Report: CompPoly Guruswami-Sudan Algorithm in Lean

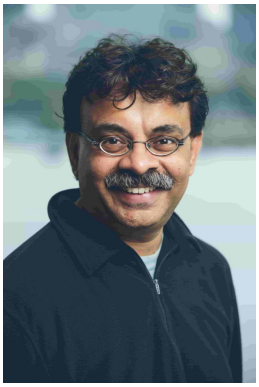
June 16, 2026

Formal Land

Contents

1 The Guruswami-Sudan algorithm

Venkatesan Guruswami



Madhu Sudan

Portrait sources: Venkatesan Guruswami's UC Berkeley profile; Madhu Sudan's Harvard profile.

Named after Venkatesan Guruswami and Madhu Sudan, the Guruswami-Sudan algorithm is a list-decoding algorithm for Reed-Solomon codes and related algebraic codes. Instead of requiring a unique closest codeword after errors, it returns all message polynomials consistent with the received word up to a chosen radius. This lets Reed-Solomon decoding go beyond the classical unique-decoding bound, and explains the two main algebraic stages used in this report: interpolation builds a bivariate constraint, and polynomial root search extracts the candidate messages.

2 Summary

Main contributor: Valerii Huhnin, Formal Land

Here is the description of the grant target from the roadmap:

> The target of this milestone is to implement the Guruswami-Sudan algorithm following what is done in `https://github.com/lambdaclass/lambdaworks/blob/main/examples/reed-solomon-codes/src/guruswami_sudan.rs`.
> As a second step, we must prove it correct in Lean.
> This implementation must come with tests and profiling, as extracted from the Lambdaclass implementation, showing the same results and reasonable efficiency. The performance of the two implementations should be compared.

We completed the core CompPoly side of this milestone in the upstream CompPoly repository. The work was implemented by Valerii Huhnin from the Formal Land team and merged into `Verified-zkEVM/CompPoly`.

The merged work gives CompPoly an executable, verified Guruswami-Sudan decoding stack. The implementation is decomposed into reusable contexts for interpolation, univariate finite-field root search, bounded bivariate root search, and the final filtered decoder. This keeps the decoder extensible: each backend can be swapped while preserving the same Lean soundness and completeness interface.

The delivered work covers the main building blocks needed by the decoder:

- deterministic univariate finite-field root search using a smooth multiplicative-subgroup splitter;

- bounded bivariate root search using Roth-Ruckenstein, plus an additional Alekhnovich backend;
- dense linear-system Guruswami-Sudan interpolation, plus a faster Lee-O'Sullivan interpolation backend;
- the assembled executable decoder, including the algebraic `gsCore` pipeline and the distance-filtering `gsFilteredCore` pipeline;
- regression tests and benchmark groups for root search, interpolation, candidate filtering, and full decoder paths.

The original implementation was first opened as one large pull request, PR 237. After review, it was split into smaller pull requests to make the code easier to review and merge. PR 237 is therefore closed unmerged, while the completed pieces listed below are merged.

3 What we get from it

With this work, CompPoly gains a verified implementation of a non-trivial coding-theory algorithm rather than only isolated polynomial operations:

- The Guruswami-Sudan decoder is available as executable Lean code with explicit correctness contracts.
- The decoder has a modular architecture, so interpolation and root-search algorithms can be improved independently.
- The reusable univariate and bivariate root-search layers can support other polynomial algorithms in CompPoly or downstream libraries.
- Dense interpolation provides a simple reference backend, while Lee-O'Sullivan gives a more practical interpolation path.
- Benchmark groups and regression tests give maintainers a way to compare backends and detect future performance regressions.
- The work is ready for downstream integration: an ArkLib integration pull request was opened as Verified-zkEVM/ArkLib PR 574.

4 Merged pull requests

All pull requests listed below are merged in `Verified-zkEVM/CompPoly`.

- PR 244, merged 2026-06-09: *feat(univariate): smooth-subgroup splitter root search*. Executable finite-field root-search API, root-product computation, deterministic smooth-subgroup splitter, KoalaBear schedule facts, and correctness proofs.
- PR 245, merged 2026-06-10: *feat(bivariate): Roth-Ruckenstein root search for GS decoding*. Guruswami-Sudan bivariate infrastructure, Hasse derivatives, weighted-degree helpers, bounded bivariate root-search API, Roth-Ruckenstein backend, and proofs.
- PR 246, merged 2026-06-11: *feat(bivariate): dense linear interpolation for GS decoder*. Dense linear algebra, homogeneous-kernel solver, shared interpolation context API, dense interpolation backend, and proofs.
- PR 247, merged 2026-06-11: *feat(bivariate): Guruswami-Sudan decoder*. Assembled `gsCore` and `gsFilteredCore`, executable wrappers, dense/Roth KoalaBear implementations, soundness and completeness proofs, tests, and benchmarks.
- PR 250, merged 2026-06-12: *feat(bivariate): Lee-O'Sullivan interpolation for GS decoder*. Polynomial-matrix infrastructure, Mulders-Storjohann shifted-row reduction, Lee-O'Sullivan interpolation backend, proofs, tests, and benchmarks.

