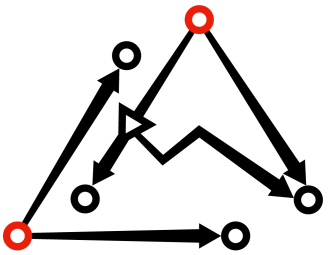

Least Authority

PRIVACY MATTERS

OrchardZSA Protocol

Security Audit Report

ZCG

Final Audit Report: 3 January 2025

Table of Contents

Overview

Background

As the Zcash Ecosystem Security Lead, ZCG has requested that Least Authority perform a security audit of the OrchardZSA Protocol by the Qedit team.

Project Dates

- **October 28, 2024 - December 10, 2024:** Initial Code Review *(Completed)*
- **December 12, 2024:** Delivery of Initial Audit Report *(Completed)*
- **3 January, 2025:** Verification Review *(Completed)*
- **3 January, 2025:** Delivery of Final Audit Report *(Completed)*

The dates for verification and delivery of the Final Audit Report will be determined upon notification from the Qedit team that the code is ready for verification.

Review Team

- Poulami Das, Security / Cryptography Researcher and Engineer
- Mirco Richter, Cryptography Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the OrchardZSA Protocol followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

- Review of the difference between the `zsa1` branch and the main branch In the Orchard repository:
  - A review for compatibility with the provided zips.
    - can be seen in https://github.com/QED-it/orchard/pull/7.
      - Excluding the circuit that was reviewed independently.
- Review of the changes proposed in the Halo2 gadgets folder:
  - Pull request:
    - https://github.com/QED-it/halo2/pull/18
- The `` `/src/circuit/' `` folder:
  - including dependencies that differ from the original Orchard code.

Specifically, we examined the following Git revisions for our initial review:

- Orchard: `a7c02d22a1e2f4310130ae2e7b9813136071bc75`
- Halo2: `290bc56539022c7b47d3da6201958ae2d5a694207`

For the verification, we examined the Git revision:

- `3d2515b3b52b2df47c0778050379e6588061fb95`

For the review, these repositories were cloned for use during the audit and for reference in this report:

- Orchard: https://github.com/LeastAuthority/QED-it-orchard/pull/1

- Halo2:
  https://github.com/LeastAuthority/QED-it-halo2/pull/1

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Background zips for reference:
  - https://zips.z.cash/zip-0226
  - https://zips.z.cash/zip-0227
- Circuit design document / specification:
  - https://docs.google.com/document/d/1DzXBqZl_l3aIs_gcelw3OuZz2OVMnYk6Xe_1lBsTji8/edit
- Derive Nullifier: modifications detailed in this section of the ZIP:
  - https://qed-it.github.io/zips/zip-0226#split-notes
- Value Commitment - Modifications detailed in this section of the ZIP:
  - https://qed-it.github.io/zips/zip-0226#value-commitment
- Value Commitment -
  - This presentation explains the changes in the circuit:
    https://docs.google.com/presentation/d/1VrieRWuceJejaMIo_CZPc3RwqFej_qP2nip30mlUJTM/edit#slide=id.p
- Note Commitment:
  - Modifications detailed in this section of the ZIP
    https://qed-it.github.io/zips/zip-0226#note-structure-commitment
  - This presentation explains the changes in the circuit
    https://docs.google.com/presentation/d/1mt1Bv92fMqaXDfLbruZ6A9HFmjGuef_rveepLd1MaHY/edit#slide=id.p
- Zcash protocol Specification Version 2024.5.1-112-gcf7a5c [NU6]
  https://zips.z.cash/protocol/protocol.pdf
- Understanding the Security of Zcash, Daira Emma-Hopwood:
  https://github.com/daira/zcash-security/blob/main/zcash-security.pdf

