PUBLIC

Code Assessment

of the CCIP Bridge Smart Contracts

May 15, 2025

Produced for

Frankencoin

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Frankencoin Team,

Thank you for trusting us to help Frankencoin with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of CCIP Bridge according to Scope to support you in forming an opinion on their security risks.

Frankencoin implements a bridging system for the ZCHF token as well as some governance mechanisms. ZCHF on chains other than mainnet allow for tokens to be deposited into savings contracts that earn interest. Accrued interest, the interest rate, and governance voting power are asynchronously and permissionlessly transferred between mainnet and the respective sidechains using the bridge.

The most critical subjects covered in our audit are functional correctness and the synchronization of the state between mainnet and the sidechains.

Functional correctness was improved after a wrong assertion, that could have led to loss of data, was corrected.

The synchronization of data between different chains is sufficient although some caveats in regards to out-of-order execution and partial data transfers should be noted.

In summary, we find that the codebase provides a good level of security.

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

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	1
• Code Corrected	1
Low-Severity Findings	3
• Risk Accepted	2
• Acknowledged	1

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 source code files inside the CCIP Bridge repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	04 May 2025	7aa8e90193a44f9cd4bd24639a3ca6c50ac68aaa	Initial Version
2	14 May 2025	a5af62a529a3b8fbbf23f0c83128aca5fba281f9	Version with Fixes

For the solidity smart contracts, the compiler version `0.8.24` was chosen. The following files were in scope:

```
contracts/bridge/CCIPAdmin.sol
contracts/bridge/CCIPSender.sol
contracts/equity/BridgeAccounting.sol
contracts/equity/BridgedGovernance.sol
contracts/equity/GovernanceSender.sol
contracts/erc20/CrossChainERC20.sol
contracts/rate/BridgedLeadrate.sol
contracts/rate/LeadrateSender.sol
contracts/savings/BridgedSavings.sol
contracts/stablecoin/BridgedFrankencoin.sol
```

2.1.1 Excluded from scope

Any file not listed explicitly above is excluded from the scope. Furthermore, external token contracts used as collateral in the system were not in the scope of this code assessment. Moreover, third party libraries are assumed to behave correctly according to their specification.

2.2 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.

Frankencoin offers the multichain extension to the pre-existing Frankencoin system. Besides allowing to bridge ZCHF tokens between chains, it extends the savings module and the governance functionality to also operate on sidechains. Collateralised minting of ZCHF and investing in equity reserve pool shares (FPS) is currently envisioned to remain on Ethereum Mainnet exclusively. The bridge used by the protocol is Chainlink's CCIP.

This review was specifically focused on the multichain extension of the system; for an overview of the functionality currently deployed on Mainnet, you can refer to our two past audit reports of the initial core system and of its first extension.

2.2.1 Bridging ZCHF

In order for ZCHF tokens to be permissionlessly bridged between supported blockchains using CCIP, a `TokenPool` must be configured by the token owner to integrate with the router and comply with the expected interface: on every chain, this pool will be a `BurnMintTokenPool` deployed from Chainlink's code repository of audited pools. Employing the burn-and-mint strategy means that the pool needs to be granted minting privileges for the token; while this is trivial for the yet-to-be-deployed bridged versions of ZCHF, it will instead have to pass through the ordinary governance procedures to enable a minter, on mainnet.
The power to manage the TokenPool, or to switch to a new one entirely, is assigned to a non-upgradeable `CCIPAdmin` contract. While this assignment can be automated for the bridged tokens (which can self-report the contract to be their CCIP admin), it will require Chainlink's manual intervention for the mainnet version (which does not already self-report an `owner()` or a CCIP admin address).
All of the functionality of the `CCIPAdmin` contract is operated by the governance: Only a qualified quorum (2% of the voting power) can propose new actions, which, within a fixed delay, can be vetoed by any qualified quorum. The exposed functions are:

