Ethereum Foundation

KZG Powers Of Tau Ceremony Review

Version: 3.0

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Ethereum Foundation KZG Ceremony. The review focused solely on the security aspects of the Rust implementation of the KZG Ceremony Sequencer code and associated React front-end, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the code. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Powers Of Tau KZG Ceremony contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Ethereum Foundation's KZG Ceremony.

Overview

The Powers of Tau KZG Ceremony is a required step to implement EIP-4844 "Proto-Danksharding" and "Dankssharding" on Ethereum.

This is a trusted setup with a 1 of N trust model, meaning we only need to trust that one actor is trustworthy and acting as intended in order to make the whole process trustworthy. As a result of this the ceremony is intended to be as large as possible with several thousand participants over its lifetime.

The Sequencer is an important part of this, as it coordinates the actions of contributors and validates their contributions. In addition to the Sequencer, a React front-end website provides a means for users to take part in the ceremony and contribute to the setup by generating entropy within their browser.

Security Assessment Summary

This review was conducted on the files hosted on the kzg-ceremony-sequencer repository and were assessed at commit 6da76be4cb for the sequencer. For the frontend, the files in this review were hosted on the trusted-setup-frontend repository and were assessed at commit c2fc0fdf8d.

Retesting was performed on commits 34b5fc8 for the sequencer and 40d421f for the frontend. Retesting covered all mitigations to raised issues. No new issues were found during retesting.

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the code. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of Rust and Typescript.

Additionally, the manual review process focused on all known Rust and Typescript anti-patterns and attack vectors. These include, but are not limited to, the following vectors: error handling and wrapping, panicking macros, arithmetic errors, UTF-8 strings handling, index out of bounds and resource exhaustion.

To support this review, the testing team used the following automated testing tools:

- cargo audit: `https://crates.io/crates/cargo-audit`
- cargo deny: `https://github.com/EmbarkStudios/cargo-deny`
- cargo tarpaulin: `https://crates.io/crates/cargo-tarpaulin`
- cargo geiger: `https://github.com/rust-secure-code/cargo-geiger`
- clippy: `https://github.com/rust-lang/rust-clippy`

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 15 issues during this assessment. Categorised by their severity:

- Critical: 1 issue.
- Medium: 2 issues.
- Low: 6 issues.
- Informational: 6 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Ethereum Foundation's codebase. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as "informational".

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
KZG-01	BLST Library Decoding Errors Not Handled Correctly	Critical	Resolved
KZG-02	Use of Bearer HTTP Authorisation Without TLS	Medium	Resolved
KZG-03	HTTP Body Unnecessarily Contains Authorisation Token `session_id`	Medium	Resolved
KZG-04	DoS Vector Fetching `info/current_state` API Endpoint	Low	Resolved
KZG-05	`LobbyIsFull` Error is Not Correctly Detected During Sign-in	Low	Resolved
KZG-06	BLST `random_fr()` Does Not Use 512 Bits of Expanded Entropy	Low	Resolved
KZG-07	Unbounded Recursion May Break Max Stack Depth	Low	Resolved
KZG-08	Use of Yanked Crates in `kzg-ceremony-sequencer` Repository	Low	Resolved
KZG-09	Inconsistent Handling of Zero X-Coordinate and Point at Infinity	Low	Resolved
KZG-10	`active_contributor` State Does Not Update on Error	Informational	Resolved
KZG-11	Potential Panics if `bytes_to_hex()` is Called With Disproportionate Sizes	Informational	Resolved
KZG-12	Inefficient Conversion of `Uint8Array` to `string`	Informational	Resolved
KZG-13	Error Handling in Lobby Will Continue to Loop	Informational	Resolved
KZG-14	Overall Test Coverage	Informational	Resolved
KZG-15	Miscellaneous General Comments	Informational	Resolved

KZG-01	BLST Library Decoding Errors Not Handled Correctly		
Asset	`crypto/src/engine/blst/g1.rs` & `crypto/src/engine/blst/g2.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

When using the BLST library to perform cryptography it is necessary to be able to decode byte inputs into the cryptography structures used in the library. The decoding is performed by the BLST external library using the functions `blst_p1_uncompress()` for G1 and `blst_p2_uncompress()` for G2. The return values of these functions are error messages from the BLST library. `kzg-ceremony-crypto` immediately drops the return values from these functions, hence allows malformed objects which are not valid points on the curve to be successfully decoded.

The following code snippet from `crypto/src/engine/blst/g2.rs` shows the decoding of a byte array into a BLST object.

```
12 fn try_from(g2: G2) -> Result<Self, Self::Error> {
       unsafe {
14         let mut p = Self::default();
           blst_p2_uncompress(&mut p, g2.0.as_ptr());
16         Ok(p)
       }
18 }
```

As seen in this snippet the return value of `blst_p2_uncompress()` is immediately dropped, thereby preventing the correct handling of errors.

