Quantstamp

Blocto Account For EIP-4337

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Smart Contract Wallet
Timeline	2023-06-12 through 2023-06-20
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• portto/4337-contracts #cfee161
Auditors	• Faycal Lalidji Senior Auditing Engineer • Michael Boyle Auditing Engineer • Shih-Hung Wang Auditing Engineer • Valerian Callens Senior Auditing Engineer

Documentation quality	High
Test quality	High
Total Findings	24 Fixed: 9 Acknowledged: 10 Mitigated: 5
High severity findings	2 Fixed: 1 Mitigated: 1
Medium severity findings	2 Fixed: 2
Low severity findings	14 Fixed: 3 Acknowledged: 7 Mitigated: 4
Undetermined severity findings	2 Fixed: 2
Informational findings	4 Fixed: 1 Acknowledged: 3

Summary of Findings

The assessment reveals a range of issues, including high-severity vulnerabilities, as well as medium, low, and minor issues. These findings require immediate attention to ensure the security and integrity of the Blocto smart contract wallet.

The identified issues include the missing Chain ID validation, insecure nonce validation, and inconsistent cosigner validation. These vulnerabilities pose significant risks, such as cross-chain replay attacks and potential manipulation of transaction ordering. It is crucial to address these issues promptly to protect users' funds and prevent unauthorized actions.

** Fix review update **

The Blocto team addressed all issues: more than half has been fixed or mitigated, and the rest was acknowledged. Also, the team significantly improved the quality of their test suite. However, special attention should be paid to the issue [BCT-2] where the nonce mechanism has been improved but different signers could still disrupt the sequence of transactions in the smart contract wallet.

ID	DESCRIPTION	SEVERITY	STATUS
BCT-1	Missing Chain ID Validation Allows Cross-Chain Replay Attack	● High	Fixed
BCT-2	Insecure Nonce Validation Allows Malicious Action in `CoreWallet` Contract	● High	Mitigated
BCT-3	Inconsistent Cosigner Validation in `CoreWallet.setAuthorized()`	● Medium	Fixed
BCT-4	`BloctoAccount` Implementation Cannot Be Updated	● Medium	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
BCT-5	Wallet Factory Is Incompatible with the `ERC-4337` Standard	● Low ⓘ	Acknowledged
BCT-6	Signature Malleability	● Low ⓘ	Fixed
BCT-7	Unchecked User Operation Hash Value	● Low ⓘ	Acknowledged
BCT-8	Privileged Roles and Ownership	● Low ⓘ	Acknowledged
BCT-9	Missing Deadline Checks on Signatures	● Low ⓘ	Acknowledged
BCT-10	Missing Storage Gaps in Upgradable Contracts	● Low ⓘ	Mitigated
BCT-11	Critical Role Transfer Not Following Two-Step Pattern	● Low ⓘ	Fixed
BCT-12	Associated Risk of Authorized Recovery Address Reset in `setRecoveryAddress()` and `emergencyRecover2()`	● Low ⓘ	Acknowledged
BCT-13	Missing Input Validation	● Low ⓘ	Mitigated
BCT-14	User Can Remove Blocto as Cosigner	● Low ⓘ	Fixed
BCT-15	User Can Remove the Recovery Key	● Low ⓘ	Mitigated
BCT-16	Users Can Arbitrarily Overwrite Merged Keys	● Low ⓘ	Acknowledged
BCT-17	Active Deposited Stake of Factory Can Be Lost	● Low ⓘ	Acknowledged
BCT-18	Initial Ownership Claimable by Anyone Both Directly and via a Proxy	● Low ⓘ	Mitigated
BCT-19	Unlocked Pragma	● Informational ⓘ	Fixed
BCT-20	Clone-and-Own	● Informational ⓘ	Acknowledged
BCT-21	Users Can Remove Themselves From Wallet	● Informational ⓘ	Acknowledged
BCT-22	Bundlers and Entrypoints Will Lose Trust in the Factory's Account if the Stake Is Withdrawn	● Informational ⓘ	Acknowledged
BCT-23	`Recovery` Address Can Be the Same as the `authorized` or `cosigner` Address in Function `init2()`	● Undetermined ⓘ	Fixed
BCT-24	Format of `mergedKeyIndex` Is Not Checked in the Function `setMergedKeys()`	● Undetermined ⓘ	Fixed

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer
Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
   1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Findings

BCT-1
Missing Chain ID Validation Allows Cross-Chain Replay Attack

• High ⓘ Fixed

Update
Marked as "Fixed" by the client. Addressed in: `b7bd8f27ddf8777979d2d909b9c4f527d78a8906` .

File(s) affected: `CoreWallet.sol`

Description: In `CoreWallet` contract, specifically in the `invoke1CosignerSends()` , `invoke1SignerSends()` , and `invoke2()` functions, the chain ID is not being properly validated, which creates a vulnerability for cross-chain replay attacks. Moreover, the utilization of the same `BloctoAccount` wallet address across multiple chains further increases the likelihood and the profitability of such attack.

Recommendation: To mitigate the risk of cross-chain replay attacks and protect users' funds, it is crucial to address the following by implementing chain ID validation to ensure that the chain ID matches the intended chain.

BCT-2
Insecure Nonce Validation Allows Malicious Action in `CoreWallet` Contract

• High ⓘ Mitigated

Alert
The team updated the mechanism and now the contract has a global nonce. However, one issue persists. The `invoke1CosignerSends()` function takes a parameter `inonce` , which needs to satisfy `inonce > nonce && (inonce < (nonce + 10)` . So, `nonce` can be increased by more than 1 by `inonce` . If `requiredCosigner == signer` , anyone can perform the front-running attack. Otherwise, only the cosigner (the Blocto server) can perform the attack.

Update
Marked as "Fixed" by the client. Addressed in: `71108a8a8ea58134606238f43fc7ba63863f67a8` .