- `registerToken()`. Only called on the sidechain. Assigns the CCIP Admin role for ZCHF to this contract and sets the token pool.
- `acceptAdmin()`. Called on mainnet. Equivalent to `registerToken()`, but has to follow up to Chainlink's manual proposition of a CCIP Admin role for the mainnet ZCHF token.
- `proposeRemotePoolUpdate()` / `applyRemotePoolUpdate()`. Subject to a delay of 7 days, this proposal can add or remove remote pools, on an already-supported blockchain. This is useful to phase out the old remote pool when doing an upgrade on the remote domain.
- `proposeRemoveChain()` / `applyRemoveChain()`. Subject to a delay of 7 days, this proposal disables cross-chain token transfers to the specified blockchain.
- `proposeAddChain()` / `applyAddChain()`. Subject to a delay of 7 days, this proposal enables a new blockchain for cross-chain token transfers and sets the initial configuration (remote pool, rate limits, etc.).
- `proposeAdminTransfer()` / `applyAdminTransfer()`. Subject to a delay of 21 days, this proposals initiates the two-step transfer of the CCIP Admin role and the `owner()` role of the TokenPool to a new address.
- `applyRateLimit()`. This is the only action that has no delay and thus cannot be vetoed on; still, it can only be launched by a qualified quorum. It applies the specified rate limits to the specified chain. The reason for not subjecting this action to a delay is to allow quick responses to a remote chain hack.
- `deny()`. Any qualified quorum can veto a pending proposal. Vetoed proposals, as well as succeeded proposals, can be proposed again in the future.

This setup enables and configures cross-chain transfers of ZCHF tokens. In order to facilitate such transfers *from a sidechain*, the bridged version of the token includes a modified `transfer()` function which takes an additional `targetChain` parameter, as well as a `getCCIPFee()` function to help estimate the bridge fees. On the contrary, the immutable ZCHF token on mainnet does not include these convenience functions: in order to bridge tokens out of mainnet, one needs to manually interact with the CCIP router to estimate the fee and to initiate the cross-chain transfer.

2.2.2 Bridging the Governance

The aforementioned actions of the `CCIPAdmin` contract, as well as the minter proposals of the bridged Frankencoin contract (more on that later), require the presence of a `BridgedGovernance` contract that can attest to a quorum of voters being or not being qualified (i.e., surpassing 2% of the total voting power).
In much the same way as the Equity contract on mainnet, this contract supports full delegation of one's voting power to another address, while also retaining the voting power for oneself; chain delegations "propagate" the voting power up to the "root"; this is the mechanism whereby a single address represents the voting power of all those who have delegated to it: The delegate address, together with an array of delegators, is enough to check whether the corresponding quorum is qualified.
However, the FPS token itself is not bridged, and therefore the mechanism allowing the voting power to automatically increase with time is impossible to perfectly replicate on the sidechains: The `BridgedGovernance` only has "partial snapshots" of the voting power (possibly dating to different timestamps) that are bridged out of mainnet by interested users. Quorum decisions, expressed by the functions `votesDelegated()` and `checkQualified()`, are made based on this "incomplete", yet sufficient, picture of the situation on mainnet.
The contract in charge of snapshotting the voting power on mainnet and syncing it to other chains is the `GovernanceSender`. It exposes two functions:

- `syncVotes()`. Permissionless function. Given a user-provided array of addresses, it snapshots their current voting power, their current delegate address, and the current total voting power in the system. This information is packed in a cross-chain message and sent over to the remote `BridgedGovernance`.
- `getSyncFee()`. Computes the fee required to pay for the cross-chain message to be sent in `syncVotes()`.

The `BridgedGovernance`, upon reception of this message, updates its internal accounting to store the latest provided snapshots, the corresponding delegations, and the latest `totalVotes` count.

2.2.3 Bridging the savings module

The savings module already present on mainnet allows users to deposit and withdraw ZCHF at will, through the functions `save()`, `withdraw()`, and `adjust()`. These deposits accrue a non-compounding interest: Users have to manually call `refreshMyBalance()` to realise the interest and start compounding. The interest rate (managed in the base abstract `Leadrate` contract) is controlled by the governance: a qualified quorum can propose updates that can be executed after 7 days if no other quorum objects (by just proposing a new update with the current rate). The interest gets paid out as system debt, which gets properly accounted for by the `coverLoss()` function in the Frankencoin contract.
This module will be redeployed on mainnet with an additional functionality: the option to set a referral fee (levied on the accrued interest, not deposits or withdrawals). Front-end operators can charge this fee to their users; however, the users can at any time drop the referrer and set the fee to 0 by simply interacting with the smart contract directly. The new module will have to be approved as a minter by the governance.

