The Journal of Open Source Software

Komodo: Cryptographically-proven Erasure Coding

Antoine Stevan [1*], **Jonathan Detchart** [1*], **Tanguy Pérennou** [1], and **Jérôme Lacan** [1]

1 ISAE-SUPAERO, France * These authors contributed equally.

DOI: 10.21105/joss.09791

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @cassiersg
- @ankitabanerjee015

Submitted: 04 December 2025
Published: 06 May 2026

Summary

We present **Komodo**, a library that allows one to (**1**) encode data with forward erasure-code (FEC) techniques such as Reed-Solomon encoding, (**2**) prove the resulting *shards* with cryptographic protocols, (**3**) verify their validity on the other end of any distributed network, and (**4**) decode the original data from a subset of said *shards* (Stevan et al., 2025, 2024). The library is implemented in the *Rust* programming language and available under the MIT license on the ISAE-SUPAERO GitLab instance[1] with a mirror on GitHub[2]. **Komodo** should be of interest for people willing to explore the field of cryptographically-proven *shards* of data in distributed systems or data availability sampling settings.

A *shard* of data is said to be *valid* if it has been generated by a correct FEC encoding of the data.

Komodo implements a protocol API to achieve the following on any input data in a distributed network:

- `encode`: encodes data into *shards* with a Reed-Solomon (k, n)-code. This adds redundancy to the data, making the network more resilient to failure, fragmentation, partitioning, loss or corruption.
- `commit`: commit the input data into a *commitment* common to all *shards*.
- `prove`: generate one proof for all n *shards* with one of three available cryptographic protocols (see below for more information). This extra information guarantees a given *shard* is valid, except with negligible probability.
- `verify`: verifies a *shard* individually for its validity using the data *commitment* and the *shard* proof. This allows to discriminate invalid or corrupted *shards* without any decoding attempt. Without this shard-level verification step, it is impossible to know if a *shard* is valid until the decoding step. Then, when decoding fails, it is not possible to know which *shards* were invalid, leading to a *try-and-error* process that is not scalable.
- `decode`: decodes the original data using any subset of k independent and valid *shards*.

Komodo provides the three following cryptographic protocols:

- **KZG+** (short for the first letters of the authors' names): based on Kate et al. (2010) and its multi-polynomial extension (Boneh et al., 2020)
- **aPlonK** (short for ***A**ggregated **P**ermutations over **L**agrange-bases for **O**ecumenical **N**oninteractive arguments of **K**nowledge*): based on **PlonK** (Gabizon et al., 2019) and **aPlonK** (Ambrona et al., 2023)
- **Semi-AVID** (short for *Semi **A**synchronous **V**erifiable **I**nformation **D**ispersal*): based on **Semi-AVID-PR** (Nazirkhanova et al., 2022)

[1] GitLab source code: https://gitlab.isae-supaero.fr/dragoon/komodo
[2] GitHub mirror for issues and pull requests: https://github.com/dragoon-rs/komodo

Komodo is based on the Arkworks library (arkworks contributors, 2022) which provides implementations of elliptic curves, fields and polynomial algebra.

Statement of need

The verification mechanisms provided by **Komodo** can help in implementing verifiable information dispersal or data availability schemes in a distributed storage system or in a blockchain.

For instance, in a distributed storage system, nodes can use **Komodo** to encode incoming data into shards; commit them to the initial data; compute individual shard proofs; bundle shards, commitments and proofs into messages; and distribute these messages across the network. Any node that receives a block can then verify its validity before storing or forwarding it. This ensures data robustness and trustworthiness when a node eventually decodes the data from the shards bundled in valid messages.

Several blockchain projects use some of these mechanisms to verify data availability:

- Celestia uses 2D Reed-Solomon-based codes with Merkle trees (Al-Bassam et al., 2019; rsmt2d contributors, 2026)
- Ethereum Danksharding (*What Is Proto-Danksharding?*, 2024) uses 2D Reed-Solomon with KZG polynomial commitments

Such projects could use or extend Komodo to benchmark different schemes.