The impact is rated as critical due to the ability to pass points that are malformed to the ceremony.

One such example breaks the sequencer's intended actions by bypassing the check that the current Tau public key is not the point at infinity.

`blst_p2_uncompress(&mut p, g2.0.as_ptr())` will not modify `p` for certain errors. `p` will therefore be `Self::default()` for these errors, which is the point at infinity. For example `G2([0u8; 96])` (i.e. in hex `0x0000...00`) will decode to the point at infinity when the valid response should be to error since the correct encoding of infinity is `0xc000...00`.

The following check occurs in crypto/src/transcript.rs.

```
92 // Non-zero check
   if contribution.pot_pubkey == G2::zero() {
94     return Err(CeremonyError::ZeroPubkey);
   }
```

Both `contribution.pot_pubkey` and `G2::zero()` are 96 byte arrays to represent compressed G2 points. `G2::zero()` is a constant which in hex form is `0xc000...00`. Since `0x0000...00` also represents the point at infinity due to the previously mentioned bug we may bypass the check for infinity in `transcript.rs` by setting `contribution.pot_pubkey` to `0x0000...00`.

The result of this would be sending current and future Tau powers to the point at infinity rendering the ceremony useless.

Recommendations

The return values need to be handled in each of `blst_p1_uncompress()` and `blst_p2_uncompress()` in `crypto/src/engine/blst/g1.rs` and `crypto/src/engine/blst/g2.rs` respectively. For the case where the return value is not `BLST_SUCCESS = 0` the error should be propagated and the malicious contribution to be rejected.

Resolution

The issue has been resolved in PR #142.

Additional checks have been added to both `TryFrom<G1>` and `TryFrom<G2>` for the BLST points. The checks verify the return value of `blst_p1_uncompress()` and `blst_p2_uncompress()` is `BLST_SUCCESS` and return an error if required.

KZG-02	Use of Bearer HTTP Authorisation Without TLS		
Asset	`kzg-ceremony-sequencer/*`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

The Sequencer uses Bearer authorisation to identify different users. However, this is not used with HTTPS and so the Bearer token is included in the HTTP Header as plain text.

The impact is that any attacker witnessing the request can read the user's Bearer token. With access to the token the user can hijack the session and input a contribution on the user's behalf or spam messages and force the user to be rejected through rate limiting.

Recommendations

We recommend using HTTPS. Adding TLS to the HTTP messages will encrypt the header and prevent the Bearer token from being read by a third party.

Resolution

The development team have planned and implemented a setup where an HTTPS proxy is used as a wrapper around the sequencer server. The HTTPS proxy is intended to be located on the same machine as the sequencer server. Any external API calls are encrypted with TLS to the HTTPS proxy which are then internally forward to the sequencer server.

KZG-03	HTTP Body Unnecessarily Contains Authorisation Token `session_id`		
Asset	`trusted-setup-frontend/src/api.ts`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

The function `tryContribute()` has a API call that contains `session_id` within the JSON body in addition to the HTTP Authorise field. The `session_id` is a Bearer token which providers authorisation and may be re-used indefinitely by any attacker who has read this token. Hence, its usage should be kept to a minimum to reduce the attack surface of stealing the authorisation token.

```
async tryContribute(
    session_id: string
): Promise<ErrorRes | TryContributeRes> {
    const res = await fetch(`${API_ROOT}/lobby/try_contribute`, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            Authorization: `Bearer ${session_id}`
        },
        body: JSON.stringify({
            session_id
        })
    })
    return await res.json()
}
```

The code snippet above shows `session_id` being included in both the HTTP header and body.

Recommendations

Remove the unneeded `session_id` from the body of the `/lobby/try_contribute` API call.

Resolution

Commit da2634d resolves this issue by removing the `session_id` from the JSON body.

KZG-04	DoS Vector Fetching `info/current_state` API Endpoint		
Asset	`trusted-setup-frontend/src/pages/entropyInput.tsx`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

The API endpoint `GET into/current_state` returns the current Powers of Tau transcript file causing a DoS (Denial of Service) attack vector.

As an unauthorised endpoint any machine can make `GET` requests to this endpoint. The transcript file will be at least 6 megabytes in size.

Additionally, the transcript file is required to be written to and read from the `contribute` API. Acquiring a write-lock over the transcript file in `contribute` means it will need to wait for the read lock to be free in `info/current_state` to be released.

The impact is therefore two-fold in using significant network resources by returning at least a 6 megabytes file and creating a contested lock.

Recommendations

Consider allowing the `info/current_state` endpoint to be disabled during periods of attack or heavy network usage.

Additionally, consider caching the transcript file in memory to avoid reading from disk and minimise requirements to obtain the file lock.

Resolution

The solution to be implemented by the development team adds an HTTP proxy server to wrap the sequencer server. The proxy will cache all `info/*` requests thereby reducing the need to read from disk.

