Urbit Keygen JS Library

Code Review

November 30th 2018

Version 2.0.0

Prepared by

Bloctrax

bloctrax

Table of Contents

Introduction

This document includes the results of the code review for Urbit’s Keygen JS library as found in the section titled ‘Source Code’. The code review was performed by the Bloctrax team from November 9th 2018 to November 18th 2018.

The purpose of this engagement is to review Urbit’s Keygen JS library source code, and provide feedback on the design, architecture, and quality of the source code.

Overall Assessment

This section contains the original assessment for posterity, however, after receiving the original version of this report the Urbit team resolved some of the issues described below. See the “Issues Descriptions and Recommendations” section for details on individual issues and their corresponding resolutions.

Our overall assessment of the Urbit key library is that there are a number of deviations from the specification that should be addressed by either updating the spec, or updating the implementation to match it.

In addition, the test suite looks to be healthier than previous versions of the codebase that we reviewed. It appears there is now 100% statement/branch/line coverage and the core logic of the library appears to be more extensively tested. For example, property tests have been added for some functionality, and the generateWallet functionality now tests a variety of different inputs and asserts on all the provided outputs. The generateWallet tests could be improved, however, by making sure that the generated wallet can actually do what it's designed to. For example, wallet2.json ends up with three different shards any 2 of which should be able to re-generate the original ticket, but this isn’t tested. In addition, it would be good to ensure that all of the derived seeds can actually be used to generate the keys/addresses they’re associated with. I.E. by definition, the test asserts that the provided ticket can be used to generate the hard-coded wallet, but it does not assert that the generated non-master seeds can be used to generate their associated keys.

Specification

Our understanding of the specification was based on the following sources:

- Our understanding of the desired behavior based on our previous review of the Urbit Constitution Solidity code.
- Discussions with the Urbit team.
- The Urbit Wallet Specification document which was provided to us by the Urbit team.

Source Code

The following source code was reviewed:

Repository	Commit
index.js	7ed2da1ad09a3aea06441c35848af3987ad21f84

Note: This document contains a review only of the code contained in the file listed above. The review does not include any of the dependencies being used, including but not limited to third party libraries and Urbit's own urbit-ob library.

Severity Level Reference

Level	Description
High	The issue poses existential risk to the project, and the issue identified could lead to massive financial or reputational repercussions.
Medium	The potential risk is large, but there is some ambiguity surrounding whether or not the issue would practically manifest.
Low	The risk is small, unlikely, or not relevant to the project in a meaningful way.
Code Quality	The issue identified does not pose any obvious risk, but fixing it would improve overall code quality, conform to recommended best practices, and perhaps lead to fewer development issues in the future.

Issues Descriptions and Recommendations

Deviations from the Urbit wallet specification

MEDIUM Resolved by updating the Urbit wallet specification to match the implementation.

The Urbit wallet specification states that the salt for argon2u should be “urbitwallet” but the implementation uses “urbitkeygen”.

The specification states that in order to generate an Ethereum address from the wallet’s private key, use `secp256k1` to get the uncompressed public key from the private key, however, the existing code just uses the already generated public key directly. This should achieve the same result, but should perhaps be called out more explicitly in the specification.

In addition to the above, the existing implementation is generating checksummed Ethereum addresses, but the specification makes no mention of the checksumming process.

The specification implies (but does not outright state) that for network keys, the generated private key / public key pairs will be returned for both the encryption and authentication keys, however, the existing implementation actually returns the public key and the seed that was used to generate the private key, but not the private key itself.

The specification states that the network seed should be run through SHA-512, and then the first 256 bits of the result should be used as the encryption key, and the last 256 bits as the authentication key, however, the existing implementation does exactly the opposite.

The `urbitKeysFromSeed` function, aside from the previously discussed deviations from the specification, does a lot of reversing of values that is not mentioned in the specification at all. We assume this has something to do with the implementation of the corresponding function in the Urbit ecosystem, but we can’t verify that unless the specification is updated to match.

Issues with the Urbit Wallet specification itself

LOW Resolved by making ship required argument and updating specification.

The specification states that a value of zero can be used for the “no one specific ship” case, however, zero is a valid ship which seems like an edge case that could potentially cause an issue.

Unnecessary function implementation

CODE QUALITY Urbit team opted to leave as is.

The keygen library defines a function called `toChecksumAddress` which is then tested against a reference implementation in the test suite. It seems like it would be better to just expose the reference implementation instead of re-implementing it and then testing against another implementation.

Lack of detail in the specification

CODE QUALITY Revision issue resolved by updating the specification, and the Urbit team does not believe that the specification needs to explicitly mention the bips32 chain code.

The specification does not specify that the revision will always default to zero if not provided.

The specification makes no mention of the bips32 chain code.

Appendix

Exhibit A - Disclaimer