MakerDAO's Pot contract.

No state-changing traces can be found after the deployment of Spark Lend. Hence, the setup on construction remains.

5.9 Incentives

5.9.1 Emission Manager

The Emission Manager (EmissionManager) addresses are:

- Aave: `0x223d844fc4B006D67c0cDbd39371A9F73f69d974`
- Spark Lend: `0xf09e48dd4CA8e76F63a57ADd428bB06fee7932a4`

The bytecodes match.

The only constructor parameter `owner` sets the owner:

- Aave: Sets it initially to: `0xb90594ea5128a8178e132286dc2b7fbac7d7266c` (EOA)

- Spark Lend: Sets it initially to: `0xd1236a6a111879d9862f8374ba15344b6b233fbd` (EOA)

The traces show that the rewards controller is set to the corresponding rewards controller address, see Rewards Controller and that ownership is transferred to MakerDAO's DSPauseProxy.

5.9.2 Rewards Controller

The Rewards Controllers (RewardsController) are upgradeable contracts. The addresses of the proxies are:

- Aave: `0x8164Cc65827dcFe994AB23944CBC90e0aa80bFcb`
- Spark Lend: `0x4370D3b6C9588E02ce9D22e684387859c7Ff5b34`

The bytecodes mismatch. Two distinct proxy files are used:

- Aave: InitializableImmutableAdminUpgradeabilityProxy
- Spark Lend: InitializableAdminUpgradeabilityProxy

The main difference is that the Spark Lend proxy contract does not have an immutable owner.

On initialization, the owner of the Spark Lend proxy is set to MakerDAO's DSPauseProxy. The owner of the Aave proxy is the PoolAddressProvider and is set on construction. As mentioned in the Pool Address Provider section, in Aave the address of the RewardsController is registered in the pool address provider upon deployment, the same is not performed in Spark Lend. Additionally, the initialization sets the implementation contract (see below).

The implementation addresses are:

- Aave: `0xE7B67F44eA304DD7f6d215b13686637ff64CD2B2`
- Spark Lend: `0x0ee554F6A1f7a4Cb4f82D4C124DdC2AD3E37fde1`

The bytecodes match with differences in the immutables. The only constructor parameter `emissionManager` sets the emission manager contract as an immutable. They are distinct since both set the corresponding emission manager contract, see Emission Manager.

5.10 Treasury Manager

The Treasury Manager addresses are:

- Aave: `0x3d569673dAa0575c936c7c67c4E6AedA69CC630C`
- Spark Lend: `0x92eF091C5a1E01b3CE1ba0D0150C84412d818F7a`

The bytecodes mismatch since Spark Lend uses a treasury contract different from Aave V3. For Spark Lend, the CollectorController contract from the repository is used. The bytecode matches the contract in the repository. The owner is a variable that is initially set to MakerDAO's DSPauseProxy.

No further traces can be found for the treasury manager. Given the state-changing functions, that is expected.

5.11 Treasury & DAI Treasury

The Treasury contracts are upgradeable. The addresses of the proxies are:

- Aave: `0x464C71f6c2F760DdA6093dCB91C24c39e5d6e18c`
- Spark Lend: `0xb137E7d16564c81ae2b0C8ee6B55De81dd46ECe5`

A difference is that Spark Lend has two treasury addresses. Additionally, there is the DAI treasury at address `0x856900aa78e856a5df1a2665eE3a66b2487cD68f`.

The bytecode of the Spark Lend contracts matches. They do not match the bytecode of the Aave contract. However, the names of the contracts match. Note that differences could be in the compiler version. However, the Spark Lend version matches the Aave V3 repository.

Both proxies are only initialized so that the implementation is set (see below) and so that the treasury manager (see Treasury Manager) is set. No other calls are made to the contract. Given the state-changing functions, that is expected.

The implementation addresses are:

- Aave: `0x1aa435ed226014407Fa6b889e9d06c02B1a12AF3`
- Spark Lend: `0xF1E57711Eb5F897b415de1aEFCB64d9BAe58D312`

Note that the Spark Lend treasuries share the same implementation contract. The bytecodes mismatch the Aave one. Note that distinct contracts are used. However, the Spark Lend implementation contract's bytecode matches the Aave v3 repository (Collector).

Similar to the common practice in Aave, the implementation contract is initialized.

5.12 Protocol Data Provider

The Protocol Pool Data Provider (AaveProtocolDataProvider) addresses are:

- Aave: `0x7B4EB56E7CD4b454BA8ff71E4518426369a138a3`
- Spark Lend: `0xFc21d6d146E6086B8359705C8b28512a983db0cb`

The bytecodes match with differences in the immutables. The only constructor parameter addressProvider sets the addressProvider contract as an immutable. They are distinct since both set the corresponding address provider contract. Given that there are no state-changing functions, the setup of the contract does not require any additional checks.

5.13 Wallet Balance Provider

The Wallet Balance Provider (WalletBalanceProvider) addresses are:

- Aave: `0xC7be5307ba715ce89b152f3Df0658295b3dbA8E2`
- Spark Lend: `0xd2AeF86F51F92E8e49F42454c287AE4879D1BeDc`

The bytecodes match. The constructor has no arguments. Given that there are no state-changing functions, the setup of the contract does not require any additional checks.