Furthermore, the proxy will have a DoS protection system similar to Cloud Flare to provide DoS protection.

KZG-05	`LobbyIsFull` Error is Not Correctly Detected During Sign-in		
Asset	`trusted-setup-frontend/src/pages/signin.tsx`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `AuthErrorPayload::LobbyIsFull` error message is not correctly handled for either of the functions `onSigninSIE()` and `onSigninGithub()` in `signin.tsx`.

`api.getRequestLink()` makes an API call to the sequencer using the `/auth/request_link` endpoint. The API `auth/request_link` will return the `AuthErrorPayload::LobbyIsFull` error when the lobby is full.

The return values of `api.getRequestLink()` are used as redirect URLs during sign-in, as seen in the following code snippet.

```
const onSigninSIE = async () => {
    setIsLoading(true);
    const requestLinks = await api.getRequestLink()
    window.location.replace(requestLinks.eth_auth_url)
}


const onSigninGithub = async () => {
    setIsLoading(true);
    const requestLinks = await api.getRequestLink()
    window.location.replace(requestLinks.github_auth_url)
}
```

When there is an error in `api.getRequestLink()` the value of `requestLinks` will be `undefined`. Hence, the user is redirected to `/undefined`, as this route does not exist the result will be a blank page.

Recommendations

This issue may be resolved by improving the error handling to catch the `LobbyIsFull` along with error in each of `onSigninSIE()` and `onSigninGithub()`.

Resolution

The issue is resolved in commit 3a99149 by handling the `LobbyIsFull` error and redirecting to the `LOBBY_FULL` page.

KZG-06	BLST `random_fr()` Does Not Use 512 Bits of Expanded Entropy		
Asset	`crypto/src/hex_format.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The function `random_fr()` is used to generate a field prime object to represent the private key or Tau of a ceremony contribution. To avoid modulo bias random entropy should first be expanded before it is reduced.

There is a bug in the current `random_fr()` implementation for BLST that entropy is expanded to 64 bytes however only 32 bytes are used.

```
pub fn random_fr(entropy: [u8; 32]) -> blst_fr {
    // Use ChaCha20 CPRNG
    let mut rng = ChaCha20Rng::from_seed(entropy);

    // Generate tau by reducing 512 bits of entropy modulo prime.
    let mut buffer = [0u8; 64];
    rng.fill(&mut buffer);

    let mut scalar = blst_scalar::default();
    let mut ret = blst_fr::default();

    unsafe {
        blst_scalar_from_be_bytes(&mut scalar, buffer.as_ptr(), 32);
        blst_fr_from_scalar(&mut ret, &scalar);
    }

    ret
}
```

From the code snippet `blst_scalar_from_be_bytes(&mut scalar, buffer.as_ptr(), 32)` is called with 32 as the final parameter. As a result only 32 bytes of the array `buffer` will be read rather than the full 64 bytes in `buffer`.

Recommendations

Consider using the `blst::blst_keygen()` function to generate a secret key. This function adheres to the CFRG Specifications.

Resolution

The recommendation to this issue has been implemented in PR #148, thereby resolving the issue.

Furthermore, the Arkworks engine has been updated to implement an equivalent key generation.

KZG-07	Unbounded Recursion May Break Max Stack Depth		
Asset	`trusted-setup-front-end/src/pages/lobby.tsx`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `poll()` function used to repeatedly make the API call `/lobby/try_contribute` operates recursively without a deterministic base case to terminate the recursion.

A recursive function should have a maximum number of recursions that can occur. The function `poll()` will continue making recursive calls until the user successfully enters the lobby and begins contributing. Progression from the lobby to contributing is a psuedorandom function, the next user to contribute after the previous user has completed their contribution. Hence, there is no strict upper bound for how much longer a user may remain in the lobby.

The user will recursively call poll every `LOBBY_CHECKIN_FREQUENCY` while they are in the lobby. The impact of the recursion is that eventually `RangeError: Maximum call stack size exceeded` will be triggered.

The following code snippet shows the recursive programming of `poll()`.

```
useEffect(() => {
  async function poll(): Promise<void> {
  // periodically post /slot/join
  const res = await tryContribute.mutateAsync()
  if (isSuccessRes(res) && res.hasOwnProperty('contributions')) {
      updateContribution(JSON.stringify(res))
      navigate(ROUTES.CONTRIBUTING)
  } else {
      const resError = res as ErrorRes
      switch (resError.code) {
      case 'TryContributeError::RateLimited':
          setError(resError.error)
          break
      case 'TryContributeError::UnknownSessionId':
          setError(
          resError.error +
              '. You might have taken more time to get into the lobby. Please reload and sign in again'
          )
          break
      case 'TryContributeError::AnotherContributionInProgress':
          setError(resError.error)
          break
      default:
          setError('Unknown error code: ' + resError.code)
          break
      }
      //  try again after LOBBY_CHECKIN_FREUQUENCY
      await sleep(LOBBY_CHECKIN_FREQUENCY)
      return await poll()
    }
  }
  poll()

  // eslint-disable-next-line react-hooks/exhaustive-deps
}, [])
```

Recommendations

To avoid breaching the maximum call stack size a loop may be used rather than recursion. A loop will garbage collect variables that fall out of scope thereby maintaining a consistent call stack size.