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Vulnerabilities within each component and whether the interaction between the components is secure;
- Key management, including secure private key storage and management of encryption and signing keys;
- Denial of Service (DoS) and other security exploits that would impact the intended use or disrupt the execution;
- Protection against malicious attacks and other ways to exploit;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Our team performed a security audit of the OrchardZSA Protocol, an extension of the Zcash Orchard protocol. The Zcash Orchard protocol is a version of the Zcash protocol with several protocol rule changes applied as part of the Network Upgrade 5 that enables Zcash's existing goal of shielded payments, with several security and privacy guarantees. Extending the Zcash Orchard protocol to OrchardZSA allows Zcash to support shielded custom assets that will, in turn, enable Zcash to interoperate with assets of other blockchains. To allow this, OrchardZSA protocol makes several subtle changes to the existing Orchard protocol by: introducing the concepts of custom assets and split input notes; updating the note; updating the value commitments and value balancing mechanism; introducing burning of assets; and simultaneously maintaining backward compatibility with the Orchard protocol for the native ZEC asset.

We audited the provided code against the security properties of the OrchardZSA protocol extensions assuming they are the same as in the original Orchard protocol, which are the properties of balance, spendability, privacy, non-malleability and diversifier address unlinkability. In particular to the spendability property, we investigated the possibility of the Faerie Gold attack and roadblock attack but did not identify any vulnerability.

In particular, we reviewed code updates in the pull requests here and here against the Zcash improvement proposals ZIP 226 and ZIP 227 under the assumption that these proposals are correct. In particular, we examined the ZSA circuit extensions of the original Orchard circuit under the assumption that the original Orchard circuit is correct, with a focus on missing constraints or deviations from ZIP 226 and ZIP 227. We did not identify any issues in the ZSA protocol extensions.

In addition, we investigated the changes made to the Halo2 gadgets and circuits, focusing on deviations from the specifications as well as missing constraints. We did not identify any issues.

Our team also reviewed the non-circuit related changes and improvements made to the Orchard protocol, focusing on cryptographic best practices and standards. We did not identify any areas of concern.

Overall, we found the system to be well-designed and clearly documented, with a strong emphasis on security.

Code Quality

The repositories in scope, as well as the original Orchard and Halo2 codebase, are well-organized and of high quality, in that they adhere closely to development best practices.

Tests

The repositories in scope, as well as the original functionality, are extensively tested.

Documentation and Code Comments

The project documentation consisted of ZIP 226 and ZIP 227, which are detailed protocol descriptions of the changes that have to be performed in order to implement the ZSA extension to the Orchard protocol. Additionally, while the codebase includes some code comments, our team noted that several of the links are broken. We also found a typographical error in the code comments. We recommend updating the broken links and correcting the error in the code comments (Suggestion 3).

Scope

The scope of this review was sufficient and included all security-relevant components.

Dependencies

We examined all the dependencies implemented in the codebase and identified some instances of unmaintained dependencies. We recommend improving dependency management (Suggestion 2).

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Suggestion 1: Precompute Nullifiers as Constants To Generate Smaller Circuits	Unresolved
Suggestion 2: Update and Maintain Dependencies	Planned
Suggestion 3: Correct Typographical Errors and Broken Links in Code Comments	Resolved
Suggestion 4: Consider Unifying the Concept of Unique AssetBase per IssuanceAction	Resolved

Suggestions

Suggestion 1: Precompute Nullifiers as Constants To Generate Smaller Circuits

Location

`src/circuit/derive_nullifier.rs#L92`

Synopsis

The circuit currently generates the nullifier constant `nullifier_l` during circuit generation.

Mitigation

We recommend that the nullifier constant `nullifier_l` be generated first outside of the circuit, and then be used as a constant in the circuit.

Status

The Qedit team has responded that they are not sure that suggested changes would result in increased efficiency.

Verification

Unresolved.

