CHAINSECURITY

PUBLIC

Code Assessment
of the Spark ALM Controller v1.11

Smart Contracts

October 21, 2026

Produced for

Spark



Contents

1	Executive Summary	3
2	Assessment Overview	4
3	System Overview	5
4	Trust Model	7
5	System Considerations	8
6	Terminology	10
7	Findings	11
8	Limitations and use of report	13

1 Executive Summary

Dear Spark Team,

Thank you for trusting ChainSecurity with the review of the CCTP V2 migration of the Spark ALM Controller v1.11. As a globally trusted security partner for tier-one DeFi protocols, ChainSecurity provides specialized, independent technical analysis of complex smart contract systems and their integrations with third-party protocols. We greatly appreciate the opportunity to continue our collaboration with Spark on the Spark Liquidity Layer.

Spark implements the Spark ALM Controller, a set of on-chain components of the Spark Liquidity Layer designed to manage and control the flow of liquidity on Ethereum mainnet and L2s, and between them. This version migrates the mainnet USDC bridging path from Circle's CCTP V1 to CCTP V2.

We have conducted a thorough review of the scope. The most important area covered in our review is the **correctness of the CCTP V2 integration**, as CCTP V2 introduces new parameters such as a fee cap and a finality threshold.

We provide this audit report to assist in forming an opinion on the risk profile of the Spark ALM Controller v1.11. All issues we reported are of informational severity and have been corrected in `Version 2`. In summary, we find that the codebase provides a good level of security, if operated under the assumptions documented in **System Overview**, **Trust Model** and **System Considerations**.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,
ChainSecurity

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

This assessment is a **differential review** performed on the source code files inside the Spark ALM Controller v1.11 repository, covering only the changes introduced in `v1.11.0-beta.0` relative to the `v1.10.0` release (`984ec546fb2c98ed729ae91d2d73e97dedbf111f`), as well as the subsequent changes in `v1.11.0`.

Note: This review is conducted under the assumption that the codebase at `v1.10` is correct and secure. Only the modifications, additions and deletions introduced by the CCTP V2 migration are considered in scope. Findings from previous audits and the security properties established in `v1.10` are assumed to hold.

The table below indicates the code versions relevant to this report and when they were received.

Version	Date	Commit Hash	Note
1	6 October 2026	899a8fd524c342d99df5f112b16235b7e2d45b0e	v1.11.0-beta.0
2	20 October 2026	9c7dffe3c1171428684bc63cbbfb17e23b8a7bf3	v1.11.0

For the Solidity smart contracts, the compiler version `0.8.25` was chosen with the EVM version set to `cancun`.

Only the changes related to the CCTP V2 migration are considered in scope for the following files:

```
src/
├── MainnetController.sol
└── libraries/
    └── CCTPLib.sol
```

In **Version 2**, `CCTPLib.sol` has been renamed to `CCTPV2Lib.sol`.

2.1.1 Excluded from scope

Any other files are out of scope. The following are explicitly out of scope:

- The entire codebase as reviewed in previous versions (`v1.0.0` through `v1.10`). These are assumed correct and secure.
- `ForeignController.sol` and its CCTP integration, which deliberately remains on CCTP V1, see ForeignController Remains on CCTP V1.
- All 3rd-party protocols that the `ALMProxy` integrates with. In particular, Circle's CCTP V2 contracts (`TokenMessengerV2`, `TokenMinterV2`, `MessageTransmitterV2`) and the off-chain attestation service are assumed to work honestly and correctly as documented.
- Dependencies and libraries, e.g. OpenZeppelin.

3 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview. Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Spark offers the Spark ALM Controller, a set of on-chain components of the Spark Liquidity Layer designed to manage and control the flow of liquidity on Ethereum mainnet and L2s, and between them.

On each chain, the following contracts are deployed:

1. `ALMProxy`: Entity that holds funds and interacts with external contracts. Thus, it holds the required privileges to interact with other contracts.
2. Controllers: Dictate which operations an `ALMProxy` shall perform. Note that multiple controllers could point to the same `ALMProxy`.
3. `RateLimits`: Computes limits for liquidity flows.