File(s) affected: `CoreWallet.sol`

Description: Within `CoreWallet` contract, the user inputs nonce in the `invoke1CosignerSends()` and `invoke2()` functions. Currently, the validation only checks if the `nonce` is greater than the `nonces[signer]` value, without enforcing that the `nonce` should be incremented by one when compared against the `nonces[signer]` .

This issue introduces a potential security risk if the transactions get reordered by resubmitting the signatures through an attacker's address.

For example, in the case of a user submitting multiple transactions including `invoke2()` or `invoke1CosignerSends()` transactions, any user can obtain the transaction parameters and submit them independently using a transaction with a higher gas price, ensuring that it gets a higher chance to be submitted first. The outcome can be random and it will depend on the action executed through the submitted transactions.

Please note that for `invoke1CosignerSends()`, this issue is applicable only when `requiredCosigner == signer`.

Recommendation: To enhance security and prevent malicious actions, it is recommended to use a single nonce for ordering transaction execution, even when different keys of the same user are used. This ensures consistent transaction ordering and mitigates the risk of transaction ordering manipulation.

BCT-3 Inconsistent Cosigner Validation in `CoreWallet.setAuthorized()`

● Medium ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `71108a8a8ea58134606238f43fc7ba63863f67a8` . The client provided the following explanation:

> The documentation has been updated, with each authorized key (device) now corresponding to a cosigner key. https://www.notion.so/portto/Blocto-Smart-Contract-Wallet-d4133925374c4b6ba6943409bf0cc55a

File(s) affected: `CoreWallet.sol`

Description: The `CoreWallet.setAuthorized()` function allows the configuration of a new pair of authorized key and cosigner. However, the current implementation does not enforce consistency between the newly provided cosigner and the previously set cosigner, contrary to the specification mentioned here. According to the specification, there should only be one cosigner address associated with the wallet.

Recommendation: Update `CoreWallet.setAuthorized()` to enforce consistency between the newly provided cosigner and the previously set cosigner address, as specified in the documentation.

BCT-4 `BloctoAccount` Implementation Cannot Be Updated

● Medium ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `211c5ead83c26e84f43f5cdf67f784178cc69fff` . The client provided the following explanation:

> We delete the ret.disableInitImplementation(); and use initImplementation.

File(s) affected: `BloctoAccountFactory.sol`

Description: In the `BloctoAccountFactory` contract, the state variables `initImplementation` and `bloctoAccountImplementation` should store the implementation address of `BloctoAccountCloneableWallet` . However, only `initImplementation` is used in the `BloctoAccountFactory.createAccount()` and `BloctoAccountFactory.createAccount2()` functions, preventing the Blocto account update features from functioning properly. The `BloctoAccountFactory.setImplementation()` function only changes the `bloctoAccountImplementation` address, resulting of the contract deploying the initial version of the `BloctoAccount` contract. Additionally, `BloctoAccountFactory.getAddress()` computes user wallet accounts using the `initImplementation` value, which is the initial implementation of `BloctoAccountCloneableWallet` set during factory deployment.

Recommendation: Following the description above the implemented code in `BloctoAccountFactory` is not production ready, we recommend carefully refactoring the factory while respecting all the project specifications.

BCT-5 Wallet Factory Is Incompatible with the `ERC-4337` Standard

● Low ⓘ Acknowledged

Update

After considering the comment of the client, we decreased the severity level from Medium to Low.

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> The functions createAccount() and createAccount2() allow the addition of merged Schnorr keys, but they are not appropriate for creating an account from Entrypoint due to the security issue. The security issue is that contract doens't check the merged key combination from authorized and cosigner because the gas usage is too high. To address this security issue, it is essential to require the grant role for creating an account instead. The severity level may be reduced since account creation is now controlled solely by the Blocto server, enhancing security compared to allowing anyone to create an account from the entrypoint.

File(s) affected: `BloctoAccountFactory.sol`

Description: According to the protocol documentation, the Blocto backend server calls the `BloctoAccountFactory.createAccount()` or `createAccount2()` functions to create Blocto wallets. However, according to the implementation and code comments of `BloctoAccountFactory` , it seems it is intended to allow users to create wallets through the ERC-4337 flow.

In fact, the `BloctoAccountFactory` contract is incompatible with the ERC-4337 standard since the `createAccount()` and `createAccount2()` functions utilize `onlyOwner` modifier, which allows only the owner (i.e., the Blocto server) to call them. Since the `Entrypoint` cannot call these functions, users cannot create wallets by sending user operations using the ERC-4337 architecture.

Recommendation: Please confirm whether users are permitted to create contracts using the ERC-4337 flow. It should be noted that both create account functions utilize input parameters as salt but omit the authorized key. Removing the `onlyOwner()` modifier can potentially expose the system to front-running attacks. For example, in the case of `createAccount2()` , an attacker could submit a transaction while adding an additional authorized key, thereby gaining access to the wallet while keeping the `CREATE2` generated address unchanged.

BCT-6 Signature Malleability

• Low ⓘ Fixed

Update
Marked as "Fixed" by the client. Addressed in: `a912e849054de3e02cdd3061e7b0b9b7dfde6365` .

File(s) affected: `CoreWallet.sol`

Description: Signature malleability is a characteristic of the ECDSA (Elliptic Curve Digital Signature Algorithm) cryptographic algorithm used in Ethereum and Solidity. It refers to the ability to produce different valid signatures for the same message by modifying the signature parameters. This can occur due to certain mathematical properties of the ECDSA algorithm.

In the context of Solidity, signature malleability can have security implications, particularly when signatures are used for authentication, verification, or as proof of authorization. Exploiting signature malleability can potentially lead to unintended behaviors or vulnerabilities in smart contracts.

One common form of signature malleability in Solidity's ECDSA is the "s-value malleability" In an ECDSA signature, there are three components: the "r", "s" value and the "v". The "s" value is the one susceptible to malleability. Any modification to the "s" value while keeping the signature valid can result in a different but still valid signature.

`CoreWallet` contract in multiple functions lacks ECDSA signature malleability validation.