Resolution

The recursion has been translated into a `while` loop in commit 32fe7bd.

KZG-08	Use of Yanked Crates in `kzg-ceremony-sequencer` Repository		
Asset	`kzg-ceremony-sequencer/*`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Rust advisory lists two crates that exist in the dependency tree that have been yanked.

- `blake2` version "0.10.4"

```
blake2 0.10.4
└── coins-core 0.7.0
    └── coins-bip32 0.7.0
        ├── ethers-signers 1.0.0
        │   └── kzg-ceremony-sequencer 0.1.0
        └── coins-bip39 0.7.0
            └── ethers-signers 1.0.0
```

- `futures-intrusive` version "0.4.1"

```
futures-intrusive 0.4.1
└── sqlx-core 0.6.2
    ├── sqlx-macros 0.6.2
    │   └── sqlx 0.6.2
    │       └── kzg-ceremony-sequencer 0.1.0
    └── sqlx 0.6.2
```

Recommendations

Crates may be yanked for a range of bugs or issues. It is strongly recommended against using yanked crates, this may require pushing changes upstream to bump dependency versions.

Resolution

The `Cargo.lock` file has been updated such that no yanked crates are used as dependencies. PR #159 shows the updates made to `Cargo.lock`.

KZG-09	Inconsistent Handling of Zero X-Coordinate and Point at Infinity		
Asset	`kzg-ceremony-sequencer/crypto/engine/*`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

There are two implementations of the underlying BLS cryptography protocol, BLST and Arkworks. Discrepancies exist between the two protocols about how the point at infinity should be handled.

Each of the protocols implement a rust trait `Engine`.

The there is a discrepancy between the two implementations in how they handle the points with X-coordinate zero (i.e. $(0, \pm 2)$) for the function `add_tau_g1(tau: &Tau, powers: &mut [G1])`. Arkworks will error if any of `powers` has X-coordinate zero whereas BLST will return `Ok()` multiplying the coordinate by Tau. The impact is low as the points are not within the correct subgroup and so will be rejected by `validate_g1()`.

A second discrepancy exists in `verify_signature()` and how the points at infinity are handled. Arkworks may verify the signature as valid if both the signature and public key are the point at infinity whereas BLST will reject this signature. The impact is low as the Powers of Tau public key validated to be non-zero.

Recommendations

We recommend aligning these implementations for both of these cases. Since zero public keys are undesirable, consider updating the Arkworks implementation to reject these signatures.

Additionally, since the points $(0, \pm 2)$ are not in the correct subgroup they may be safely rejected by returning error in `add_tau_g1()`.

Resolution

The issue has been resolved by rejecting the point at infinity in the Arkworks implementation. The resolution can be seen in PR #173.

KZG-10	`active_contributor` State Does Not Update on Error
Asset	`kzg-ceremony-sequencer/src/api/v1/contribute.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

During processing a user's contribution, if an error occurs in the signer `receipt.sign()` on line [**88**] then this triggers an immediate return without modifying the contributor state.

The result of this is that the `active_contributor` state would be stuck in `Contributing` until the `expire_current_contributor()` thread is triggered. This unnecessarily consumes the sequencer's contributing time as `lobby/try_contribute` will not promote any other users to the `active_contributor`.

Furthermore, the transcript state is updated before `receipt.sign()` and the storage state is updated after. Therefore, the storage state and transcript state will no longer be synchronised.

This issue is raised as informational as it should not occur in production. For `receipt.sign()` to error an external signer must be used. That is a signer where the private key is not handled directly by the `kzg-ceremony-sequence` program.

Recommendations

Move `receipt.sign()` after the `lobby_state.clear_current_contributor()`, `storage.finish_contribution()` and `num_contributions.fetch_add()` then an error will not impact the transcript, lobby or storage state.

Resolution

PR #150 resolves this issue by applying the recommendation. The transcript, lobby and storage states will now remain synchronised.