All of the contracts inherit the standard `AccessControl` and grant the `DEFAULT_ADMIN_ROLE` to an `admin` which can configure other roles.

This version migrates the **mainnet** USDC bridging path from Circle's CCTP V1 to CCTP V2. For full reviews of previous releases, please refer to the Spark ALM Controller Audits.

3.1 Migration to CCTP V2

Changed in this version; only in the **mainnet** controller.

Compared to CCTP V1, CCTP V2 additionally takes a `destinationCaller`, a `maxFee` and a `minFinalityThreshold` as input and returns nothing.

`CCTPLib` has been rewritten accordingly. Two of the three new parameters are fixed as constants:

- `DESTINATION_CALLER = 0`, meaning any address may relay the message on the destination chain.
- `MAX_FINALITY_THRESHOLD = 2_000`, which selects a standard (finalized) transfer rather than a fast transfer. Note that the corresponding CCTP V2 parameter is named `minFinalityThreshold` and acts as a lower bound: a lower value opts into a fast transfer, which is attested earlier and carries a fee.

Because CCTP V2 no longer returns a nonce, the `CCTPTransferInitiated` event has lost its `nonce` field.

3.1.1 Fee cap

CCTP V2 charges a fee for fast transfers, deducted from the amount minted on the destination chain.

A new storage variable `cctpMaxFeeRate` has been introduced on `MainnetController`, expressed in basis points and settable by the `DEFAULT_ADMIN_ROLE` through `setCCTPMaxFeeRate()`. It is bounded by `10_000`.

The same rate applies to every destination domain. Since the finality threshold is fixed at `2_000`, standard transfers currently carry no fee and the default value of `0` is the operationally correct setting. The parameter exists for the case where Circle introduces a fee on standard transfers.

3.1.2 Other refactoring

- The `LIMIT_USDC_TO_CCTP` and `LIMIT_USDC_TO_DOMAIN` rate limit keys are now defined in `CCTPLib` as public constants; `MainnetController` still exposes them through its public getters. The derived key values are unchanged, and both limits are still decreased by the gross transfer amount before any validation.

- `CCTPLib.sol` now declares its own interfaces for CCTP V2 (`ICCTPLike` , `ICCTPTokenMinterLike` , `IERC20Like`). `src/interfaces/CCTPInterfaces.sol` is unchanged and still used by `ForeignController` for CCTP V1.
- `CCTPLib.transfer()` now receives the `mintRecipients` mapping by storage reference and resolves the recipient itself, instead of receiving a pre-resolved recipient.
- `MintRecipientSet` now indexes its `mintRecipient` parameter.

3.2 Changes in Version 2

In **Version 2** , `CCTPLib` has been renamed to `CCTPV2Lib` , and its interface `ICCTPLike` to `ICCTPV2Like` . The `CCTPTransferInitiated` event, the rate limit keys and the constants are unchanged.

4 Trust Model

The trust model of the ALM Controller is unchanged by this version. This section restates the roles relevant to the CCTP integration and records the assumptions that the migration to CCTP V2 adds.

4.1 Roles

`DEFAULT_ADMIN_ROLE` (`admin`) of `MainnetController`

- Trust level: Fully trusted.
- Permissions: Grant and revoke the `RELAYER` and `FREEZER` roles. Configure the mint recipient per destination domain through `setMintRecipient()`. Configure the global CCTP fee cap through `setCCTPMaxFeeRate()`. Through the deployment and upgrade flow, choose the `cctp` messenger address a controller is bound to.
- Worst-case impact: Set the mint recipient of a domain to an address it controls and relay the full CCTP rate limit to itself, stealing the bridged USDC.

