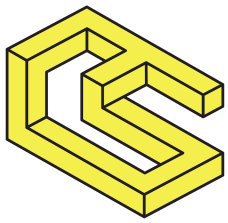


ChainSafe

Toronto, Canada
chainsafe.io

xx Network Substrate Chain

Code Review, Testing and Benchmarking

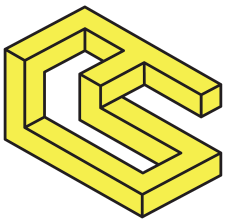
Prepared for:

xx Network

Prepared by:

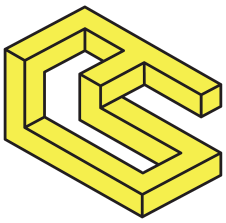
Willem Olding | ChainSafe Systems

willem@chainsafe.io



Warranty

This Code Review is provided on an “as is” basis, without warranty of any kind, express or implied. It is not intended to provide legal advice, and any information, assessments, summaries, or recommendations are provided only for convenience (each, and collectively a “recommendation”). Recommendations are not intended to be comprehensive or applicable in all situations. ChainSafe Systems does not guarantee that the Code Review will identify all instances of security vulnerabilities or other related issues.

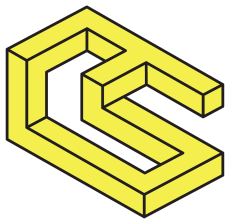


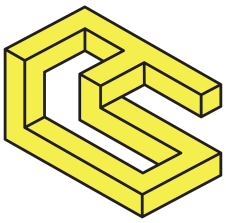
Engagement Summary

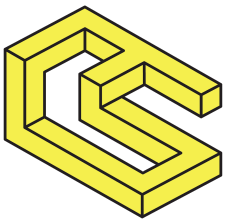
Project Type	Substrate Blockchain
Repository	https://gitlab.com/xxnetwork/xx-substrate
Commit	`849dfa3239fa456ee2541e5f03fd3f26aca546ed`
Dates	16/8/2021 - 17/9/2021
Engagement Type	Code review and automated testing

Document History

Version	Changes
xx-review-16-9-2021	Initial report version
xx-review-20-9-2021	Update CS-XX-05 given clarification
xx-review-22-9-2021	Include review of code fixes for each issue
xx-review-22-9-2021-rev1	Adds missing review for fix of CS-XX-01

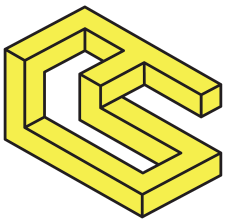






1. Goals

- Validate the modifications made to the ChainBridge pallet and ensure it is correctly integrated with the other pallets in the runtime
- Validate the swap pallet that integrates with the ChainBridge pallet to allow token swapping between Ethereum and the native chain.
- Validate the modifications made to the Substrate FRAME staking pallet
- Validate the functionality added by the new xxNetwork pallet
- Ensure all existing tests are updated to reflect pallet changes and are passing. Write full test suites for new functionality added by the xxNetwork pallet
- Ensure all extrinsics have appropriate weights through automated benchmarking



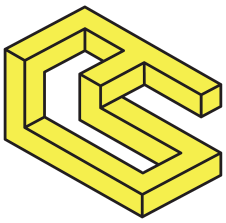
2. Executive Summary

For the period of the 16 August to the 17 September ChainSafe Systems was engaged by xx Network to conduct a code review of the xx Network blockchain built with the Substrate framework. In addition to the code review ChainSafe was also enlisted to write a comprehensive test suite for all added functionality and to derive appropriate weights for all extrinsics.

The first week was spent getting an understanding of the expected operations of the xx Network blockchain and conducting a review of the diff between the modified pallets (`ChainBridge`, `Swap`, `Staking`) and their original implementations. Part of this included reviewing the existing tests and modifying them where appropriate. No major issues were found during this phase.

The second week focused further on the staking pallet and concluded the migration of the test suite. After discussion with the xx Network team it was decided that the XXNetwork pallet should be split into three pallets - `xx-cmix`, `xx-team-custody` and `xx-economics`. This refactor was done by ChainSafe as part of the code review. A test suite for the `xx-cmix` and `xx-economics` pallets was completed based on the specification provided by xx Network in the scope of work document.