A virtually identical savings module will be deployed on all the sidechains, with two major differences.
The interest rate (managed in the base abstract `BridgedLeadrate` contract) is not independently voted on by the sidechain governance; instead, it is simply bridged over from mainnet by a dedicated `LeadrateSender`. This contract has a permissionless `pushLeadrate()` function (with its batched variant) to send a cross-chain message to the remote `BridgedLeadrate` contract (i.e. the savings module on the sidechain) informing it of the new interest rate.
Additionally, the `coverLoss()` function, used by the savings module to pay out interest, is implemented differently in the `BridgedFrankencoin` contract (see next section). Its end result, however, is still that the savings module gets transferred the specified amount of ZCHF, so it does not have to worry about the implementation details.

2.2.4 Bridging profits and losses

The `BridgedFrankencoin` contract mainly differs from its mainnet counterpart in the way it handles profits and losses, i.e., how the functions `collectProfits()` and `coverLoss()` are implemented. As it is the case on mainnet, profits and losses are absorbed by the governance contract; should the losses outweigh the profits (which is expected to be the case most of the time, since, apart from the savings contract, no other minting contracts are allowed on sidechains), the governance's balance will just go to 0 and the difference will be minted by the ZCHF contract and recorded as a debt, stored in a variable called `accruedLoss`. New incoming profits will first offset this debt, then go to the governance.
This debt is left to be handled by the mainnet ZCHF contract, through a bridge call. Anyone can call the permissionless `synchronizeAccounting()` function, which transmits a cross-chain message to the dedicated `BridgeAccounting` contract on mainnet. This message contains the current value of `accruedLoss` (which then gets reset to 0), and the current balance of the governance contract (which gets then emptied and sent over to the `BridgeAccounting` through a token transfer which "piggybacks" on the cross-chain message). Notice that an invariant holds that, at all times, one of `accruedLoss` and the governance balance is 0 (up to donations to the governance contract).
Once the `BridgeAccounting` contract receives the message on mainnet, it handles the received profits through a call to `collectProfits()` and the received losses through a call to `coverLoss()`.

As was mentioned, no minting module is currently foreseen to be deployed on sidechains. Nevertheless, the `BridgedFrankencoin` contract does have the necessary functionality to propose and veto new minters (using votes from the `BridgedGovernance`), as well as to register minters' positions. This is largely the same as in the mainnet contract, with the only difference being that no minter reserve is set aside for use by some special functions (`mintWithReserve()` and the like), which are completely absent in the sidechain contract.

2.2.5 Changes in Version 2

In Version 2, the functions of the `GovernanceSender` have been renamed:

- `syncVotes()` is now called `pushVotes()`.
- `getSyncFee()` is now called `getCCIPFee()`.

2.3 Roles and trust model

This system is fully decentralized, therefore no admin roles exist that can act outside of the will of the users. Only one step in the bootstrapping phase involves a privileged action: the registration of a CCIP Admin address for the Mainnet ZCHF token; this registered address is trusted to be the `CCIPAdmin` contract.
The deployment and initial configuration are out of the scope of this review and is trusted to be non-malicious; in particular, contracts are expected to be deployed without proxy and the `BridgedFrankencoin` contract is trusted not to recognise any initial minters besides the savings module and the token pool.
The CCIP bridge is trusted to be live and to always work correctly and according to specification. The blockchains on which this system will be deployed are trusted to be operated non-maliciously, and to always behave according to specification.
Users of the system are generally considered untrusted.

3 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.

4 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.

5 Open Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	3

5.1 Initialization Frontrunning

Security Low Version 1 Risk Accepted

CS-ZCHF-CCIP-001

Some contracts (e.g., `BridgedFrankencoin`) have `initialize()` functions without access control even though they are not behind proxies. This makes it impossible without auxiliary contracts, or smart accounts, to deploy the contracts and call their `initialize()` function atomically.

This could result in deployment DoS as malicious actors could frontrun these calls.

Risk accepted:

Frankencoin is aware of the risk and accepts it. Frankencoin states:

> If there is a malicious party that DoSes our deployment process we will either create a MEV-Bundle or create a small deployment contract that creates the necessary contracts atomically.

5.2 Missing Zero-Rate Check

Correctness Low Version 1 Acknowledged

CS-ZCHF-CCIP-002

`AbstractSavings.save()` does not allow the deposit of new funds when the lead rate is 0. In an older version, the following check was done additionally:

```
if (nextRatePPM == 0 && (nextChange <= block.timestamp + INTEREST_DELAY)) revert ModuleDisabled();
```

This ensures that no deposits happen even when the change of the lead rate to 0 is already active but has not been applied yet.