KZG-11	Potential Panics if `bytes_to_hex()` is Called With Disproportionate Sizes
Asset	`crypto/src/hex_format.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The function `bytes_to_hex()` can panic if the input arguments `N` or `M` are of disproportionate size.

```
12  pub fn bytes_to_hex<S: Serializer, const N: usize, const M: usize>(
        serializer: S,
14      bytes: [u8; N],
    ) -> Result<S::Ok, S::Error> {
16      assert_eq!(2 + 2 * N, M);
        if serializer.is_human_readable() {
18          let mut hex = [0_u8; M];
            hex[0] = b'0';
20          hex[1] = b'x';
            hex::encode_to_slice(bytes, &mut hex[2..])
22              .expect("BUG: output buffer is of the correct size");
            let str = std::str::from_utf8(&hex).expect("BUG: hex is valid UTF-8");
24          serializer.serialize_str(str)
        } else {
26          serializer.serialize_bytes(&bytes)
        }
28  }
```

The first panic will occur on line [16] if the `assert_eq!()` macro fails.

A second index out of bounds panic will occur if `N = 0` and `M = 2`, when `hex[2..]` is indexed on line [21].

These values are type level arguements that are supplied at compile time. All occurrences in the `kzg-ceremony-sequencer` repository have been checked against the panic conditions and therefore the issue is raised as informational.

Recommendations

Consider returning an error for both potential panics. That is if `N == 0 || 2 + 2 * N != M`.

Resolution

The code has been updated such that it will not panic in the case of malformed arguements. Two separate pull requests are made to resolved this issue, PR #161 and PR #150.

KZG-12	Inefficient Conversion of `Uint8Array` to `string`
Asset	`trusted-setup-frontend/src/pages/entropyInput.tsx`
Status	**Resolved:** See Resolution
Rating	Informational

Description

There is an inefficient conversion of `Uint8Array` to `string` and then back to `Uint8Array`. The following code snippet is from the function `processGeneratedEntropy()`.

```
const entropy = mouseEntropy + keyEntropy + randomBytes(32)
const entropyAsArray = Uint8Array.from(
    entropy.split('').map((x) => x.charCodeAt(0))
)
```

`randomBytes(32)` is of type `Uint8Array` and `mouseEntropy` and `keyEntropy` are of type `string`. Therefore, `entropy` is of type `string`. `randomBytes(32)` is cast to a string object which uses the default format `1,2,3,4,5,6,...,32`.

`entropyAsArray` reads each character in the string as an ASCII value. The values of `randomBytes(32)` will be 44 (",") or 48-57 ("0"-"9").

It is inefficient to cast the values of a `Uint8Array` to `string` then back to `Uint8Array`.

The issue is raised as informational as there is no loss in cryptographic entropy used in HKDF.

Recommendations

Consider having `entropy` only include `mouseEntropy` and `keyEntropy` and convert this to a `Uint8Array` then append `randomBytes(32)`.

Resolution

The issue is mitigated by first converting entropy input from the mouse and keyboard to a `Uint8Array` then appending `randomBytes()`. Commit 46f5532 contains the updated functionality.

KZG-13	Error Handling in Lobby Will Continue to Loop
Asset	`trusted-setup-front-end/src/pages/lobby.tsx`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The function `poll()` will intermittently call the API `/lobby/try_contribute`. Certain errors are non-recoverable such as `TryContributeError::RateLimited` which removes the user when they are rate limited and `TryContributeError::UnknownSessionId` which implies the user does not have a valid Bearer token. These errors will continue the recursion, although they cannot recover.

```
async function poll(): Promise<void> {
  // periodically post /slot/join
  const res = await tryContribute.mutateAsync()
  if (isSuccessRes(res) && res.hasOwnProperty('contributions')) {
      updateContribution(JSON.stringify(res))
      navigate(ROUTES.CONTRIBUTING)
  } else {
      const resError = res as ErrorRes
      switch (resError.code) {
      case 'TryContributeError::RateLimited':
          setError(resError.error)
          break
      case 'TryContributeError::UnknownSessionId':
          setError(
          resError.error +
              '. You might have taken more time to get into the lobby. Please reload and sign in again'
          )
          break
      case 'TryContributeError::AnotherContributionInProgress':
          setError(resError.error)
          break
      default:
          setError('Unknown error code: ' + resError.code)
          break
      }
      //  try again after LOBBY_CHECKIN_FREUQUENCY
      await sleep(LOBBY_CHECKIN_FREQUENCY)
      return await poll()
  }
}
  poll()
```

The code block above shows how `TryContributeError::UnknownSessionId` and `TryContributeError::RateLimited` will break the switch statement and then recursively call `poll()`. Each recursion iteration will repeat these errors, except `TryContributeError::RateLimited` removes the session, hence the next error will be `TryContributeError::UnknownSessionId`.

The issue is considered informational as it does not pose a security risk. The client will continue the recursion causing the `/lobby/try_contribute` API to be called every `LOBBY_CHECKIN_FREQUENCY` which is a minor drain on resources for both the client and server.

Recommendations

Consider handling the non-recoverable errors by displaying the error message and either redirecting to the sign-in page or stopping the recursion.

Resolution

Commit bdff216 adds a case to handle the `LobbyIsFull` error.