Recommendation: To ensure the integrity and security of the system, it is recommended to implement the validation of ECDSA signature values in all related functions within the `CoreWallet` contract. This can be achieved by following the guidelines provided in OZ reference implementation here

BCT-7 Unchecked User Operation Hash Value

• Low ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> The 'Operation Hash' is retrieved from the entrypoint and is calculated directly within the entrypoint. This process includes a double-check, which results in an additional cost of 1711 gas usage is not necessary. Only allow whitelisted entrypoint to use BloctoAccount. Gas 1711 refer: https://github.com/blocto/4337-contracts/commit/3d86e7a0306b0dc5d397c7d3c4993cac7fcb85b9

File(s) affected: `BlocktoAccount.sol`

Description: The `BlocktoAccount` contract relies on the user operation hash, which is computed by the `entryPoint` and passed as an input parameter to `BlocktoAccount._validateSignature()` . However, the contract does not currently validate this hash. Considering the possibility of the `_entryPoint` being compromised, we recommend implementing hash validation in the `BloctoAccount` contract wallet to ensure the integrity of the user operation.

However, the `entryPoint` has direct access to users' `BloctoAccount` wallets through the `execute()` and `executeBatch()` functions, which allows the `_entryPoint` to execute calls from users' wallets without any restrictions. This poses a risk in case the `entryPoint` is compromised. It is important to note that all Account Abstraction contracts are not within the scope of this audit. Furthermore, developers should exercise caution when implementing such functionality.

Recommendation: To enhance security, we recommend calculating the hash value using the `userOp` input.

BCT-8 Privileged Roles and Ownership

● Low ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> Adding access control to Account Factory, not only owenr can control the factory but only the admin from Blocto. Because the contract doesn't check merged key which is combined from authorized and cosigner, the gas usage too high. To address this, a grant is required to create an account.

File(s) affected: `BloctoAccountFactory.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.

The `BloctoAccountFactory` contract has the following owner privileges:

1. The owner can reset the `BloctoAccountCloneableWallet` implementation address.
2. The owner can update the `_entryPoint` address using `BloctoAccountFactory.setEntrypoint()`.
3. The owner can renounce the ownership of the factory contract. As a result, no one can create wallets anymore since the `onlyOwner` check reverts the transaction.

A malicious owner (in case of an EOA hack) could make the contract inoperable in multiple ways such as not providing enough stake or providing incorrect data when creating a new account. Most notably, a user could send funds to a wallet that has not been deployed yet and the owner could refuse to deploy the account, locking the funds from the user.

Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

BCT-9 Missing Deadline Checks on Signatures

● Low ⓘ Acknowledged

Alert
The deadline is not used to order signed transactions like the nonce, but to avoid a transaction being executed after a given period.

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> It checks by nonce, may not be necessary check by deadline. In Blocto mode, using invoke(), CoreWallet will hash nonce in invoke() series functions. In 4337 mode, using execute(), UserOpeartion including nonce and hash it.

File(s) affected: `CoreWallet.sol`

Description: The `invoke1CosignerSends()`, `invoke1SignerSends()`, and `invoke2()` functions validate signatures from the signer or cosigner (or both) against an operation hash. If the signatures are valid, the corresponding action is invoked. However, these signatures are not restricted by a deadline. Therefore, a malicious actor (e.g., the cosigner or a block producer) could delay the execution of the transaction but execute it after some time if they would benefit from it.

According to the ERC-4337 specification, when validating a user operation's signature, the wallet can return a packed value of `authorizer`, `validUntil`, and `validAfter` timestamps. Such a design intends to restrict the valid time range of a signature. However, no specific time-range is returned when implementing the `BloctoAccount._validateSignature()` function. As a result, a malicious bundler may be able to delay sending the user operation if they could potentially benefit from doing it.

Recommendation:
Consider including a `deadline` field in the signed data and ensuring the deadline has not passed when the `invoke()` functions are called. Consider returning a time range from the `_validateSignature()` function to indicate the valid time for the user operation.

BCT-10 Missing Storage Gaps in Upgradable Contracts

• Low ⓘ Mitigated

Alert
Only the contracts `BloctoAccount.sol` and `BloctoAccountFactory.sol` have been updated.

Update
Marked as "Fixed" by the client. Addressed in: `250c2e5f458b7b9420bea0af4b5bf2757e6c1f73` .

File(s) affected: `CoreWallet.sol` , `BloctoAccount.sol` , `BloctoAccountFactory.sol`

Description: The following upgradable contracts do not include a `__gap` state variable to preserve storage slots for future updates. Without the storage gap, newly added state variable may overwrite existing ones in the inherited child contracts and causes incorrect results.

Notice that the `BaseAccount` contract inherited by `BloctoAccount` is not designed as an upgradable contract. If further modifications to the `BaseAccount` contract introduce new state variables, the current state variables declared in the `BaseAccount` contract will be overwritten.

Recommendation: Consider adding a storage gap to these contracts to avoid state variable overwrites when introducing new state variables to the upgraded contracts.

To avoid the issue related to `BaseAccount` , consider writing a wrapper contract of `BaseAccount` , which allocates additional storage gaps. Whenever the upstream `BaseAccount` contract inserts new state variables, reduce the storage gap in the wrapper contract to maintain storage consistency.

BCT-11 Critical Role Transfer Not Following Two-Step Pattern

• Low ⓘ Fixed

Update
The original issue no longer exists since the contract does not inherit from `Ownable` . However, using `AccessControlUpgradeable` instead introduces new centralization risks. Now, there are two roles: `CREATE_ACCOUNT_ROLE` and `DEFAULT_ADMIN_ROLE` . New privileged roles and centralization risks: 1) `DEFAULT_ADMIN_ROLE` can:

- Update the wallet implementation.
- Grant or revoke any address with the `DEFAULT_ADMIN_ROLE` or `CREATE_ACCOUNT_ROLE` role.
- Renounce their role so that no one can update the implementation.
  2) CREATE_ACCOUNT_ROLE can create new wallets.