`RELAYER` of `MainnetController`

- Trust level: Partially trusted.
- Permissions: Call `transferUSDCToCCTP()`, which burns USDC held by the `ALMProxy` on mainnet and mints it to the pre-configured recipient on the destination domain, bounded by the rate limits.
- Worst-case impact: Move USDC to the configured recipients on other domains up to the rate limits, and exhaust those rate limits, delaying legitimate bridging until they recharge. It cannot redirect funds.

`FREEZER` of `MainnetController`

- Trust level: Partially trusted.
- Permissions: Freeze the controller, which disables `transferUSDCToCCTP()` along with all other relayer functions.
- Worst-case impact: Denial of service of the bridging path.

4.2 External Dependencies

Circle CCTP V2

- `TokenMessengerV2`, `TokenMinterV2` and `MessageTransmitterV2` are assumed to work honestly and correctly as documented.
- Circle governance is trusted not to misbehave. It can pause the protocol, blocklist the `ALMProxy` or the mint recipients, or change `burnLimitsPerMessage()`, each of which denies service to the bridging path. In the worst case it can upgrade the CCTP contracts and seize the USDC in flight.
- The off-chain attestation service is assumed to be live and to attest valid burns. A burn that is never attested leaves the USDC destroyed on the source chain with no corresponding mint on the destination chain.

USDC

- USDC is assumed to behave as a standard ERC-20 with 6 decimals. The CCTP integration deliberately does not use `ApproveLib` and performs a plain `approve()` on the assumption that the token is USDC.
- Circle can blocklist the `ALMProxy`, which denies service to the bridging path.

5 System Considerations

We leverage this section to highlight findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require a modification inside the project. Instead, they should raise awareness in order to improve the overall understanding for users and developers.

SC1 CCTP V2 Reuses the CCTP V1 Rate Limits

System Consideration **Version 1**

The rate limit keys `LIMIT_USDC_TO_CCTP` and `LIMIT_USDC_TO_DOMAIN` are unchanged by this version. Hence, the limits configured for CCTP V1, including their remaining capacity, apply to CCTP V2 transfers without any reconfiguration. Both limits are decreased by the full burned amount, including any fee deducted on the destination chain. Spark should ensure that the configured limits remain appropriate for CCTP V2.

SC2 Diverging CCTP V2 Integrations

System Consideration **Version 1**

CCTP V2 is implemented twice, in `CCTPLib` for `MainnetController` and in the Diamond PAU `CCTPFacet`. Some notable differences are:

- `MainnetController` configures the fee cap globally, `CCTPFacet` per domain.
- `MainnetController` uses a fixed rate (`cctpMaxFeeRate`), `CCTPFacet` a range (`minFeeCapRate` to `maxFeeCapRate`) within which the allocator chooses the rate per transfer.
- `MainnetController` does not include `maxFee` in its transfer event, `CCTPFacet` does.
- `MainnetController` accepts a zero mint recipient in `setMintRecipient()`, as in previous versions, `CCTPFacet` rejects it.
- The standard finality threshold constant is named `MAX_FINALITY_THRESHOLD` in `CCTPLib` but `MIN_FINALITY_THRESHOLD` in `CCTPFacet`.

SC3 Event Changes Affect Indexers and Relayers

System Consideration **Version 1**

This version changes two events emitted by `MainnetController`:

- `CCTPTransferInitiated` no longer contains the `nonce`, as CCTP V2 does not return one at burn time. Its signature, and thus its topic, changes.
- `MintRecipientSet` now indexes `mintRecipient`, which moves it from the event data into an additional topic. As the event signature is unchanged, consumers using the previous ABI decode the event incorrectly.

Off-chain consumers such as indexers and relayers have to be updated accordingly. Apart from the added `setCCTPMaxFeeRate()` and `cctpMaxFeeRate()`, the function ABI of `MainnetController` is unchanged.