KZG-14	Overall Test Coverage
Asset	`/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Testing is a core procedure in quality assurance (QA) to help minimise bugs. The overall quality and quantity of the tests is reasonable in `kzg-ceremony-sequencer` and `trusted-setup-frontend`.

The output of `cargo tarpaulin` as seen in Appendix Test Coverage shows the test coverage of the `kzg-ceremony-sequencer` workspace to be 75%. Test coverage should aim for 100% code coverage and to explore all possible code paths.

A reasonable level of testing is also performed on the `trusted-setup-frontend` repository.

Due to the large scale nature and limit time frame during which the protocol the load testing should also be performed.

Recommendations

We recommend extensive load testing for the `kzg-ceremony-sequencer` server on a machine equivalent to the production machine. Additionally include any production specific setups such as HTTP Proxies. Furthermore, load testing should be performed for integration between the server and `trusted-setup-frontend`.

We also recommend increasing the unit test coverage of `kzg-ceremony-sequencer` to 100% such that all code paths are covered. Consider adding `cargo tarpaulin` to the development life cycle of the project.

Resolution

`cargo tarpaulin` has been added to the CI pipeline to measure test coverage on an iterative basis. With the additional tests included in PR #150 coverage has been increased to 83%.

As some code paths may not be reachable it is infeasible to reach 100% coverage. The development team have performed a manual review of the lines which do not have test coverage.

Load testing has not yet been performed but will be implemented and is considered outside the scope of this repository.

KZG-15	Miscellaneous General Comments
Asset	`/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

Sequencer comments

1. **Unnecessary unwraps in the code base.**
   The unwrap on line [15] of `src/receipt.rs` is unreachable however, it is preferable to return a result.
2. **Redundant code.**
   `mem::replace()` on line [144] of `src/lobby.rs` is unnecessary and can be replaced with a read-only reference.
3. **TODO comments in code that pose low security threat.**
   - `crypto/src/engine/arkworks/mod.rs` on line [235] Tau entropy should be less than the subgroup order.
   - `crypto/src/engine/arkworks/endomorphism.rs` on line [52, 65, 78, 171] optimise code with WNAF.
   - `crypto/src/engine/arkworks/endomorphism.rs` on line [213, 244, 252] improve test coverage.
   - `crypto/src/engine/mod.rs` on line [8] include rust docs.
   - `crypto/src/engine/blst/mod.rs` on line [38, 64] BLST decoding of points is buggy as seen in KZG-01.
   - `crypto/src/engine/blst/mod.rs` on line [248] the Tau secret should be zeroised where possible to prevent reading a secret from memory after use, however this is considered a low security threat for the current system.
   - `crypto/src/engine/both.rs` on line [52] generating Tau will only use the first implementation.
   - `crypto/src/batch_contribution.rs` on line [61] `ChaCha20Rng` should be zeroised but this is considered a low security threat.
   - `src/oauth/ethereum.rs` on line [72] logging is desirable but is considered a low security risk.
   - `src/io.rs` on line [1, 125] panics should be avoided and errors handled for both `read_json_file()` and `write_json_file()`.
   - `src/api/v1/auth.rs` on line [389] use an error instead of option.
   - `src/lib.rs` on line [4] handle clippy lints.

Frontend comments

1. **Switch statement does not handle all error cases**
   The switch statement starting on line [44] of `src/pages/lobby.tsx` does not handle all error cases. The following errors are not handled `LobbyIsFull` and `StorageError` instead these trigger the default case.

2. **Outdated comments**
   The comment on line [37] of `src/pages/lobby.tsx` states `"// periodically post /slot/join"` but the actual route is `/lobby/try_contribute` not `/slot/join`.
3. **Unused API functions**
   In `src/api.ts` the following functions are unused and can be removed:
   - `getStatus()`
   - `getCurrentState()`
   - `getAuthorized()`

   Alternatively these functions could be implemented and used in the following cases rather than making raw queries:
   - `getStatus()` in `src/hooks/useSequencerStatus.ts`
   - `getCurrentState()` in `src/hook/useRecord.ts`
4. **Unused files**
   The following pages are unused and can be safely deleted:
   - `pages/gate.tsx`
   - `pages/mobile.tsx`
5. **Unused type**
   `GetAuthorizedRes` in `src/types.ts` is unused and can be safely removed.
6. **Unprogrammed button**
   The button `View Contribution` in `pages/complete.tsx` does not have an action.
7. **Spelling mistakes**
   These typos were found in a newer commit f091b87:
   - In `src/locales/en/translation.json` on line [27] `"Your Secret, Sigil, and Sample"` should read `"Your Secret, Sigil and Sample"`.
   - In `src/locales/en/translation.json` on line [28] `"Don't forget to return for the summoning ending & spread the words."` should read `"Don't forget to return for the summoning ending & spread the word."`.
   - In `src/locales/en/translation.json` on line [101] `"Use the same wallet you used to signing with Ethereum"` should read `"Use the same wallet you used to sign-in with Ethereum"`.
   - In `src/locales/en/translation.json` on line [121] `"An unexpected number of contributions have been send"` should read `"An unexpected number of contributions have been sent"`.
8. **Unnecessary mobile route**
   `ROUTES.DOUBLE_SIGN` should not be accessible from mobile in `main/src/routes.ts`.
9. **TODO comments in code that pose low security threat.**
   - `public/wasm/pkg/wrapper_small_pot.js` on line [112] quantity of tests should be increased.
   - `src/hooks/useRecord.ts` on line [9] data should be fetched via API.
   - `src/pages/doubleSign.ts` on line [69] potential name change in dependency.
   - `src/pages/complete.ts` on line [28] extra feature for verifying contribution.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team have acknowledged these findings, addressing them where appropriate as follows.

