|galois|

Formal Verification of the c-kzg library: Establishing the Basis

Final Report

Grant ID FY24-1544 — Ethereum Foundation

Brett Decker decker@galois.com
Ryan McCleeary mccleeary@galois.com
Roberto Saltini saltini.roberto@gmail.com
Thanh-Hai Tran thanhhai1302@gmail.com

Galois, Inc.
421 SW 6th Ave., Ste. 300
Portland, OR 97204

Contents

1. Final Report

The goal of this final report is to detail the outcomes, lessons learned, and path forward at the completion of this grant, to establish the basis for formally verifying that the c-kzg implementation is equivalent to its stated reference, the Polynomial Commitments section of the Python Deneb consensus specification (Python KZG Specification, hereafter) which encodes the KZG commitment scheme.

All software artifacts, the specifications, proofs, Makefiles, and documentation are found in this open-source GitHub repository.

During this grant, we were able to analyze the Python KZG Specification; create a formal specification for the Python KZG Specification in Cryptol; show correctness of our Cryptol specification through a test bench and visual inspection; create arithmetic properties proofs for our polynomial extension fields and curve in the Cryptol spec; and, finally, create proofs in the Software Analysis Workbench (SAW) for some of the ”bit-reversal permutation” functions defined in the Python KZG Specification to show equivalence of the corresponding Cryptol and c-kzg functions.

2. Results

The main results of our work are as follows:

1. A Cryptol specification of the Python KZG Specification
2. A Cryptol specification of the elliptic curve operations required by the Python KZG Specification. We modeled the py_ecc implementation. This includes proofs for some arithmetic properties of the polynomial extension fields (e.g. commutativity of multiplication)
3. A Cryptol specification for the serialization and deserialization of EC Points used by the Python KZG Specification. We modeled the py_ecc implementation
4. A test bench from randomized inputs to show correctness of the Cryptol specification and the Python KZG Specification
    - The test inputs were generated using the `:dumptests` REPL command in Cryptol. This is Cryptol’s built-in randomized, fuzz testing generator. We added these tests into our Cryptol source
    - We also created a Git patch file that adds these tests to the Python KZG Specification test bench
5. A SAW proof script that proves a couple of the ”bit-reversal permutation” functions in the Cryptol specification equivalent to the c-kzg functions
6. An additional, partial Cryptol specification for some of the functions in 1 written to match the Python KZG Specificatiothat did not complete formal verification with SAW in 5

Table 1: Python KZG Specification mapping to Cryptol

Category	Python KZG Specification	Cryptol
Bit-reversal permutation	is_power_of_two reverse_bits bit_reversal_permutation	is_power_of_two reverse_bits bit_reversal_permutation
BLS12-381 helpers	multi_exp hash_to_bls_field bytes_to_bls_field bls_field_to_bytes validate_kzg_g1 bytes_to_kzg_commitment bytes_to_kzg_proof blob_to_polynomial compute_challenge bls_modular_inverse div g1_lincomb compute_powers compute_roots_of_unity	g1_mul , g2_mul hash_to_bls_field bytes_to_bls_field bls_field_to_bytes validate_kzg_g1 bytes_to_kzg_commitment bytes_to_kzg_proof blob_to_polynomial compute_challenge bls_modular_inverse bls_div g1_lincomb compute_powers compute_roots_of_unity
Polynomials	evaluate_polynomial _in_evaluation_form	evaluate_polynomial _in_evaluation_form
KZG	blob_to_kzg_commitment verify_kzg_proof verify_kzg_proof_impl verify_kzg_proof_batch compute_kzg_proof compute_quotient _eval_within_domain compute_kzg_proof_impl compute_blob_kzg_proof verify_blob_kzg_proof verify_blob_kzg_proof_batch	blob_to_kzg_commitment verify_kzg_proof verify_kzg_proof_impl verify_kzg_proof_batch compute_kzg_proof compute_quotient _eval_within_domain compute_kzg_proof_impl compute_blob_kzg_proof verify_blob_kzg_proof verify_blob_kzg_proof_batch

There is a README in the repository for each of these. These READMEs include details on how to run all the tests, randomized test vectors, and proofs for repeatable results.

2.1. Cryptol Implementation of the Python KZG Specification

Table 1 gives a mapping of each function in the Python KZG Specification to the corresponding Cryptol function for reference.

3. Future Work

There is some more work needed to finalize the Cryptol specification:

1. The BLS helper function `hash_to_bls_field` is currently a placeholder
   (a) We need to include a SHA2-256 implementation in our repo and then call the hash function on the data bytes. Galois (and others) have previously implemented and verified SHA2 with Cryptol and SAW.
2. The following KZG functions have not been included in our Python test bench
   (a) `compute_challenge`
   (b) `compute_blob_kzg_proof`
   (c) `verify_blob_kzg_proof`
   (d) `verify_kzg_proof_batch`
   (e) `verify_blob_kzg_proof_batch`

The Cryptol specification took longer than we had assumed at proposal time. Galois has previously formally verified the BLST library with Cryptol and SAW. But upon further investigation, the Cryptol existing code was not well suited for our purposes. Thus, we had to create all of the code in BlsEC in order to have an fully executable specification. This was required for determining functional correctness with the test bench.