- PR 251, merged 2026-06-11: *feat(bivariate): Alekhnovich root search for GS decoder*. Alekhnovich bounded bivariate root-search backend, correctness proofs, tests, and Roth-Ruckenstein comparison benchmarks.

5 Details

5.1 Decoder architecture

The decoder is organized around explicit context interfaces:

- `GSInterpContext` packages a Guruswami-Sudan interpolation backend and proves that it returns a valid interpolation witness.
- `FieldRootContext` packages finite-field root search for nonzero univariate polynomials.
- `GSRootContext` packages bounded-degree bivariate root search.
- `GSFilteredCoreContext` exposes the filtered decoder with its own soundness and completeness contracts.

The high-level decoder has three stages. First, interpolation turns the received points and Guruswami-Sudan parameters into a nonzero bivariate witness Q. Second, bivariate root search finds all degree-bounded message polynomials p such that $Q(X, p(X)) = 0$. Third, the packed filter evaluates the candidates on the received word and keeps only the polynomials within the requested mismatch radius.

PR 247 assembled this into:

- `gsCore`, the algebraic pipeline that interpolates a witness and finds bounded-degree message-polynomial roots;
- `gsFilteredCore`, the filtered pipeline that keeps only candidates within the requested mismatch radius;
- top-level executable wrappers and concrete dense/Roth KoalaBear implementation bundles;
- Lean soundness and completeness proofs for the assembled core and filtered core;
- regression tests for the core pipeline, filtered pipeline, and packed candidate filtering.

5.2 Univariate finite-field root search

PR 244 added executable root search for univariate polynomials over finite fields. This is needed by Roth-Ruckenstein root search, but is also a reusable component in its own right.

The implementation handles zero, constant, and linear polynomials directly. For higher-degree inputs, it computes the field-root product by reducing X^q modulo the input polynomial, rather than materializing $X^q - X$. It then uses a splitter to break the squarefree product of linear factors and extracts roots that are validated against the original polynomial.

The merged backend is deterministic and targets fields whose multiplicative group has a certified smooth schedule. This is a good fit for KoalaBear: its multiplicative group order is $2^{24} \cdot 127$, so the smooth-subgroup splitter can refine the group through a small-factor schedule. The PR proves that, under the splitter contract, the returned roots are exactly the roots of a nonzero input polynomial.

5.3 Bounded bivariate root search

PR 245 added the first bounded bivariate root-search backend for Guruswami-Sudan decoding. It introduced the shared Guruswami-Sudan bivariate infrastructure, including Hasse derivatives, composition in Y, weighted-degree monomials, and helper lemmas.

The Roth-Ruckenstein backend reconstructs candidate message polynomials coefficient by coefficient. At each step, it reduces the bivariate problem to a univariate root-search call, then validates candidates against the exact degree and composition checks. The backend is proved sound and complete relative to the supplied univariate `FieldRootContext`.

PR 251 added Alekhnovich root search as an alternative backend. The PR includes the recursive algorithm, supporting lemmas, a `GSRootContext` instance, regression tests, and benchmarks comparing Alekhnovich with Roth-Ruckenstein. In the benchmarked cases, Alekhnovich is the faster bivariate root-search backend, while end-to-end decoder performance is usually dominated by interpolation rather than bivariate root search.

5.4 Guruswami-Sudan interpolation

PR 246 added dense linear interpolation for the Guruswami-Sudan interpolation problem. This backend builds the Hasse-constraint matrix over the weighted-degree monomial basis, solves for a nonzero homogeneous kernel vector, and turns that vector into an interpolation witness.

The PR also added a small dense linear-algebra layer: row operations, RREF shape tracking, homogeneous-kernel witnesses, and correctness proofs for the dense row-operation and kernel-solving routines. The dense interpolation backend is proved sound and complete and is packaged as a `GSInterpContext`.

PR 250 added Lee-O'Sullivan interpolation. This backend builds an interpolation module from the vanishing polynomial and coefficient-form interpolation polynomial, then reduces a shifted polynomial-row matrix using Mulders-Storjohann reduction. The selected least-shifted-degree row is normalized through the same interpolation-witness policy used by the rest of the decoder.