Sequencer comments

1. Fixed in PR #162.
2. Fixed in PR #149.
3. Each of the `TODO` statements has been addressed. The testing team have reviewed all comments provided by the development team in relation to each `TODO`.

Frontend comments

1. Wontfix
2. Fixed in commit a1de4cb
3. Fix in commit 2cf65a7
4. Fixed in commits 21a8604 and be306ef
5. Fixed in commit a1de4cb
6. Fixed in commit b331cff
7. Fixed in commit a1de4cb
8. Fixed in commit aff4eb1
9. Partially fixed in various commits

Appendix A Test Coverage

The output of `cargo tarpaulin --avoid-cfg-tarpaulin` shows the test coverage and lines lacking test coverage.

```
INFO cargo_tarpaulin::report: Coverage Results:
|| Uncovered Lines:
|| crypto/src/batch_contribution.rs: 33, 35, 46-48, 51-54, 56, 74-76, 78-81, 84
|| crypto/src/batch_transcript.rs: 61-63, 74, 80, 87, 93-94
|| crypto/src/contribution.rs: 20-21, 47, 49-52
|| crypto/src/engine/arkworks/endomorphism.rs: 25, 41, 60, 73, 86, 99, 137-138, 141, 152, 164
|| crypto/src/engine/arkworks/ext_field.rs: 30
|| crypto/src/engine/arkworks/hashing/hash_to_curve.rs: 119, 122-123, 225-226, 232-235, 240, 289-290, 296, 379, 466-470, 474, 479
|| crypto/src/engine/arkworks/hashing/hash_to_field.rs: 107, 112-113
|| crypto/src/engine/arkworks/hashing/xmd_expander.rs: 45-48, 53
|| crypto/src/engine/arkworks/mod.rs: 46-48, 60-62, 77, 100, 128, 202, 205, 209, 212, 221, 226, 250, 258-260
|| crypto/src/engine/arkworks/zcash_format.rs: 55, 62-63, 97-99, 124, 128, 132, 134, 137-138, 150, 154, 159
|| crypto/src/engine/blst/g1.rs: 16, 87, 92-94
|| crypto/src/engine/blst/g2.rs: 16, 78, 83-85
|| crypto/src/engine/blst/mod.rs: 88, 90, 101, 103, 122, 149, 182, 210, 214
|| crypto/src/engine/blst/scalar.rs: 37-38, 40, 54-55
|| crypto/src/engine/both.rs: 83, 92
|| crypto/src/error.rs: 17-19, 21, 87-88, 113-114
|| crypto/src/group.rs: 26-27, 34-35, 44-45, 70-71, 76-77
|| crypto/src/hex_format.rs: 17, 53-56, 64, 67, 70, 73, 86-87, 103-105, 119, 123, 127
|| crypto/src/powers.rs: 44-46, 50-52
|| crypto/src/signature/identity.rs: 20, 23, 25, 40, 49, 88, 92, 95, 97, 105, 115, 119
|| crypto/src/signature/mod.rs: 184-185
|| crypto/src/transcript.rs: 56-57, 72, 75-77, 81-83
|| src/api/v1/auth.rs: 195, 197, 203, 206, 208, 214, 246-247, 264-265, 268-269, 273-274, 336-337, 344-345, 363-366, 378-379,
    ↪  435-438