The third through fifth weeks completed the bulk of the testing phase. 6 issues were found including an exploitable bug in the xx-team-custody pallet (which was present in the original xxNetwork pallet). Automated benchmarking code was added for the pallets lacking existing weights (xx-cmix, xx-economics, xx-team-custody, swap). Some difficulty was encountered in building the runtime for benchmarking due to dependency issues. Fixing this required some changes across the runtime and CLI crates which were submitted in the testing/benchmarking branch. Benchmarking was conducted for all pallets and relevant extrinsics were updated to make use of the derived weights.



Overall the changes to the chainbridge and swap pallets were minor and no issues were found. Changes to the staking pallet were well thought out and implemented. The provided test specification made writing the automated tests a quick process. All changes made as part of this review are available in branches submitted to the repo to be merged into the codebase at the owners discretion.

3. Evaluation of Pallet Modifications

3.1. ChainBridge Pallet

Summary of Changes

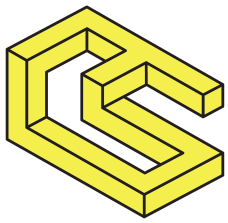
Changes were derived by computing the diff of the review branch with the latest release of the ChainBridge pallet: f2cce10

- All Substrate dependencies in the `Cargo.toml` have been replaced with those from the xx-network fork of Substrate which is a fork of the 4.0.0-dev release
- Serde upgraded from 1.0.101 to 1.0.126
- Changes made for compatibility with Substrate 4.0.0
- Switch from `opaque_blake2_256` to `twox_64_concat` as the hashing algorithm for the keys of all storage maps
- Remove the const `ModuleId` and use a `PalletId` set in the config, exposed as a pallet const
- Minor changes due to upstream changes in Substrate
    - E.g. `Pallet` -> `Module`, `system` -> `frame_system`

Assessment

Migration to a later version has been done with only minor changes to styling and no changes to logic.

The change of hashing algorithm for storage maps from the cryptographic but opaque blake2 to the non-cryptographic but transparent twox-concat is deemed safe in this context as it is impossible for non-privileged accounts to mutate the storage map.



3.2. Swap pallet

The swap pallet is a modified copy of the bridge pallet from Centrifuge, which is in turn a modification of the ChainBridge example pallet.

The pallet can accept the transfer of the native token. This token will be held by the bridge account and trigger a signal to the bridge relayers to initiate the corresponding transfer on the remote chain. It also supports transfering from the bridge account when called by the `BridgeOrigin` (e.g. the relayers have voted to execute a given transaction). It is therefore an example of a token bridge using lock-and-mint/burn-and-release strategy on the lock/release side only. This is to support a wrapped version of XX Coin on Ethereum and/or Binance Smart Chain.

A test suite for the swap pallet was created as part of this review and is included in the chainsafe/handover/tests-and-bench branch.

Summary of Changes

- Accepts a fee when executing a transfer native that is held by a fee custodian account, set by the AdminOrigin
- Removes all functionality not related to transferring in/out of fungible tokens
- Adds extrinsics to manage the fee amount and fee destination account.
- Uses genesis config to set initial values for Chainbridge pallet (e.g. whitelisted chains, resources and initial relayer set)

Assessment

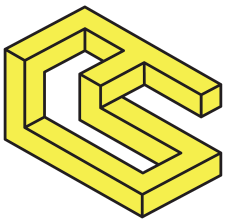
Minor changes made to this pallet appear to have been made without issue and test coverage shows the pallet working as expected when combined with the ChainBridge pallet.

3.3. Staking Pallet

Summary of Changes

Changes were derived by computing the diff of the review branch with commit from which the staking pallet was forked: [ac277db]

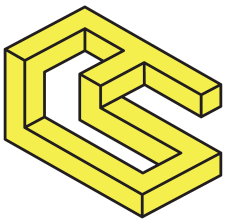
- Validators are associated with a cmix root hash
    - This is stored in the `ValidatorPref` struct which is stored in a map from validator stash key to prefs
    - cmix hashes are stored in a map that can be used to check for uniqueness and error if a validator tries to join with a duplicate cmix hash
- `Config` has an `AdminOrigin`
- `Config` has a `XXNetworkHandler` to allow calling functions on the xxnetwork pallet (split into `CMixHandler` and `CustodianHandler` in refactor)
- Add `end_era` hook that calls back to xxnetwork pallet
- Adds a configurable minimum stake value
    - stored in `MinimumStake` storage item
    - Can be set by `AdminOrigin` `set_validator_min_bond`
    - Checks in validate and unbond to ensure minimum bond continues to be met
- Remove separate reward destination and all rewards to go stash account
    - Simplifies `make_payout` since there is only one valid reward destination