Update
Marked as "Fixed" by the client. Addressed in: `7a42737f65a15ceef7a8a0298025adc0122826bc` . The client provided the following explanation:

> The BloctoAccountFactory's upgrade from OwnableUpgradeable to AccessControlUpgradeable streamlines the process by eliminating the need for two steps. AccessControlUpgradeable offers additional functionality, including grantRole() and revokeRole(), surpassing OwnableUpgradeable, which only provides transferOwnership().

File(s) affected: `BloctoAccountFactory.sol`

Description: The owner of the contracts can call `transferOwnership()` to transfer the ownership to a new address. If the new owner's address is provided incorrectly as an uncontrollable address including zero address, the contract loses the ownership. No one can execute any function with the `onlyOwner` modifier anymore.

Recommendation: Consider using OpenZeppelin's `Ownable2StepUpgradeable` contract to adopt a two-step ownership pattern to prevent this issue.

BCT-12 Associated Risk of Authorized Recovery Address Reset in `setRecoveryAddress()` and `emergencyRecover2()`

• Low ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> The assessment of this issue will be carried out in the next version because it's not a straightforward matter.

File(s) affected: `CoreWallet.sol`

Description:

- The current implementation of the `setRecoveryAddress()` function allows resetting the recovery address using any authorized address that can pass the co-signing process. However, granting the ability to change the recovery address to any authorized address poses a significant risk. If any of the user's devices are compromised or stolen, an attacker could overwrite all other authorized addresses by resetting the recovery address and executing an emergency recovery operation.
- If a user calls `emergencyRecover2()` and changes the recovery address, deploying the account to another chain could result in a new address if the new recovery address is used as part of the salt. Alternatively, maintaining the same address would result in using a stale recovery address.

Recommendation:

- Enhance the security of the system by restricting the ability to reset the recovery address to a more limited set of trusted addresses or require more than a single device.
- Consider if removing the recovery key as a part of the salt can be beneficial for the project design.

BCT-13 Missing Input Validation

• Low ⓘ Mitigated

Update

- `BloctoAccount.sol` : all is fixed
- `BlocToAccountFactory.sol` : all is fixed except hardcoding the `entryPoint` . I think it's fine, they considered it, not required to do it.
- `CoreWallet.sol` : The only fix is in `init()` and `init2()` , where `cosigner` address is checked.

Update

Marked as "Fixed" by the client. Addressed in: `26acf4f9d925757c3426648d1df1faf2487c1577` .

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. Specifically:

1- `BloctoAccount`

- `executeBatch()`
  - Require the length of `value[]` to match the lengths of `dest[]` and `func[]`.

2- `BloctoAccountFactory`

- `initialize()`
  - Require `_bloctoAccountImplementation` to be non zero.
  - Consider hardcoding the known `entryPoint` address.
- `setEntryPoint()`
  - Require `_entrypoint` to be non zero.
- `setImplementation()`
  - Require `_bloctoAccountImplementation` to be non zero.
- `withdrawTo()`
  - Require `withdrawAddress` to be non zero.
  - Require `amount` to be greater than zero.

3- `CoreWallet`

- `init()`
  - Require `_cosigner` to be non zero.
  - Require `_mergedKeyIndexWithParity` to be non zero.
  - Require `_mergedKey` to be non zero.
- `init2()`
  - Require `_cosigner` to be non zero.
  - Require all `_mergedKeyIndexWithParitys` to be non zero.
  - Require all `_mergedKeys` to be non zero.
- `emergencyRecovery2()`
  - Require `_recoveryAddress` to be different than cosigner address.
- `setRecoveryAddress()`
  - Require `_recoveryAddress` to be different than cosigner.

Recommendation: We recommend adding the relevant checks.

BCT-14 User Can Remove Blocto as Cosigner

• Low ⓘ Fixed

Update
Marked as "Fixed" by the client. Addressed in: `71108a8a8ea58134606238f43fc7ba63863f67a8`. The client provided the following explanation:

> Add some description to documentation. https://www.notion.so/portto/Blocto-Smart-Contract-Wallet-d4133925374c4b6ba6943409bf0cc55a

File(s) affected: `CoreWallet.sol`

Description: Given that a user has access to the recovery key, the user can call `emergencyRecovery()` and assign a new authorized account and cosigner. The new authorized account could be two accounts owned by the user, making the wallet continue as a multi-sig. However, the two accounts can also be the same. This would effectively make their account a smart contract wallet extension of the EOA linked. If the mobile application does not support sending transactions that are not cosigned by Blocto, users may not be able to use the app.

Recommendation: Consider making this known to users via documentation.

BCT-15 User Can Remove the Recovery Key

• Low ⓘ Mitigated

Update
A check was added to make sure that the new `recoveryKey` is non-zero. However, a two-step mechanism was not added, and the user can add a `recoveryKey` of an address owned by no one and the result will be the same.

Update
Marked as "Fixed" by the client. Addressed in: `cb3dbabd9000f86ee7fe37b5c6a8288b2731759c`.

File(s) affected: `CoreWallet.sol`

Description: It is possible to set the recovery key to the zero address. The original assignment of the recovery key comes from Blocto as they own the factory contract. However, a user in custody of the recovery key can call `emergencyRecovery2()` and set the new recovery key to the zero address.

Recommendation: Consider making this a two-step process where `emergencyRecover2()` requires the address to be nonzero and a new function revokes the recovery key by setting it to zero. If this is intended behavior, make it known to users via documentation.

BCT-16 Users Can Arbitrarily Overwrite Merged Keys

• Low ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> 1. Merging keys on-chain becomes prohibitive due to the excessively high gas usage.
> 2. Since the merged key may be revoked, we can set it to 0 without checking if it is already 0.
> 3. The merged key challenge mechanism will be added in future work.

File(s) affected: `CoreWallet.sol`

Description: The `setMergedKey()` function does not check for an existing key at a particular `mergedKeyIndex`. Therefore, a working key can be overridden and render Schnorr signatures useless.