|| src/api/v1/contribute.rs: 75, 103-104, 117, 125, 133, 151, 156
|| src/api/v1/error_response.rs: 25-26, 28, 30, 32, 39, 42, 69, 76, 86-88, 90, 103, 106-107, 117-120, 123
|| src/api/v1/info.rs: 52
|| src/api/v1/lobby.rs: 46, 119
|| src/io.rs: 34, 39, 54, 66, 73, 145-148
|| src/keys.rs: 38-39, 53-54, 70-73, 88, 96, 102
|| src/lib.rs: 57, 135, 172, 188, 191
|| src/lobby.rs: 184, 221, 249, 271, 352-353
|| src/main.rs: 5-6
|| src/sessions.rs: 27-28, 45-46, 86, 88
|| src/storage.rs: 58-59, 97-98, 103, 105-106, 111, 115-116, 121, 126, 132-133, 140-142, 144, 148
|| src/util.rs: 14, 20-21, 31-32, 38, 40
|| Tested/Total Lines:
|| crypto/src/batch_contribution.rs: 16/34 +14.33%
|| crypto/src/batch_transcript.rs: 27/35 +32.88%
|| crypto/src/contribution.rs: 8/15 -2.92%
|| crypto/src/engine/arkworks/endomorphism.rs: 63/74 -1.35%
|| crypto/src/engine/arkworks/ext_field.rs: 5/6
|| crypto/src/engine/arkworks/hashing/hash_to_curve.rs: 81/102
|| crypto/src/engine/arkworks/hashing/hash_to_field.rs: 23/26
|| crypto/src/engine/arkworks/hashing/xmd_expander.rs: 34/39
|| crypto/src/engine/arkworks/mod.rs: 116/135 -1.47%
|| crypto/src/engine/arkworks/zcash_format.rs: 61/76 -1.07%
|| crypto/src/engine/blst/g1.rs: 42/47 -2.13%
|| crypto/src/engine/blst/g2.rs: 41/46 -2.17%
|| crypto/src/engine/blst/mod.rs: 113/122 -0.29%
|| crypto/src/engine/blst/scalar.rs: 42/47 +1.27%
|| crypto/src/engine/both.rs: 52/54 +0.07%
|| crypto/src/error.rs: 0/8
|| crypto/src/group.rs: 14/24
|| crypto/src/hex_format.rs: 29/46 +4.22%
|| crypto/src/powers.rs: 13/19
|| crypto/src/signature/identity.rs: 43/55
|| crypto/src/signature/mod.rs: 42/44
|| crypto/src/transcript.rs: 32/41 -2.44%
|| src/api/v1/auth.rs: 160/188 +8.06%
|| src/api/v1/contribute.rs: 50/58 -6.82%
```

```
|| src/api/v1/error_response.rs: 28/49 +1.82%
|| src/api/v1/info.rs: 13/14 +42.86%
|| src/api/v1/lobby.rs: 39/41 +1.79%
|| src/io.rs: 48/57 +30.64%
|| src/keys.rs: 19/30
|| src/lib.rs: 61/66 +1.00%
|| src/lobby.rs: 138/144 +1.43%
|| src/main.rs: 0/2
|| src/oauth/ethereum.rs: 5/5
|| src/oauth/github.rs: 3/3
|| src/receipt.rs: 5/5
|| src/sessions.rs: 10/16
|| src/storage.rs: 50/69
|| src/util.rs: 17/24
82.69% coverage, 1543/1866 lines covered
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

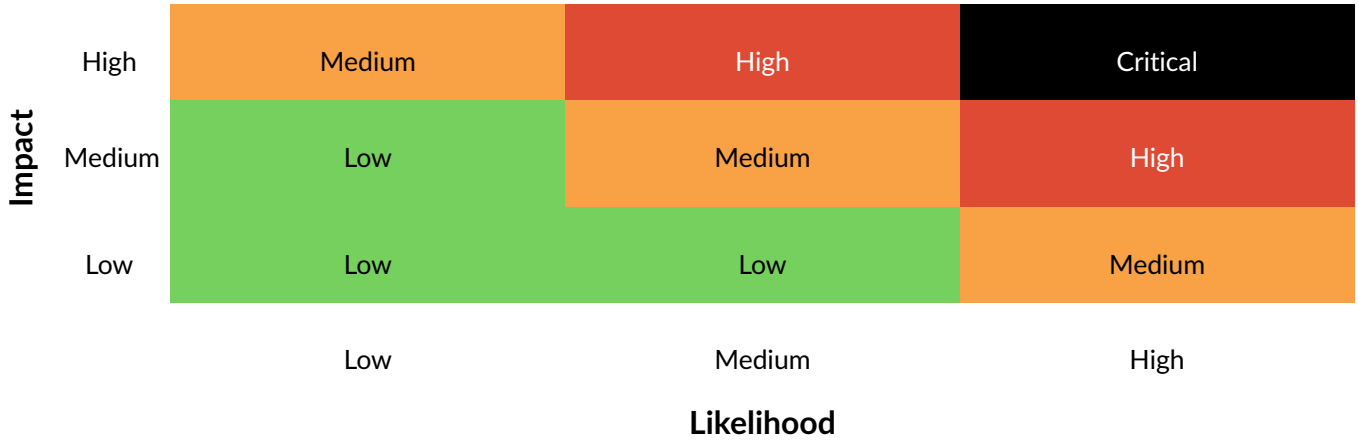

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.