Because the Cryptol specification took longer than anticipated, we were only able to use SAW to formally verify some of the functions defined in the Bit-Reversal Permutations section of the Python KZG Specification.

While doing this we determined that the c-kzg code needs to be modified for the proofs to complete. The C function `bit_reversal_permutation` needs to be decomposed into two or more functions, which will then be small enough for us to prove equivalent to the functions in Cryptol. Thus, we have a software decomposition task that will be necessary to further the SAW formal verification. We have used this methodology in the first author's previous work (NFM 2021 and SPIN 2022).

The following describes the proof development process:

1. Create a SAW proof to show equivalence between a c-kzg function F and its corresponding Cryptol specification function S.
2. If the proof cannot complete, do the following:
   (a) Decompose F into subparts (split out the inner computation of loops into subfunctions)
   (b) Create a SAW proof of equivalence between F and decomposed F
   (c) Decompose S into similar subparts

i. We may additionally be required to rewrite S such that it models the computation in a more "imperative" programming style

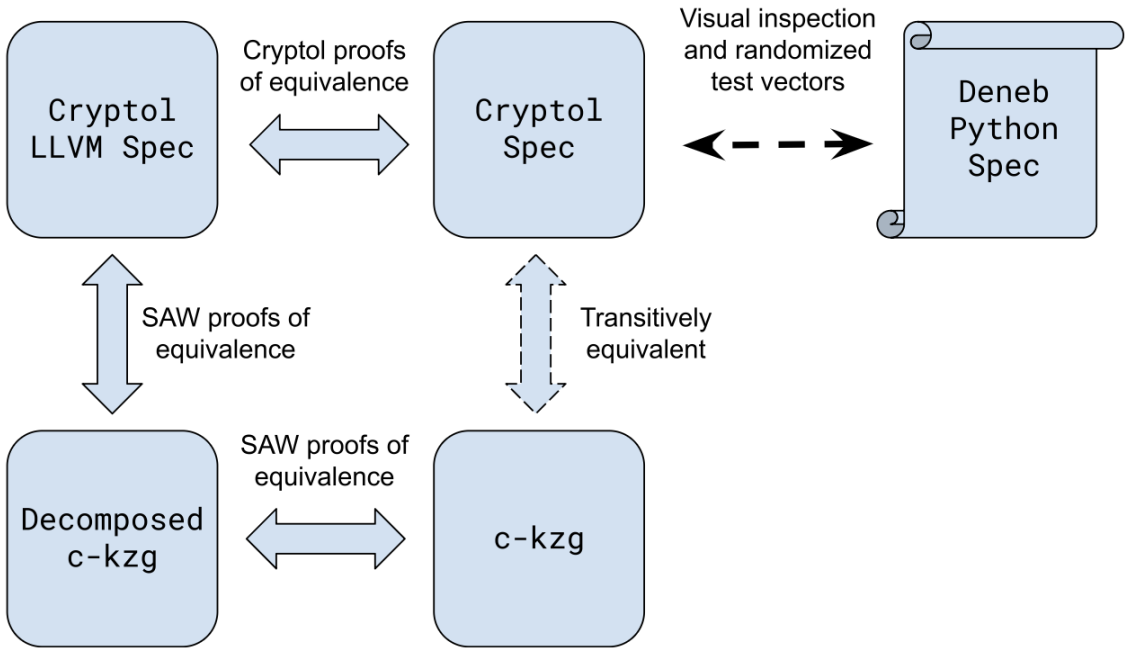
(d) If the proof does not complete, repeat a through c

3. Repeat 1 and 2 for next function to formally verify

In order to truly determine the feasibilty of using SAW to prove equivalence of the c-kzg library for the KZG commitment scheme functions to our Cryptol Specification we will need to use the above methodology on the remaining functions in Bit-Reversal Permutations, BLS12-381 helpers, and Polynomials, of the Python KZG Specification.

We will need to make some more process on the SAW proofs before we can give a reasonable estimate for the level of effort required to completely verify the c-kzg implementation. Another 4-5 weeks effort solely on the proofs would give us that ability to create a reasonable estimate for the completing the full verification.

The overall formal verification process can be visualized with the following diagram:



4. Conclusion

Overall, this grant was successful in funding the creation of a formal Cryptol specification of the Python KZG Specification that can be used for further formal verification activities in both Cryptol (which has proof language capabilities) and SAW.

The repository artifacts show a proof of concept from end-to-end for formally verifying the c-kzg library using SAW.