Recommendation: Consider checking that the signature is nonzero before assigning the new value.

BCT-17 Active Deposited Stake of Factory Can Be Lost

• Low ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation:

> Due to the implementation of access control in the factory, creating an account from the entrypoint is no longer possible. Consequently, there is no need to add stake to the entrypoint.

File(s) affected: `BlocToAccountFactory.sol`

Description: The owner of the contract `BlocToAccountFactory.sol` has permission to add to or withdraw a stake from a given entry point. He can also modify the current entry point used by the factory. Still, if there is an active amount of stake deposited in an entry point and the owner renounces his role, it will not be possible anymore to withdraw that stake from the previous entry point.

Recommendation: If the ownership can be renounced (or transferred to an address not owned by anyone), consider making sure that there is no active deposited stake before executing that operation. If the ownership cannot be renounced, there is no issue because the owner can reassign the specific entryPoint by calling the function `setEntryPoint()`.

BCT-18
Initial Ownership Claimable by Anyone Both Directly and via a Proxy

• Low ⓘ Mitigated

Update

The contract `TransparentUpgradeableProxy` is used to initialize the implementation from the proxy's constructor. However, no detail was provided about how the function `initialize()` of the implementation contract is called.

Update

Marked as "Fixed" by the client. Addressed in: `23fb8d19a8f15483397f1e7097b9ac13c9a68699`. The client provided the following explanation:

> We follow OpenZepplin upgrades pattern using upgrades.deployProxy(). Like the BCT-18 recommendation, it calling the function initialize() from the proxy when setting the new implementation contract. refer: https://docs.openzeppelin.com/upgrades-plugins/1.x/platform-deploy Detail of implementation of OpenZepplin upgrades:
> 1. change proxy admin: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.2.0/contracts/proxy/transparent/TransparentUpgradeableProxy.sol#L39
> 2. use ERC1967Proxy(_logic, _data) data
> 3. the constructor data of ERC1967Proxy will call initialize() in BloctoAccountFactory for initializing BloctoAccountFactory's admin and init data. see https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.2.0/contracts/proxy/ERC1967/ERC1967Proxy.sol#L23

```
_upgradeToAndCall(_logic, _data, false);
```

File(s) affected: `BloctoAccountFactory.sol`

Description: As the contract `BloctoAccountFactory.sol` is expected to be run as an implementation contract following a proxy-implementation pattern, a function `initialize()` exists to play the role of the constructor and can be called a single time. The drawback is that, once deployed, anyone can call that function and will get ownership of the contract. Note that this function can be called both directly and via the proxy.

Recommendation: Consider making sure that the function to initialize the contract is not called by anyone else than the team:
- for the direct way, consider calling the function from a constructor with values that will make the contract inoperable.
- for the way via the proxy, consider calling the function `initialize()` from the proxy when setting the new implementation contract.

BCT-19 Unlocked Pragma

• Informational ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `a71f01224843e67a4975b88720b6565984f9b8ba`. The client provided the following explanation:

> ^0.8.12 → 0.8.17, note: cannot be 0.8.20, because 0.8.20 have new PUSH0 op, may some blockchain not support

File(s) affected: `All files`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.*.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

BCT-20 Clone-and-Own

• Informational ⓘ Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > Due to modify lot of code in CoreWallet, it may not use package manager. But like `BaseAccount` from 4337 official and upgradeable contract from OpenZeppelin we use package manager.

File(s) affected: `CoreWallet.sol`, `BytesExtractSignature.sol`, `BloctoAccountFactory.sol`

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries.

Recommendation: Rather than the clone-and-own approach, good industry practice is to use a package manager (e.g., npm) for handling library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries. If the file is cloned anyway, a comment including the repository, commit hash of the version cloned, and the summary of modifications (if any) should be added. This helps to improve the traceability of the file.

BCT-21 Users Can Remove Themselves From Wallet

• Informational ⓘ Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > In non-custodial mode, user can remove cosigner.

File(s) affected: `CoreWallet.sol`

Description: A user can call `setAuthorized()` to remove their own authorization by setting the associated cosigner to the zero address. The account can still be recovered via the recovery key.

Recommendation: Consider allowing the removal of an authorized key only if multiple keys are still set within the wallet contract. However, if only one key is remaining, it should only be possible to reset it using another valid key.

BCT-22 Bundlers and Entrypoints Will Lose Trust in the Factory's Account if the Stake Is Withdrawn

• Informational ⓘ Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > Same as BCT-17. For security reason, we don't create account from Entrypoint.

File(s) affected: `BlocToAccountFactory.sol`

Description: The owner has the right to withdraw stakes associated with the contract `BlocToAccountFactory`. If it does completely, the `entryPoint` or the bundlers (ERC-4337) will not trust any more contracts created with that factory, which would impact the already deployed `bloctoAccount`s.

Recommendation: Consider documenting that aspect in the user-facing documentation.

BCT-23 `Recovery` Address Can Be the Same as the `authorized` or `cosigner` Address in Function `init2()`

• Undetermined ⓘ Fixed

Update
The issue is fixed, but the check `require(address(uint160(_cosigner)) != _recoveryAddress, "Do not use the recovery address as a cosigner.");` could be performed outside of the loop instead of inside, because there are only one cosigner and one recovery address parameters, resulting in redundant checks.

Update
Marked as "Fixed" by the client. Addressed in: `801fa6af820cdbeafbb61e11b04aea55860c70d3` .

File(s) affected: `CoreWallet.sol`

Description: In the function `init()` , checks make sure that it is not possible for the `authorized` address to also be the `recovery` address. Same for the `cosigner` address. However, these checks are not found in the function `init2()` . As a result, it is possible in the latter to set a `recovery` address that is also an `authorized` and/or `cosigner` address.

Recommendation: Consider fixing this inconsistency or documenting it in the NatSpec of the function `init2()` .