5.14 UI Pool Data Provider

The UI Pool Data Provider (UiPoolDataProviderV3) addresses are:

- Aave: `0x91c0eA31b49B69Ea18607702c5d9aC360bf3dE7d`
- Spark Lend: `0xF028c2F4b19898718fD0F77b9b881CbfdAa5e8Bb`

The bytecodes mismatch. Spark Lend has deployed the contract at the commit of the repository while Aave has not.

The constructor arguments are equal and are set for both Aave and Spark Lend to `0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419` (see ETH / USD | Chainlink). Given that there are no state-changing functions, the setup of the contract does not require any additional checks.

5.15 UI Incentive Data Provider

The UI Incentive Data Provider (UiIncentiveDataProviderV3) addresses are:

- Aave: `0x162A7AC02f547ad796CA549f757e2b8d1D9b10a6`

- Spark Lend: `0xA7F8A757C4f7696c015B595F51B2901AC0121B18`

The bytecodes mismatch. Spark Lend has deployed the contract at the commit of the repository while Aave has not. The constructor has no arguments. Given that there are no state-changing functions, the setup of the contract does not require any additional checks.

5.16 WETH Gateway

The WETH Gateway (WrappedTokenGatewayV3) addresses are:

- Aave: `0xD322A49006FC828F9B5B37Ab215F99B4E5caB19C`
- Spark Lend: `0xBD7D6a9ad7865463DE44B05F04559f65e3B11704`

The bytecodes match with differences in the immutables. The following constructor parameters set immutables:

- `weth`: Equal. Set to the WETH9 address (`0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2`).
- `pool`: Distinct. Set to the corresponding pool address.

The following parameters set state variables in the constructor:

- `owner`: The initial owner for Spark Lend is the DSPauseProxy (`0xbe8e3e3618f7474f8cb1d074a26affef007e98fb`). The initial owner for Aave is an EOA (`0xb90594ea5128a8178e132286dc2b7fbac7d7266c`).

Given that the WETH Gateway contract's functions integrate with the core or can only be called by the owner, the setup of the contract does not require any additional checks.

6 Appendix

6.1 Appendix: Token-Specific Addresses

token	domain	aTokenAddress	stableDebtTokenAddress
DAI	AAVE	0x018008bfb33d285247A21d44E50697654f754e63	0x413AdaC9E2Ef8683ADf5DDAEce8f19613d60D1bb
	SPARK	0x4DEDf26112B3Ec8eC46e7E31EA5e123490B05B8B	0xfe2B7a7F4cC0Fb76f7Fc1C6518D586F1e4559176
USDC	AAVE	0x98C23E9d8f34FEFb1B7BD6a91B7FF122F4e16F5c	0xB0fe3D292f4bd50De902Ba5bDF120Ad66E9d7a39
	SPARK	0x377C3bd93f2a2984E1E7bE6A5C22c525eD4A4815	0x887Ac022983Ff083AEb623923789052A955C6798
WBTC	AAVE	0x5Ee5bf7ae06D1Be5997A1A72006FE6C607eC6DE8	0xA1773F1ccF6DB192Ad8FE826D15fe1d328B03284
	SPARK	0x4197ba364AE6698015AE5c1468f54087602715b2	0x4b29e6cBeE62935CfC92efcB3839eD2c2F35C1d9
WETH	AAVE	0x4d5F47FA6A74757f35C14fD3a6Ef8E3C9BC514E8	0x102633152313C81cD80419b6EcF66d14Ad68949A
	SPARK	0x59cD1C87501baa753d0B5B5Ab5D8416A45cD71DB	0x3c6b93D38ffA15ea995D1BC950d5D0Fa6b22bD05
sDAI	AAVE		
	SPARK	0x78f897F0fE2d3B5690EbAe7f19862DEacedF10a7	0xEc6C6aBEd4DC03299EFf82Ac8A0A83643d3cB335
wstETH	AAVE	0x0B925eD163218f6662a35e0f0371Ac234f9E9371	0x39739943199c0fBFe9E5f1B5B160cd73a64CB85D
	SPARK	0x12B54025C112Aa61fAce2CDB7118740875A566E9	0x9832D969a0c8662D98fFf334A4ba7FeE62b109C2