SC4 ForeignController Remains on CCTP V1

System Consideration | Version 1

This version migrates only the mainnet controller. `ForeignController` is left unchanged and keeps its CCTP logic against the V1 interface. This is intended, as CCTP V2 on the foreign chains is introduced through a separate Diamond PAU controller with a CCTP V2 facet, deployed alongside `ForeignController` and operating the same `ALMProxy` (see spark-address-registry#115 for Arbitrum One and Base). The test suite reflects this split:

```
address internal constant BASE_CCTP_TOKEN_MESSENGER_V1 = Base.CCTP_TOKEN_MESSENGER_V1;  //
    ForeignController stays on v1
```

Hence, the `MainnetController` alone does not provide a full CCTP V2 setup. Transfers from mainnet are minted directly to the `ALMProxy` on the destination chain and do not depend on the foreign controllers. Transfers to mainnet, however, require the Diamond PAU controller on the respective foreign chain to be active, as `ForeignController` can only use CCTP V1, which Circle pauses on December 1, 2026.

6 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- **Likelihood** represents the likelihood of a finding to be triggered or exploited in practice
- **Impact** specifies the technical and business-related consequences of a finding
- **Severity** is derived based on the likelihood and the impact

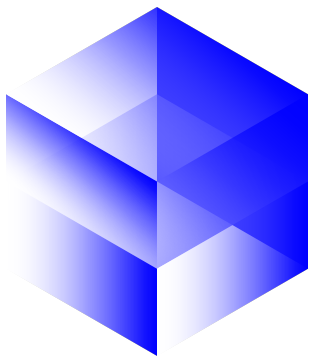
We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	**Impact**		
	High	Medium	Low
High	**Critical**	**High**	**Medium**
Medium	**High**	**Medium**	**Low**
Low	**Medium**	**Low**	**Low**

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

7 Findings

In this section, we describe the findings identified during the course of the engagement. The findings are categorized by severity as explained in the Terminology section. Below we provide an overview of the total number of findings per severity, as well as the number of open issues.



No Open Issues

Total number of findings

Critical	High	Medium	Low	Informational
0	0	0	0	2

7.1 Overview

The following table lists all identified findings along with their severity and current status. A finding is considered resolved once it has been fully corrected or the specification has been changed accordingly. Non-informational findings with any other status, including partial fixes, acknowledgements, and accepted risks, remain open.

Info	#001	`cctpMaxFeeRate` May Be Set to 100%	Code Corrected
Info	#002	USDC Allowance Is Not Reset After a Transfer	Code Corrected

7.2 Descriptions

#001 `cctpMaxFeeRate` May Be Set to 100%

Informational **Version 1** **Code Corrected**

`setCCTPMaxFeeRate()` bounds the configured rate to `<= 10_000`, and `CCTPLib.transfer()` computes the `maxFee` of each chunk as a percentage of the chunk amount (`amount * cctpMaxFeeRate / 10_000`). Since the bound is inclusive, `maxFee == amount` is possible, which CCTP V2 rejects (`maxFee < amount` is required). Hence, every `transferUSDCToCCTP()` reverts until the rate is lowered. Moreover, rates just below `10_000` are accepted and allow nearly the whole amount to be consumed as fee. Note that the Diamond PAU `CCTPFacet`, which integrates CCTP V2 for the parallel controllers, rejects this value (`maxFeeCapRate < 10_000`).

Code Corrected

`setCCTPMaxFeeRate()` now requires `maxFeeRate < 10_000`, matching the bound of `CCTPFacet`.

#002 USDC Allowance Is Not Reset After a Transfer

Informational **Version 1** **Code Corrected**

`CCTPLib.transfer()` approves `usdcAmount` to the CCTP messenger before the chunking loop and does not reset the allowance afterwards. Under the current CCTP V2 implementation each `depositForBurn()` pulls exactly the chunk amount, so the allowance ends at `0` and no funds are at risk. This, however, relies on the behavior of an external contract. Resetting the allowance to `0` after the loop would make the property hold locally.

Code Corrected

`CCTPV2Lib.transfer()` (formerly `CCTPLib.transfer()`) now resets the allowance of the CCTP messenger to `0` after the chunking loop.

8 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. 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's code 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.