BCT-24
Format of `mergedKeyIndex` Is Not Checked in the Function `setMergedKeys()`

● Undetermined ⓘ Fixed

Update
Marked as "Fixed" by the client. Addressed in: `dbfb4c8d91692eff36be13b3c1e1ec43a5944b4a` .

File(s) affected: `CoreWallet.sol`

Description: It is possible in the function `setMergedKey()` to assign a new `mergedKey` for a particular `mergedKeyIndex` . However, the format of keys for the mapping `mergedKeys` is `(authVersion,96)(padding_0,152)(isSchnorr,1) (authKeyIdx,6)(parity,1)` and the function `setMergedKey()` takes a uint256 value for the parameter `mergedKeyIndex` . As a result, invalid authVersions can be used if `mergedKeyIndex` can be represented by a bytes32 value where the `96` most-significant bytes are different than `0` . Also, the value of the bytes used for `isSchnorr` and `parity` is not checked.

Recommendation:
1. replacing the type of the parameter `mergedKeyIndex` from `uint256` to `uint8` .
2. checking the value of the bytes used for `isSchnorr` and `parity` .

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Code Documentation

- BloctoAccount.sol#L24 typo "etnrypoint" → "entrypoint"
- BloctoAccount.sol#L94 typo "etnrypoint" → "entrypoint"
- BloctoAccount.sol#L136 The clone functionality is not described above and the init function is below this comment.

- CoreWallet.sol contract description has not been updated to include EIP 4337 functionality.
- CoreWallet.sol#L62 describes the upper 12 bytes of `authorizations` as "reserved for future meta-data purposes". Consider changing the docs to reflect its use of storing the `authVersion`.
- In `BloctoAccount.sol` a comment should explain why the function `_authorizeUpgrade()` only contains the line `(newImplementation);`.
- In `BloctoAccountFactory.sol` the name `full` of the third field of the event `WalletCreated` is not explicit enough.
- In `CoreWallet.sol`, there are typos where `mereged` is used instead of `merged`.

Adherence to Best Practices

1. The `BloctoAccountFactory.WalletCreated` event emitted in the `BloctoAccountFactory.createAccount2()` function should include the complete list of authorized addresses. Currently, it does not provide this information, which can be useful for tracking and auditing purposes.
2. Remove unused functions: `calculateParity()` in `CoreWallet.sol`
3. `CoreWallet.recoverGas()` should use `0xffffffffffffffffffffffff` as the max of `_version`. In the 32 byte word, 20 bytes are used by the address and 12 bytes can be used for the version. The max value for 12 bytes is `0xffffffffffffffffffffffff`. Remove the `TODO` comment.
4. Modifiers should be before functions in `BloctoAccount.onlyOnceInitImplementation()`.
5. In all contracts: it is possible to reduce the size of the contract and ultimately the amount of gas to pay at deployment by limiting the number of characters in the error message of the require statements, or by using custom errors.
6. In `BloctoAccountProxy.sol`, the slot number where the implementation is stored is used twice and could be recorded in a single constant variable.
7. In `BloctoAccountFactory.sol`:
   - the `bytes32` casting of the local variable `bytes32 salt` in the function `getAddress()` is redundant.
   - the following portion of code is used three times in the contract and could be factorized in a single pure function: `bytes32 salt = keccak256(abi.encodePacked(_salt, _cosigner, _recoveryAddress));`

1. In `CoreWallet.sol`:
   - the storage variable `authVersion` is assigned to the constant value `AUTH_VERSION_INCREMENTOR` only once in the functions `init()` and `init2()`. As a result, this assignment could instead directly be done when the storage variable `authVersion` is declared.
   - the comparison operator `> 0` used in the first require statement of the function `init2()` can be replaced by the operator `!=` to save gas.
   - there is a `// TODO` instruction in the function `recoverGas()` which suggests that the code is not yet ready to be deployed.
   - the function `calculateParity()` is defined but never used. Also, its behavior is not described by a natSpec.
   - the statement `address(uint160(authorizations[authVersion + uint256(uint160(msg.sender))]))` could be factorized in a single function.

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

- `7a5...fdb ./contracts/BloctoAccountFactory.sol`
- `f27...bbd ./contracts/BloctoAccountProxy.sol`
- `737...801 ./contracts/BloctoAccountCloneableWallet.sol`
- `333...cad ./contracts/TokenCallbackHandler.sol`
- `7b8...d25 ./contracts/BloctoAccount.sol`
- `638...a7b ./contracts/CoreWallet/CoreWallet.sol`
- `5f0...6f8 ./contracts/CoreWallet/BytesExtractSignature.sol`

Tests

- `af0...ad7 ./test/bloctoaccount.test.ts`
- `41e...235 ./test/schnorrUtils.ts`
- `76e...486 ./test/schnorrMultiSign.test.ts`
- `039...618 ./test/testutils.ts`
- `fae...367 ./test/entrypoint/entrypoint.test.ts`
- `059...c63 ./test/entrypoint/entrypoint_utils.ts`
- `d03...75d ./test/entrypoint/aa.init.ts`

- `0dc...92a ./test/entrypoint/chaiHelper.ts`
- `677...53e ./test/entrypoint/debugTx.ts`
- `5a3...aad ./test/entrypoint/UserOp.ts`
- `384...2c2 ./test/entrypoint/solidityTypes.ts`
- `236...d91 ./test/entrypoint/UserOperation.ts`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:
- Slither v0.9.3

Steps taken to run the tools:
1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

Slither did not raise any significant findings.