Suggestion 2: Update and Maintain Dependencies

Location

`blob/zsa1/Cargo.toml`

Synopsis

Orchard and Halo2 Crate use the following unmaintained, yanked, or unsound dependencies:

Crate: atty
Version: 0.2.14
Warning: unmaintained
Title: `atty` is unmaintained
Date: 2024-09-25
ID: RUSTSEC-2024-0375
URL: https://rustsec.org/advisories/RUSTSEC-2024-0375

Crate: atty
Version: 0.2.14
Warning: unsound
Title: Potential unaligned read
Date: 2021-07-04
ID: RUSTSEC-2021-0145
URL: https://rustsec.org/advisories/RUSTSEC-2021-0145

Crate: proc-macro-error
Version: 1.0.4
Warning: unmaintained
Title: proc-macro-error is unmaintained
Date: 2024-09-01
ID: RUSTSEC-2024-0370
URL: https://rustsec.org/advisories/RUSTSEC-2024-0370

Crate: serde_cbor
Version: 0.11.2
Warning: unmaintained
Title: serde_cbor is unmaintained
Date: 2021-08-15
ID: RUSTSEC-2021-0127
URL: https://rustsec.org/advisories/RUSTSEC-2021-0127

Crate: const-cst
Version: 0.3.0
Warning: unsound
Title: const-cstr is Unmaintained
Date: 2023-03-12
ID: RUSTSEC-2023-0020
URL: https://rustsec.org/advisories/RUSTSEC-2023-0020

Crate: wasm-bindgen
Version: 0.2.88
Warning: yanked

Crate: bytemuck
Version: 1.16.1
Warning: yanked

Mitigation

We recommend following a process that emphasizes secure dependency usage to avoid introducing vulnerabilities to the OrchardZSA Protocol and to mitigate supply-chain attacks, which includes:

- Manually reviewing and assessing currently used dependencies;
- Upgrading dependencies with known vulnerabilities to patched versions with fixes;
- Replacing unmaintained dependencies with secure and battle-tested alternatives, if possible;
- Pinning dependencies to specific versions, including pinning build-level dependencies in the
- `package.json` file to a specific version;
- Only upgrading dependencies upon careful internal review for potential backward compatibility issues and vulnerabilities; and
- Including Automated Dependency auditing reports in the project's CI/CD workflow.

Status

The Qedit team has stated that they plan to address this suggestion prior to the network update.

Verification

Unresolved.

Suggestion 3: Correct Typographical Error and Broken Links in Code Comments

Location

Examples (non-exhaustive):

`src/circuit/circuit_zsa.rs#L70`

`zsa1/src/issuance.rs#L481`

Synopsis

During our review, our team identified a typographical error and incorrect references in the code comments that impact the quality, readability, and maintainability of the codebase. To illustrate, the following is a non-exhaustive list of examples:

- In `src/circuit/circuit_zsa.rs#L70`, this link is broken and needs to be updated.
- In `zsa1/src/issuance.rs#L481`, the sentence should be corrected to "*Asset description size is correct*."

Mitigation

We recommend addressing the items listed above.

Status

The Qedit team has corrected errors and broken links in aforementioned locations.

Verification

Resolved.

Suggestion 4: Consider Unifying the Concept of Unique AssetBase per IssuanceAction

Location

zsa1/src/issuance.rs#L340

zsa1/src/issuance.rs#L40

zsa1/src/issuance.rs#L224

zsa1/src/issuance.rs#L332

Synopsis

As defined in issuance.rs, based on the structure of the `IssueAction` and `add_recipient` functions, there should not be more than one `IssueAction` with the same asset description. However, the functions `get_actions_by_asset` and `get_actions_by_desc` have a `vector` return type, denoting there could be more than one element that corresponds to a specific asset description.

Mitigation

If an `IssueBundle` does not contain more than one `IssueAction` with the same asset description, we recommend updating the functions, such as `get_actions_by_asset`, `get_actions_by_desc`, and `finalize_action` to reflect the same.

Status

The Qedit team has updated the functions `get_actions_by_desc`, `get_actions_by_asset` and relevant code locations to reflect that an `IssueBundle` contains at most one unique `IssueAction` with the same asset description.

Verification

Resolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.