- Introduces another class of account, custody accounts, which can use their funds to nominate validators but have zero exposure so cannot earn rewards or be slashed.
- Calls `T::Reward::on_unbalanced(imbalance)` in `do_payout_stakers` to decrease the value of the Rewards pool.
    - This included in the base staking pallet but not used
- Change how points are assigned to block producers, instead of the default: -20pts for block authorship, 2 pts for referencing a new uncle, 1 pt for authoring an uncle, points are calculated as:
    - `T::XXNetworkHandler::get_block_points()` for the block author
    - None for authoring/referencing uncles
- Change `reward_by_ids` to initialize new validators with 1 point rather than with 0

As part of the review the existing test suite for the staking pallet was updated to reflect the changes made. The majority of changes were around how reward points are assigned and the payouts destination.

An additional test suite for added functionality was created as part of this review and is included in the chainsafe/handover/tests-and-bench branch in `staking/src/xx_tests.rs`.



Issues

Updating the ValidatorMinBond does not affect existing bonded accounts

Identifier	CS-XX-01
Severity	Informational
Tests	`N/A`
Resolved	`Yes (commit` 6fa5dcd)

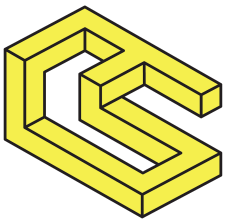
The minimum bond requirement is only checked on calls to validate and unbond. This means that updating the value of the minimum bond will only affect new validators or existing validators if they attempt to reduce their bond amount.

An alternative would be to iterate the full list of validators and chill those who do not meet the minimum requirement on a call to `set_validator_min_bond`. This adds a risk that calling this extrinsic may be prohibitively large to fit in one block for a very large validator set.

Follow-up

Commit 6fa5dcd adds a `chill_other` extrinsic to the staking pallet which allows anyone to force a validator to chill if their bond is less than the current minimum validator bond. Also adds some new tests for this extrinsic.

This addresses the issue without requiring a potentially unbounded gas cost for calling `set_validator_min_bond` as described above.



Existing validators cannot update validator prefs by calling validate

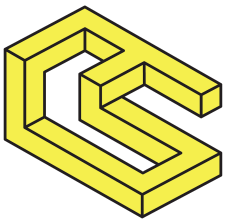
Identifier	CS-XX-02
Severity	Informational
Tests	`N/A`
Resolved	`Yes (commit` 6fa5dcd)

Calling validate to update validator prefs will fail if the validator is already registered due to how the uniqueness of cmix root hashes are checked. This requires them to unregister and then register again. This is a regression from the original staking pallet which does allow validators to update their preferences with a single call.

Follow-up

Commit 6fa5dcd adds a fix for this issue. This only applies the check for a unique cmix root if a new validator is being added and ensures that the cmix root has not changed in the case of existing validators.

Test case `calling_validate_with_existing_cmix_root_fails` shows successful updating of validator prefs.

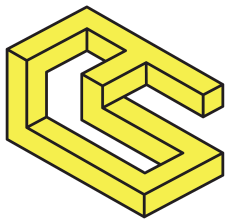


Assessment

The cmix root, minimum stake and enforced reward destinations have been added with minimal impact on the operations of the pallet. Tests in file `xx_tests.rs` illustrate these working as expected. Some minor changes could be made to allow validators to update their preferences in a single call, as in the original staking pallet.

Tests show the imbalances created during payouts are successfully passed to the Reward handler allowing the rewards pool to be updated. As designed, custody accounts are unable to be slashed or receive rewards.

Some further investigation may be required into the impact of removing rewards for uncle block production and uncle block referencing. It is outside the scope of this review to undertake this research but caution should be exercised when altering consensus related parameters.



4. xx Network Pallets Review

4.1. Team Custody Pallet

Overview

This pallet is responsible for managing the vesting of xx coins allocated to team members at chain genesis. These coins are not immediately available and vest at a fixed rate over a given period. The team members are not able to use the unvested coins to participate in nominations until they are vested. The vested coins for each team member are split between a custody and a reserve account.

This pallet adds the ability for a privileged set of custodians to bond and use the unvested funds to participate in governance and staking (although they do not accrue rewards due to modifications to the staking pallet).

Team members can receive their unvested balance by calling the payout extrinsic. This will transfer the unvested amount into their account. It will prefer withdrawal from the custody account if available but if funds are bonded will withdraw from the reserve account. It is the responsibility of the custodian to ensure there are sufficient unbonded funds to pay the team member at each payout interval.