Test Suite Results

Fix Review Update

To run the tests:
- Go to `hardhat.config.ts` and replace the rpc url for Ethereum:
- run `yarn` and `yarn test`
- run `npx hardhat test test/schnorrMultiSign.test.ts`

```
  BloctoAccount Upgrade Test
    wallet function
createAccount gasUsed:  BigNumber 223095
      ✔ should receive native token (73ms)
createAccount with multiple authorized address gasUsed:  BigNumber 275350
      ✔ should create account with multiple authorized address (50ms)
    should upgrade account to different version implementation
createAccount gasUsed:  BigNumber 223107
      ✔ new factory get new version and same acccount address (83ms)
      ✔ upgrade fail if not by contract self
      ✔ upgrade test (193ms)
      ✔ factory getAddress sould be same
createAccount gasUsed:  BigNumber 228703
      ✔ new account get new version (49ms)
      ✔ should entrypoint be v070 address
    should upgrade factory to different version implementation
      ✔ new factory get new version but same acccount address (56ms)

  EntryPoint
node version: HardhatNetwork/2.12.4/@nomicfoundation/ethereumjs-vm/6.0.0
createAccount gasUsed:  BigNumber 223095
    #simulateValidation
createAccount gasUsed:  BigNumber 223095
      ✔ should fail if validateUserOp fails
      ✔ should report signature failure without revert
      ✔ should revert if wallet not deployed (and no initcode)
      ✔ should revert on oog if not enough verificationGas
      ✔ should succeed if validateUserOp succeeds (44ms)
      ✔ should return empty context if no paymaster
      ✔ should prevent overflows: fail if any numeric value is more than 120 bits
createAccount gasUsed:  BigNumber 223083
      ✔ should not call initCode from entrypoint (46ms)

  Schnorr MultiSign Test
createAccount gasUsed:  BigNumber 223095
    ✔ should generate a schnorr musig2 and validate it on the blockchain (41ms)
createAccount gasUsed:  BigNumber 223107
    ✔ check none zero mergedKeyIndex


  19 passing (2s)


**Fix review update**


  BloctoAccount Upgrade Test
    ✔ creat previous version account (61ms)
    ✔ should not deploy again with create3
    ✔ should delpoy new cloneable wallet and upgrade factory  (445ms)
    ✔ should upgrade by account (111ms)
    wallet functions
      ✔ should not init account again (63ms)
      ✔ should not init2 account again (61ms)
      ✔ should not initImplementation again (58ms)
      ✔ should initImplementation with a contract (40ms)
      ✔ should receive native token (78ms)
      ✔ should receive 0 native token (77ms)
      ✔ should send ERC20 token (227ms)
      ✔ should revert if invalid data length (170ms)
      ✔ should create account with multiple authorized address
    should upgrade account to different implementation version
      ✔ new factory get new version and same account address (80ms)
      ✔ upgrade fail if not by contract self
      ✔ upgrade test (104ms)
      ✔ factory getAddress sould be same
      ✔ new account get new version (53ms)
      ✔ should entrypoint be v070 address
    4337 functions
      ✔ should execute transfer ERC20 from entrypoint (277ms)
      ✔ should revert execute transfer ERC20 from entrypoint with error signature (51ms)
      ✔ should execute approve ERC20 from entrypoint (257ms)
      ✔ should revert execute transfer ERC20 from entrypoint (73ms)
      ✔ should executeBatch transfer ERC20 from entrypoint (288ms)
      ✔ should deposit & getDeposit by anyone (79ms)
      ✔ should withdraw deposit (333ms)
      ✔ should revert withdraw deposit if call from other address
    should upgrade factory to different version implementation
      ✔ new factory get new version but same account address (45ms)
    factory functions
      ✔ should implementation not be zero address
      ✔ should init factory (52ms)
      ✔ should not initiate again
      ✔ should revert if sender is not grant role for createAccount
      ✔ should revert if sender is not grant role for createAccount2
      ✔ should revert if sender is not grant role for setImplementation
      ✔ should revert if setImplementation with zero address
    should create account if account has create account role
      ✔ shoule crate account with grant role (138ms)
    EOA entrypoint for _call fail test
      ✔ should revert for execute non exist function
      ✔ should revert for execute batch with wrong array length
      ✔ should revert for execute batch with wrong array length 2

  BloctoAccount CoreWallet Test
    emergency recovery performed - emergencyRecovery
      ✔ should not be able to emergencyRecovery with wrong key
      ✔ should be able to emergencyRecovery (53ms)
      ✔ backup key is different (check new authorized & cosigner)
      ✔ should be able to perform transactions with backup key (send ERC20) (123ms)
      ✔ should not be able to perform transactions with old key
      ✔ should see that the auth version has incremented
      ✔ should be able to recover gas for previous version (68ms)
      ✔ should not authorized with zero address
      ✔ should not authorized same as recovery
      ✔ should not init cosigner is zero address
    emergency recovery 2 performed - emergencyRecovery2
      ✔ should not perform emergencyRecovery2 if wrong key
      ✔ backup key is different (check new authorized & cosigner)
      ✔ should be able to perform transactions with backup key (send ERC20) (124ms)
      ✔ should not be able to perform transactions with old key
      ✔ should see that the auth version has incremented
      ✔ should be able to set a new recovery address (94ms)
      ✔ should not authorized with zero address
      ✔ should not authorized same as recovery
      ✔ should not cosigner same as recovery
      ✔ should not cosigner is zero address
      ✔ should not recovery is zero address
    recoverGas function
      ✔ should not be able to recover gas for wrong version
      ✔ should not be able to recover gas for now version
      ✔ should be able to recover gas for previous version (47ms)
    setRecoveryAddress function
      ✔ should not be able to set a new recovery address by wrong key
      ✔ should not be able to directly call setRecoveryAddress
      ✔ should not recovery address be zero address
      ✔ should not use an authorized address as the recovery address
      ✔ should not be able to directly set a new recovery address
      ✔ should be able to set a new recovery address (97ms)
    authorized wallet send tx
      ✔ should not be able to use invoke0
      ✔ should be able to perform transactions with authorized key (send ERC20) (126ms)
      isValidSignature test
        ✔ shoule return 0 for wrong authorized signature
        ✔ shoule revert if too big authorized signature
        ✔ shoule return 0 for wrong authorized with cosigner signature
        ✔ shoule revert if too big authorized signature - 2 signature
        ✔ shoule revert if too big cosigner signature - 2 signature
        ✔ shoule return 0 for wrong signature length
        ✔ shoule return 0 for none zero authorized but zero for cosigner
    authorized wallet same as cosigner wallet send tx
      ✔ should be able to perform transactions with authorized key (send ERC20) (88ms)
      ✔ should be able to receive native token
    wallet delegate function
      ✔ should not be able to delegate function with wrong key
      ✔ should not be able to directly call delegate function
      ✔ should not be able to delegate to COMPOSITE_PLACEHOLDER (100ms)
      ✔ should be able to delegate function with payable (199ms)
    init function test
      ✔ should not init authorized with zero address
      ✔ should not init authorized same as recovery
      ✔ should not init cosigner same as recovery
      ✔ should not init cosigner is zero address
    init2 function test
      ✔ should not init2 authorized zero length array
      ✔ should not init2 array length fail
      ✔ should not init2 array length fail 2
      ✔ should not init2 cosigner address be zero
      ✔ should not init2 authorized address be zero
      ✔ should not init2 use the recovery address as an cosigner address
      ✔ should not init2 use the recovery address as an authorized address
    invoke1CosignerSends function
      ✔ should revert if v of signature is invalid
      ✔ should revert if s of signature is invalid
      ✔ should revert if signature is invalid
      ✔ should revert if nonce is invalid
      ✔ should revert if not authorized addresses must be equal
    invoke1SignerSends function
      ✔ should revert if v of signature is invalid
      ✔ should revert if s of signature is invalid
      ✔ should revert if signature is invalid
      ✔ should revert if not authorized addresses must be equal

  EntryPoint
node version: HardhatNetwork/2.12.4/@nomicfoundation/ethereumjs-vm/6.0.0
    #simulateValidation
      ✔ should fail if validateUserOp fails
      ✔ should report signature failure without revert
      ✔ should revert if wallet not deployed (and no initcode)
      ✔ should revert on oog if not enough verificationGas
      ✔ should succeed if validateUserOp succeeds (44ms)
      ✔ should return empty context if no paymaster
      ✔ should prevent overflows: fail if any numeric value is more than 120 bits
      ✔ should not call initCode from entrypoint (41ms)

  Schnorr MultiSign Test
    ✔ should generate a schnorr musig2 and validate it on the blockchain (202ms)
    ✔ check none zero mergedKeyIndex (56ms)
    should update account key
      ✔ should sign Schnorr message
      ✔ should return 0 if sign Schnorr with wrong key
      ✔ should revert if setMergedKey not invoke from intrenal
      ✔ should revert if revoke merged key (136ms)
      ✔ should revert if merged key index is wrong (39ms)
      ✔ should recover merged key (132ms)
      ✔ should revert if revoke merged key 2 (115ms)
      setAuthorized() function test
        ✔ should revert if not internal invoke
        ✔ should revert if authorized address is zero
        ✔ should revert if authorized address same as recovery address
        ✔ should revert if cosigner address same as recovery address
        ✔ should update key by setAuthorized() (131ms)
        ✔ should update key by setAuthorized() to zero (100ms)


  127 passing (10s)

Schnorr MultiSign Test
    ✔ should generate a schnorr musig2 and validate it on the blockchain (241ms)
    ✔ check none zero mergedKeyIndex (59ms)
    should update account key
      ✔ should sign Schnorr message
      ✔ should return 0 if sign Schnorr with wrong key
      ✔ should revert if setMergedKey not invoke from intrenal
      ✔ should revert if revoke merged key (147ms)
      ✔ should revert if merged key index is wrong (43ms)
      ✔ should recover merged key (132ms)
      ✔ should revert if revoke merged key 2 (112ms)
      setAuthorized() function test
        ✔ should revert if not internal invoke
        ✔ should revert if authorized address is zero
        ✔ should revert if authorized address same as recovery address
        ✔ should revert if cosigner address same as recovery address
        ✔ should update key by setAuthorized() (130ms)
        ✔ should update key by setAuthorized() to zero (94ms)


  15 passing (2s)
```