The Lee-O'Sullivan PR includes polynomial-matrix infrastructure, shifted-row reduction, correctness proofs for the reduction layer and interpolation backend, regression tests, and benchmarks comparing Lee-O'Sullivan with dense linear-system interpolation. The development notes report that Lee-O'Sullivan was generally faster than Koetter in Valerii's benchmarks, both for messages with no errors and with many errors. The merged accelerated interpolation backend is Lee-O'Sullivan.

5.5 Tests and benchmarks

The merged work adds regression tests for:

- univariate finite-field root products and generic field enumeration;
- KoalaBear root contexts;
- Hasse derivatives and multiplicity checks;
- composition in Y;
- dense linear algebra and dense interpolation;
- Lee-O'Sullivan interpolation;
- Roth-Ruckenstein and Alekhnovich bivariate root search;
- `gsCore`, `gsFilteredCore`, and packed mismatch filtering.

The benchmark suite was extended with Guruswami-Sudan groups covering interpolation-system construction, interpolation solving, Lee-O'Sullivan setup, Hasse multiplicity checking, composition in Y, bivariate roots, candidate filtering, and full decoder paths. The rows compare dense and Lee-O'Sullivan interpolation where applicable, Roth-Ruckenstein and Alekhnovich root search, and KoalaBear field variants.

5.6 Profiling and comparison with Lambdaworks

We treated the Lambdaworks code as a useful reference point: it is clear, compact, and well suited to explaining the algorithm and validating expected behavior on representative examples.
The two implementations have different goals. The Lambdaworks example is educational and does not try to use the fastest available algorithms for every subproblem. In particular, its interpolation and root-finding components are not the best known choices in terms of algorithmic complexity. The CompPoly implementation instead separates these subproblems behind verified interfaces, and includes faster backends such as Lee-O'Sullivan interpolation and certified finite-field root search. For large inputs, this gives the Lean implementation better scaling.
The Lambdaworks root-finding path uses heuristics, which means that it is not complete in the general case. The Lean root-search backends are proved against explicit completeness contracts, so a direct end-to-end comparison would not be comparing two implementations with the same specification.
During this work, we found an edge case in the Lambdaworks example when the message length is one, where the code can divide by zero. We have been in contact with the LambdaClass team and have kept them informed about the CompPoly development and the (small) issue we found. We are grateful for their reference implementation, which helped anchor the implementation and testing work.

6 Follow-up work

Several additional pieces were implemented or opened but are not counted as merged deliverables in this report:

- PR 237, the original large Guruswami-Sudan pull request, was closed unmerged after being split into smaller PRs.
- PR 253 adds Shoup's deterministic univariate root-search algorithm and binary-field infrastructure. It is open.
- PR 254 adds a Las Vegas randomized Cantor-Zassenhaus root-search backend. It is open.
- PR 255 adds approximant-basis and hybrid interpolation backends. It is open.
- Verified-zkEVM/ArkLib PR 574 integrates CompPoly's executable Guruswami-Sudan decoder into ArkLib. It is open.

These follow-ups point to the natural next improvements: broader finite-field root search, faster interpolation on long and heavily corrupted codes, and downstream use from ArkLib.

7 Files of interest

The main files and folders touched by the merged work are:

- `CompPoly/Univariate/Roots/`: univariate finite-field root-search API, root-product construction, extraction, enumeration, smooth-subgroup splitter, and correctness.
- `CompPoly/Bivariate/GuruswamiSudan/`: top-level Guruswami-Sudan implementation, core/filter pipeline, executable wrappers, contexts, and shared polynomial infrastructure.
- `CompPoly/Bivariate/GuruswamiSudan/Interpolation/Dense/`: dense interpolation backend and correctness.
- `CompPoly/Bivariate/GuruswamiSudan/Interpolation/LeeOSullivan/`: Lee-O'Sullivan interpolation backend and correctness.

- `CompPoly/Bivariate/GuruswamiSudan/Root/RothRuckenstein/`: Roth-Ruckenstein bounded bivariate root search and correctness.
- `CompPoly/Bivariate/GuruswamiSudan/Root/Alekhnovich/`: Alekhnovich bounded bivariate root search and correctness.
- `CompPoly/LinearAlgebra/Dense/`: dense row operations, RREF semantics, homogeneous-kernel solving, and proofs.
- `CompPoly/LinearAlgebra/PolynomialMatrix/`: polynomial-matrix infrastructure and Mulders-Storjohann shifted-row reduction.
- `bench/CompPolyBench/Bivariate/GuruswamiSudan/`: Guruswami-Sudan benchmark groups.
- `tests/CompPolyTests/Bivariate/GuruswamiSudan/`: Guruswami-Sudan regression tests.

8 Thanks

We thank the Ethereum Foundation and the Verified-zkEVM maintainers for the opportunity to contribute this verified Guruswami-Sudan implementation to CompPoly. We are happy to discuss any details in this report and the project.