At the conclusion of the vesting period, a call to payout will forcefully unbond all balance and transfer the full amounts in both the custody and reserve accounts to the team member.

A tests suite covering all extrinsics was created as part of the review and is included in the chainsafe/handover/tests-and-bench branch.

Issues

Transfering balance to custody account leads to underflow error

Identifier	CS-XX-03
Severity	High
Tests	`payout_pays_out_additional_contributions_to_custody_account` `payout_pays_out_additional_contributions_to_reserve_account`
Resolved	`Yes (commit` 6fa5dcd)

It is possible at any time to transfer funds to a team member's custody or reserve accounts. This possibility was not accounted for in `custody.rs` when calculating the amount to withdraw in a given payout when using this value to update the value of the total amount under custody (`TotalCustody`). This logic can be seen at lines 309-328 in `custody.rs`.

```
309    // 3. Calculate amounts to withdraw from custody and reserve
310    let withdraw_custody = amount.min(custody_balance);
311    let withdraw_reserve = amount - withdraw_custody;
312    let withdraw_reserve = withdraw_reserve.min(reserve_balance);
313    let withdraw = withdraw_custody + withdraw_reserve;
314
315    // 4. Make transfer from custody, if possible
316    if !withdraw_custody.is_zero() {
317          // Transfer from custody to team member account
318          <T as Config>::Currency::transfer(
319          &custody,
320          &who,
321          withdraw_custody.into(),
322          AllowDeath
323          )?;
```

```
324         // Emmit TeamPayoutCustody event
325         Self::deposit_event(RawEvent::PayoutFromCustody(who.clone(),
    withdraw_custody));
326         // Update total amount under custody
327         <TotalCustody<T>>::mutate(|n| *n -= withdraw_custody);
328     }
```

Prior to this code snippet the value of `custody_balance` is set by directly reading the reducible balance of the team members custody account. On line 310, `withdraw_custody` is set using the minimum value of the balance and the expected payout. The intention here being that the payout be taken preferentially from the custody account and the remainder from the reserve account.

In calls to payout after the custody period has ended, if balance has been added to the custody account this can result in greater balance being withdrawn from the custody account than was allocated to it. This leads to allocated funds being stuck in the reserve account and, much more seriously, to incorrect deductions being made to the `TotalCustody` (line 327). This can lead to an underflow panic and failure of all subsequent calls to payout leaving the remainder of funds in custody or balance accounts locked.

This attack can be conducted by any actor as there are no limitations on who can transfer balance to custody accounts.

Recommendations

- Introduce additional fields to the `CustodyInfo` struct to keep track of the amounts initially allocated to each account (custody and reserve) and the amounts deducted so far from each account. This allows for an additional check that only the allocated amount can be deducted from the custody account in each payout and subtracted from the total custody.
- Use safe arithmetic operations that produce handleable errors rather than panic

- Add logic to payout the remaining balance of custody and reserve accounts at the completion of the custody period to prevent stuck funds.

Follow-up

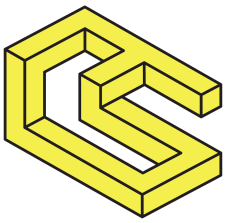
Commit 6fa5dcd adds a fix for this issue by calculating a value to subtract from the `TotalCustody` that accounts for the cases when it is the final payout and:

1. More is vested than allocated (e.g. balance was added to the custody or reserve accounts and is now being cleared out)
2. Less is vested than allocated (e.g. balance was subtracted from the custody account through proxy actions and payouts end prematurely)

It also counts the balances in the reserve accounts when initially calculating the `TotalCustody` and counts payouts from both accounts when updating.

This fixes the failing tests `payout_pays_out_additional_contributions_to_custody_account` and `payout_pays_out_additional_contributions_to_reserve_account.`

It should be noted that the `TotalCustody` variable does not reflect the total combined balances of the custody accounts if funds have been added to those accounts after initialization. It will only be updated on deductions but will be correctly deducted to zero when all accounts are empty. This is good for safety as it means the `TotalCustody` variable can only decrease and cannot be manipulated by coin holders by transferring into custody accounts.



Any deduction to custody accounts other than through payout results in invalid state

Identifier	CS-XX-04
Severity	Medium
Tests	`custody_set_proxy_can_call_function_to_mutate_custody_account_balance` `can_payout_if_custody_account_deducted`
Resolved	`Yes (commit` 6fa5dcd)

Team custody accounts can be used by custodians to make calls that are permitted by the `GovernenceProxy` type set in the pallet config. As the authorized calls can be set externally to the pallet, no guarantees can be made internally as to what side effects may occur on the custody account. Such side effects may include balance deductions that are not accounted for in the pallet logic.