While the check on L2s is not necessary, it still makes sense to check this on L1 (or, alternatively, apply lead rate changes directly).

Acknowledged:

Frankencoin said:

```
This check is no longer necessary, as fund withdrawals are no longer delayed and there is no locking period. The check has been updated to only verify that the current rate is greater than zero if the module is active.
```

5.3 Vote De-Synchronization

Security | Low | Version 1 | Risk Accepted

CS-ZCHF-CCIP-003

Reducing votes on mainnet can lead to a de-synchronization of the vote counts on other chains.

If votes are sent to the `BridgedGovernance` contract and then immediately reduced on mainnet, the votes can still be used to vote on L2.

While this, in itself, is not problematic, governance users can decrease the voting power of other actors themselves using the `kamikaze()` function. Consider the following example:

1. User A has a voting power of 4%.
2. User B and user C each have a voting power of 2%.
3. User A wants to create a proposal on some L2. He knows that users B and C are the only users opposing their proposal.
4. User A syncs his own votes to the L2.
5. User A uses the `kamikaze()` function on User B and C, leaving all three users with 0 voting power.
6. User A now syncs the voting power of user B and C to the L2. They both now have 0 voting power on the L2 while user A still holds his 4%.
7. User A can now create his proposal without opposition.

This attack is also possible directly on mainnet.

Risk accepted:

Frankencoin is aware of the risk and accepts it. Frankencoin said:

```
Users B and C can still voice their concerns in the community and assemble votes to surpass the required 2% quorum and veto the proposal of user A.
```

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	1

- Wrong Assertion Code Corrected

Low-Severity Findings	0
Informational Findings	4

- Ambiguous Naming Code Corrected
- Typographical Errors Code Corrected
- Wrong Comment in Cross-Chain transfer() NatSpec Comments Code Corrected
- BridgeAccounting Contract Does Not Check Whether Source Chain Is Supported Code Corrected

6.1 Wrong Assertion

Correctness Medium Version 1 Code Corrected

CS-ZCHF-CCIP-007

`BridgeAccounting._ccipReceive()` contains the following assertion:

```
assert(ZCHF.balanceOf(address(this)) == 0);
```

This is based on the assumption that the function previously sends out all ZCHF tokens to another address. The function receives two values in the respective CCIP message:

- `profits`
- `losses`

If profits are greater than 0, the full balance is sent to the `Equity` contract. If losses are greater than 0, the amount of losses that have been sent are burned.

While the assertion holds when profits are greater than 0, it does not hold when losses are greater than 0, since a malicious actor could donate tokens to the `BridgeAccounting` before execution.

The failed message would need to be successfully re-executed in order to repair the consistency of the system (i.e. to actually handle the bridged losses); however, by CCIP rules, this can only happen within a certain timeframe (usually 1 hour, depending on the chain) otherwise the message becomes permanently non-executable. Therefore, the only way to remedy is to bridge over some profits before the specified timeframe elapses.

Code corrected:

The assertion has been removed.

6.2 Ambiguous Naming

Informational | Version 1 | Code Corrected

CS-ZCHF-CCIP-008

Message-sending functions and their corresponding fee-calculation functions do not have a consistent naming scheme.

For example, the function for syncing the lead rate is called `pushLeadrate()` while the function for syncing votes is called `syncVotes()`.

Similarly, the function for calculating the fee for the lead rate synchronization message is called `getCCIPFee()` while the vote-syncing equivalent function is called `getSyncFee()`.

Code corrected:

The naming convention is now more uniform: `syncVotes()` has been renamed to `pushVotes()` and `getSyncFee()` has been renamed to `getCCIPFee()`.

6.3 Typographical Errors

Informational | Version 1 | Code Corrected

CS-ZCHF-CCIP-009

The following (non-exhaustive) list of typographical errors have been identified:

1. `CCIPAdmin:L12` contains the word "*briding*".
2. `CCIPSender:L50` contains the word "*combition*".
3. `Governance:L71` contains the word "*helpes*".
4. `BridgedFrankencoin` uses the variable name `_accuredLoss` on multiple occasions.

Code corrected:

The listed typographical errors, as well as several others, have been corrected.