		variableDebtTokenAddress	interestRateStrategyAddress
DAI	**AAVE**	0xcF8d0c70c850859266f5C338b38F9D663181C314	0x694d4cFdaeE639239df949b6E24Ff8576A00d1f2
	SPARK	0xf705d2B7e92B3F38e6ae7afaDAA2fEE110fE5914	0xfD0cc3F39d48a2393443e18E7d3758FC4c3c5c37
USDC	**AAVE**	0x72E95b8931767C79bA4EeE721354d6E99a61D004	0xD6293edBB2E5E0687a79F01BEcd51A778d59D1c5
	SPARK	0x7B70D04099CB9cfb1Db7B6820baDAfB4C5C70A67	0x4d988568b5f0462B08d1F40bA1F5f17ad2D24F76
WBTC	**AAVE**	0x40aAbEf1aa8f0eEc637E0E7d92fbfFB2F26A8b7B	0x24701A6368Ff6D2874d6b8cDadd461552B8A5283
	SPARK	0xf6fEe3A8aC8040C3d6d81d9A4a168516Ec9B51D2	0xf2812d7a07573322D4Db3C31239C837081D8294E
WETH	**AAVE**	0xeA51d7853EEFb32b6ee06b1C12E6dcCA88Be0fFE	0x53F57eAAD604307889D87b747Fc67ea9DE430B01
	SPARK	0x2e7576042566f8D6990e07A1B61Ad1efd86Ae70d	0x764b4AB9bCA18eB633d92368F725765Ebb8f047C
sDAI	**AAVE**		
	SPARK	0xaBc57081C04D921388240393ec4088Aa47c6832B	0xeC4cf692c18E62159a39704Aa1Db82ca2306fF90
wstETH	**AAVE**	0xC96113eED8cAB59cD8A66813bCB0cEb29F06D2e4	0x7b8Fa4540246554e77FCFf140f9114de00F8bB8D
	SPARK	0xd5c3E3B566a42A6110513Ac7670C1a86D76E13E6	0x0D56700c90a690D8795D6C148aCD94b12932f4E3

6.2 Appendix: Reserve Configuration

token	DAI		USDC		WBTC		WETH		sDAI		wstETH	
domain	AAVE	SPARK	AAVE	SPARK	AAVE	SPARK	AAVE	SPARK	AAVE	SPARK	AAVE	SPARK
id	4	0	3	2	2	5	0	3		1	1	4
unbacked	0	0	0	0	0	0	0	0		0	0	0
ltv	6400	7400	7400	0	7000	7000	8000	8000		7400	6850	6850
liq threshold	7700	7600	7600	0	7500	7500	8250	8250		7600	7950	7950
liq bonus	10400	10450	10450	0	10625	10625	10500	10500		10450	10700	10700
decimals	18	18	6	6	8	8	18	18		18	18	18
is active	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE		TRUE	TRUE	TRUE
is frozen	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE		FALSE	FALSE	FALSE
borrowing is enabled	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE		FALSE	TRUE	TRUE
stable rate borrowing is enabled	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE		FALSE	FALSE	FALSE
asset is paused	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE		FALSE	FALSE	FALSE
borrowing isolation mode enabled	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE		FALSE	FALSE	FALSE
reserved bits	2	2	2	2	2	2	2	2		2	2	2
reserve factor	1000	10000	1000	1000	2000	2000	1500	1500		1000	1500	1500
borrow cap	271000000	0	1.58E+09	0	28000	500	1400000	1400000		0	6000	3000
supply cap	338000000	0	1.76E+09	0	43000	1000	1800000	0		0	200000	200000
liquidation protocol fee	2000	2000	2000	0	1000	1000	1000	1000		2000	1000	1000
emode category	0	0	0	0	0	0	1	1		0	1	1
unbacked mint cap	0	0	0	0	0	0	0	0		0	0	0
debt ceiling isolation mode	0	0	0	0	0	0	0	0		0	0	0
unused	0	0	0	0	0	0	0	0		0	0	0

6.3 Appendix: Default Interest Rate Strategy Configuration

token	DAI	USDC		WBTC	
domain	AAVE	AAVE	SPARK	AAVE	SPARK
OPTIMAL_USAGE_RATIO	8E+26	9E+26	1E+27	4.5E+26	6.5E+26
getBaseVariableBorrowRate	0	0	1E+25	0	0
getVariableRateSlope1	4E+25	4E+25	0	7E+25	8E+25
getVariableRateSlope2	7.5E+26	6E+26	0	3E+27	3E+27
IAL_STABLE_TO_TOTAL_DEBT_	2E+26	2E+26	0	2E+26	0
getStableRateSlope1	5E+24	5E+24	0	7E+25	0
getStableRateSlope2	7.5E+26	6E+26	0	3E+27	0
getBaseStableBorrowRate	5E+25	5E+25	0	9E+25	8E+25
getStableRateExcessOffset	8E+25	8E+25	0	5E+25	0

token	WETH		sDAI	wstETH	
domain	AAVE	SPARK	SPARK	AAVE	SPARK
OPTIMAL_USAGE_RATIO	8E+26	8E+26	1E+27	4.5E+26	4.5E+26
getBaseVariableBorrowRate	1E+25	1E+25	1E+25	2.5E+24	2.5E+24
getVariableRateSlope1	3.8E+25	3.8E+25	0	4.5E+25	4.5E+25
getVariableRateSlope2	8E+26	8E+26	0	8E+26	8E+26
IAL_STABLE_TO_TOTAL_DEBT_	2E+26	0	0	2E+26	0
getStableRateSlope1	4E+25	0	0	4E+25	0
getStableRateSlope2	8E+26	0	0	8E+26	0
getBaseStableBorrowRate	6.8E+25	3.8E+25	0	7.5E+25	4.5E+25
getStableRateExcessOffset	5E+25	0	0	5E+25	0