If the balance of the team custody account is deducted below (`allocation - vested`) then the following erroneous effects will follow:

- The team member will never fully vest according to the `is_vested` function which keeps track of how much has been transferred out of the account by the `payout` function only.
- The value of `total_custody` will be incorrect and will never reach zero
- The given account is never properly cleared from the chain state

Recommendations

- To the reviewers knowledge there is no way to limit proxy access to an accounts balance, only the ability to make certain calls. Therefore it should be made very clear to developers using this pallet that they are responsible for ensuring that these calls are unable to trigger changes in account balances.

- `do_custody` should update the total custody based on the value of the balance on the account rather than by how much has been paid out. This should account for the fact that the balance of a custody (or reserve account) can increase or decrease through means other than allocation and vesting.

Follow-up

Commit 6fa5dcd adds a fix for this issue as described in the follow-up for CS-XX-03.

4.2. cMix Pallet

Overview

The cMix pallet serves as a public permissioned store for data relating to the cmix protocol component of the xx network. This includes:

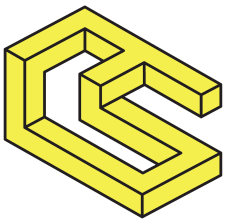
- The hashes of the current approved versions of the cmix software
- The account belonging to the cmix scheduling server
- The cmix address space byte
- The current and next-era cmix variables. This includes:
    - Points allocated or deducted for each success/failure in a mixing round
    - How countries map to geo bins
    - Points multipliers for each geo bin
    - team/batch sizes for each round
    - Number of registered users

It also exposes extrinsics to update cmix performance points and deductions accrued by each mix node in an era as determined by the scheduler. This is passed directly to the staking pallet and used in calculating era rewards.

A test suite covering all extrinsics was created as part of the review. It is included in the chainsafe/handover/tests-and-bench branch.

Issues

No issues were found with the cmix pallet and its integration with the staking pallet.



4.3. Economics Pallet

Overview

The economics pallet deals with:

- Managing the issuance and inflation of the xx coin
- Tracking the balance of the liquidity pool on Eth
- Managing the rewards pool account.

All extrinsics in this pallet are callable by admin origin only and are used to set system economic constants in the store. The pallet primarily operates by interfacing with the staking pallet and acting as the `EraPayout`, `Reward` and `RewardRemainder` handlers.

As Reward handler it is responsible for ensuring that the rewards pool is deducted when validator and nominator rewards are paid out up until the pool is depleted after which rewards will still be paid but will increase the total supply (inflation).

This pallet handles the `RewardRemainder` via its `RewardRemainderAdapter`. This handles the funds that result from the difference between the desired inflation and the validator/nominator rewards. Usually in this case the remainder would go directly to a treasury but the adaptor intercepts this event and ensures this is also taken from the rewards pool, if possible, and only minted once this is depleted. The imbalance is finally passed to the pallet's `RewardRemainder` handler.

The pallet also provides the inflation curve for the xx-substrate chain via its implementation of the `EraPayout` trait.

A test suite covering all extrinsics, `EraPayout` and `Reward`/`RewardRemainder` handlers was created as part of the review. It is included in the [chainsafe/handover/tests-and-bench](#) branch.

Issues

Evaluating ideal interest in first half of era results in panic

Identifier	CS-XX-05
Severity	Low
Tests	`get_era_payout_at_block_smaller_than_half_session` `get_era_payout_at_block_lower_than_first_point`
Resolved	`Yes (commit` 6fa5dcd)

`compute_ideal_interest` assumes that it is called with both `block > half_era_blocks` (which is defined on line 127 as half the length of an era) and that block is greater than the block number in `start`. Violating either of these assumptions will result in an attempted subtraction with overflow panic.

```
115  fn compute_ideal_interest(
116        block: T::BlockNumber,
117        start: IdealInterestPoint<T::BlockNumber>,
118        end: IdealInterestPoint<T::BlockNumber>) -> Perbill {
119        // Compute interest difference (start-end to ensure result is
     positive)
120        let diff = start.interest.clone().saturating_sub(end.interest);
121        // If difference is zero, must be constant part, take interest
     from start (or end)
122        if diff.is_zero() {
123        return start.interest
124        }
125        // Compute block ratio
126        let block_diff = end.block - start.block;
127        let half_era_blocks = Perbill::from_rational(1u32,2u32) *
     T::EraDuration::get();
128        let ratio = Perbill::from_rational(block - half_era_blocks,
```

```
    block_diff);
129         // Compute interest = start - diff*ratio
130         start.interest.saturating_sub(ratio * diff)
131     }
```

Under normal circumstances the staking pallet should never call `era_payout` such that it results in `get_ideal_interest` being called in the first half but this cannot be guaranteed.