6.4 Wrong Comment in Cross-Chain `transfer()` NatSpec Comments

Informational | Version 1 | Code Corrected

CS-ZCHF-CCIP-005

The NatSpec comments above the cross-chain version of the function `CrossChainERC20.transfer()` say that the user needs to approve the token contract itself. However, since the internal `_transfer()` function is used, no approval is required.

Code corrected:

The comments have been updated to clarify that the token to be approved is not the bridged ZCHF itself, but the fee token (LINK), unless the CCIP fee is paid in the chain's native token.

6.5 `BridgeAccounting` Contract Does Not Check Whether Source Chain Is Supported

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

CS-ZCHF-CCIP-006

The function `_validateSender()` in the `BridgeAccounting` contract takes `TokenPool.getRemoteToken()` as the expected source address, comparing it to the actual one as stated by the CCIP bridge. `getRemoteToken()`, however, simply returns the remote token from the respective storage struct without checking that it has been written to before (i.e., it returns an empty `bytes` array for unsupported chains). If, hypothetically, a chain existed where a privileged actor could send transactions from an "empty" address and that chain were supported by CCIP, the `_validateSender()` check would pass and that privileged actor could send an arbitrary message to the `BridgeAccounting` contract, causing arbitrary profits and losses.

Code corrected:

A check has been added requiring that the address returned by `getRemoteToken()` is not an empty `bytes` array.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Duplicated Code

Informational **Version 1** **Acknowledged**

CS-ZCHF-CCIP-004

`Frankencoin` and `BridgedFrankencoin` share some functions (e.g., `suggestMinter()`) with the same code which could be extracted out into an abstract contract.

Acknowledged:

Frankencoin states:

> `This was done deliberately because we probably will never do a new mainnet deployment.`

8 Notes

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

8.1 Asynchronous Delegations

Note Version 1

Votes are strictly synced from mainnet to other chains unidirectionally. While delegations are synced in the same way, they can also be created on the L2s directly.

Users creating delegations on L2 should be aware that anyone can overwrite their delegations at any time if they are not in sync with mainnet.

8.2 Effective Governance Quorum Can Be Temporarily Lowered

Note Version 1

As mentioned in Vote de-synchronization, user A can `kamikaze()` someone else using their full voting power `x`, then only bridge the victim's voting power. This would lead to the snapshot on the sidechain being inconsistent with A's voting power still being `x`, while the total vote count being `T-2x` (T being the old value for the vote count). A quick calculation shows that, for `x` to be above 2% of `T - 2x`, it is sufficient for `x` to be 1.923% of `T`. Therefore, this attack temporarily lowers the governance threshold to reach a qualifying quorum. The threshold can be restored again, however, by anyone bridging A's voting power.
A less-effective version of this attack involves user A calling `redeem()` and then bridging somebody else's voting power. The snapshot on the sidechain then has `x` as A's votes, and `T-x` as the total votes count.

8.3 Governance Discrepancy in CCIPAdmin

Note Version 1

`CCIPAdmin` enforces a 7-day waiting period for the removal of a chain. However, setting the rate limit of the same chain to 0 does not trigger a waiting period and has a similar effect.

8.4 Monitoring Expected

Note Version 1

Since the chosen setup (out-of-order execution) can lead to slight synchronization differences, it is important to monitor the system for any anomalies.

Consider the following example:

1. The lead rate is 3%.
2. The lead rate is changed to 2% on mainnet.

3. A user frontruns this change and syncs the 3% lead rate to the L2, setting out-of-order execution, but does not execute it.
4. The new 2% lead rate is synced to the L2 and executed.
5. The user now executes their message containing the old data (if the message is still executable).

The lead rate now is still 3% on the L2. Any further synchronization will bring it back to the correct value of 2%. However, if the out-of-order execution was never observed, this additional synchronization might not happen in a timely manner. It is therefore expected that all contracts are monitored and synchronization anomalies trigger the necessary actions.

8.5 Out-of-order Vote Synchronization

Note **Version 1**

All synchronization functions allow `extraArgs` that allow the user calling the function to specify that the message can be executed out of order. This is also true for `GovernanceSender.syncVotes()`.

Successfully bridged messages stay executable for a certain period of time (usually 1 hour, depending on the chain). Users should be aware that this allows actors to overwrite bridged votes / delegations with data that is at most the mentioned time-period old.