Code Coverage

Quantstamp usually recommends developers to increase the branch coverage to `90%` and above before a project goes live, in order to avoid hidden functional bugs that might not be easy to spot during the development phase. For code coverage, the current branch coverage achieves a low score that must be improved before deployment.

Fix review update

The team significantly improved the test coverage of the system.
Command to run: `npx hardhat coverage`

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	54.05	26.47	51.72	63.64	
BloctoAccount.sol	38.89	37.5	61.54	52	... 116,123,132
BloctoAccountCloneableWallet.sol	100	100	100	100	
BloctoAccountFactory.sol	86.67	21.43	50	81.82	112,118,125,131
BloctoAccountProxy.sol	100	100	100	100	
TokenCallbackHandler.sol	0	0	0	0	23,32,41,45
contracts/CoreWallet/	46.85	22.46	40.91	47.33	
BytesExtractSignature.sol	100	100	100	100	
CoreWallet.sol	46.36	22.46	38.1	46.62	... 672,675,677
contracts/Paymaster/	0	0	0	0	
VerifyingPaymaster.sol	0	0	0	0	... 108,116,118
All files	46.15	22.22	42.11	48.18	
Fix review update					
File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
BloctoAccount.sol	100	100	100	100	
BloctoAccountCloneableWallet.sol	100	100	100	100	
BloctoAccountFactory.sol	100	100	100	100	
BloctoAccountProxy.sol	100	100	100	100	
contracts/CoreWallet/	100	97.86	100	100	
BytesExtractSignature.sol	100	100	100	100	
CoreWallet.sol	100	97.86	100	100	
contracts/Create3/	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
Bytes32AddressLib.sol	100	100	100	100	
CREATE3.sol	100	100	100	100	
CREATE3Factory.sol	100	100	100	100	
ICREATE3Factory.sol	100	100	100	100	
All files	100	98.28	100	100	

Changelog

- 2023-07-01 - Initial report
- 2023-08-04 - Final report

About Quantstamp

Quantstamp