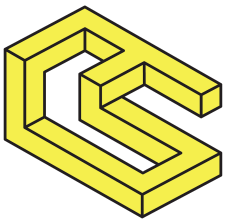
Recommendations

- Add logic to explicitly handle the case where `get_ideal_interest` is called with `block < half_era_blocks` or called with a block number less than the first point.

Follow-up

Following the recommendation commit 6fa5dcd adds an explicit check for the case where the block number is less than the length of half an era and uses the block number itself for interpolation rather than the midpoint in this case.

This resolves tests `get_era_payout_at_block_smaller_than_half_session` and `get_era_payout_at_block_lower_than_first_point`.



Unordered interest points can result in panic

Identifier	CS-XX-06
Severity	Medium
Tests	`get_era_payout_with_unordered_points` `get_era_payout_with_increasing_interest`
Resolved	`Yes (commit` 6fa5dcd)

The points that define the ideal interest curve can be set at any time by the admin origin. No checks are in place when setting the points and it is possible for the admin origin to set points which lead to an underflow panic when the staking pallet calls `era_payout` (line 126 in `inflation.rs`). This would result in a total failure of chain consensus.

```
115  fn compute_ideal_interest(
116        block: T::BlockNumber,
117        start: IdealInterestPoint<T::BlockNumber>,
118        end: IdealInterestPoint<T::BlockNumber>) -> Perbill {
119        // Compute interest difference (start-end to ensure result is
     positive)
120        let diff = start.interest.clone().saturating_sub(end.interest);
121        // If difference is zero, must be constant part, take interest
     from start (or end)
122        if diff.is_zero() {
123        return start.interest
124        }
125        // Compute block ratio
126        let block_diff = end.block - start.block;
127        let half_era_blocks = Perbill::from_rational(1u32,2u32) *
     T::EraDuration::get();
128        let ratio = Perbill::from_rational(block - half_era_blocks,
     block_diff);
```

```
129         // Compute interest = start - diff*ratio
130         start.interest.saturating_sub(ratio * diff)
131     }
```

It is also possible to set points which define a curve that has an increase in interest rate between points. This will not cause a panic but will not produce a value correctly interpolated between the points due to the `saturating_sub` in line 120. This case should also be prohibited.

The possible attacks are limited as the extrinsic to set the interest points can only be called by the admin origin.

Recommendations

- Add checks of the validity of the interest points when setting them from genesis config or via the extrinsic
- Use safe arithmetic operations (e.g. `saturating_sub`) where possible
- Use a `log::error!` in cases where requirements are not met rather than fail silently (e.g. line 120 if `start - end > 0`)

Follow-up

Commit 6fa5dcd adds a fix by

- Modifying the `compute_ideal_interest` function to support increasing as well as decreasing interest curves
- Adding a sort by increasing block number operation when setting the interest points both from genesis and via the `set_interest_points` extrinsic

This resolves test cases `get_era_payout_with_unordered_points` and `get_era_payout_with_increasing_interest`

5. Dependency Audit

The cargo-audit tool was used to detect known vulnerabilities in the dependencies of xx-substrate. 4 known vulnerabilities were found. Vulnerabilities are listed along with their RUSTSEC identifiers which can be used to find recommended fixes.

It is unknown if these vulnerabilities affect the security of xx-substrate.

Dependency	Vulnerability	Description
hyper=0.12.36 and 0.13.10	RUSTSEC-2021-0079	Integer overflow in parsing of the Transfer-Encoding header leads to data loss
	RUSTSEC-2021-0078	Lenient header parsing of Content-Length could allow request smuggling
libsecp256k1	RUSTSEC-2021-0076	libsecp256k1 allows overflowing signatures
prost-types	RUSTSEC-2021-0073	Conversion from prost_types::Timestamp to SystemTime can cause an overflow and panic

5.1. Recommendations

- RUSTSEC recommended fixes should be applied where possible.
- Cargo-audit should be added to CI so that the maintainers can be alerted if new vulnerabilities are found or introduced.