A few libraries provide similar functionalities, with a few gaps filled by **Komodo**:

- The `arkworks` ecosystem (arkworks contributors, 2022) is probably the closest library, providing many of the necessary building blocks involved in Data Availability Sampling: prime fields, possibly paired with elliptic curves like BLS12-381 or BN254 among many others, and linear algebra operations like polynomial operations and polynomial commitment. On top of those features, **Komodo** adds Reed-Solomon encoding, tightly integrated with proof generation.
- The Rust implementation of Reed-Solomon erasure coding (rust-rse contributors, 2021) provides mechanisms to encode and decode data into raw shards, using elements of finite fields $\mathbb{F}_{2^8}$ or $\mathbb{F}_{2^{16}}$, containing respectively 2^8 and 2^{16} elements. **Komodo** adds the proving mechanisms, and makes it possible to use elements from `arkworks`' prime fields.
- **Komodo** also adds a unified benchmarking API, allowing to compare different combinations of prime fields, elliptic curves and polynomial commitment schemes, as we did in two publications (Stevan et al., 2025, 2024). Finally, a modular design allows one to extend **Komodo** with new polynomial commitment schemes or new encoding methods, which performance can be evaluated in the same benchmarking conditions.

Some measurements

Building on the work from (Stevan et al., 2024), we have conducted some measurements of the performance of the three methods. All experiments were run on a laptop with x86-64 Intel Core i7-12800H (14 cores / 20 threads, 0.4–4.8 GHz) system with a 3-level cache hierarchy (L1d 544 KiB, L1i 704 KiB, L2 11.5 MiB, L3 24 MiB) and a single NUMA node. Only one thread was used for all experiments.

The time to run `commit`, `prove`, and `verify` has been measured for $k = 8$ and a code rate $\rho = \frac{1}{2}$, i.e., $n = 16$, on the BN-254 elliptic curve, and for small and large input data. The encode and decode steps are the same for all protocols and thus have been omitted from the analysis below. `encode` involves a matrix multiplication which takes around $10\mu s$ with $k = 8$. `decode` has an extra matrix inversion of size $k = 8$ taking around $250\mu s$.

All raw results have been pushed to a separate dedicated repository[3] and all the pre-compiled figures are available on the *GitLab pages* instance of ISAE SUPAERO[4].

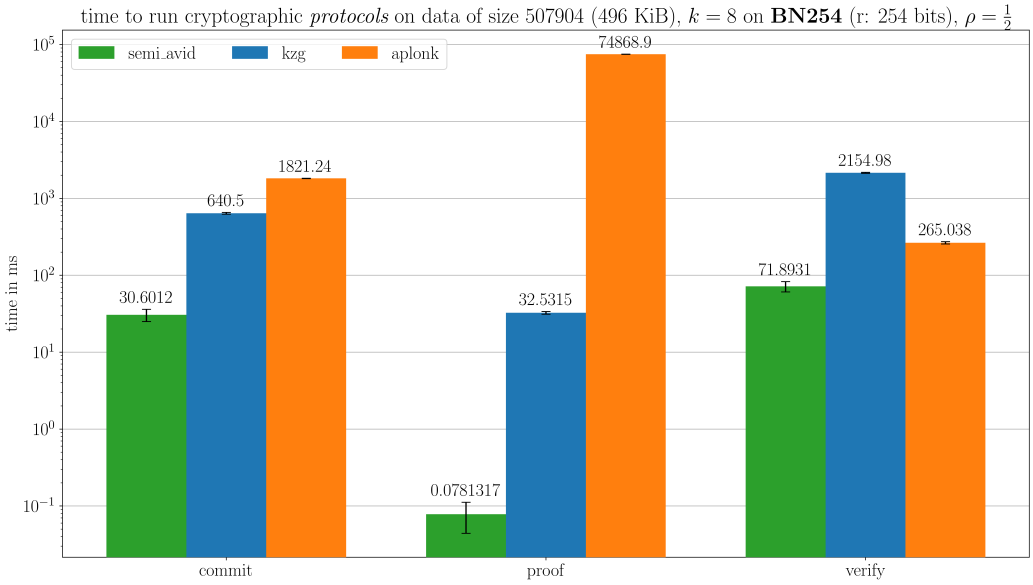

Figure 1: Performance for small files. Average over 10 runs.

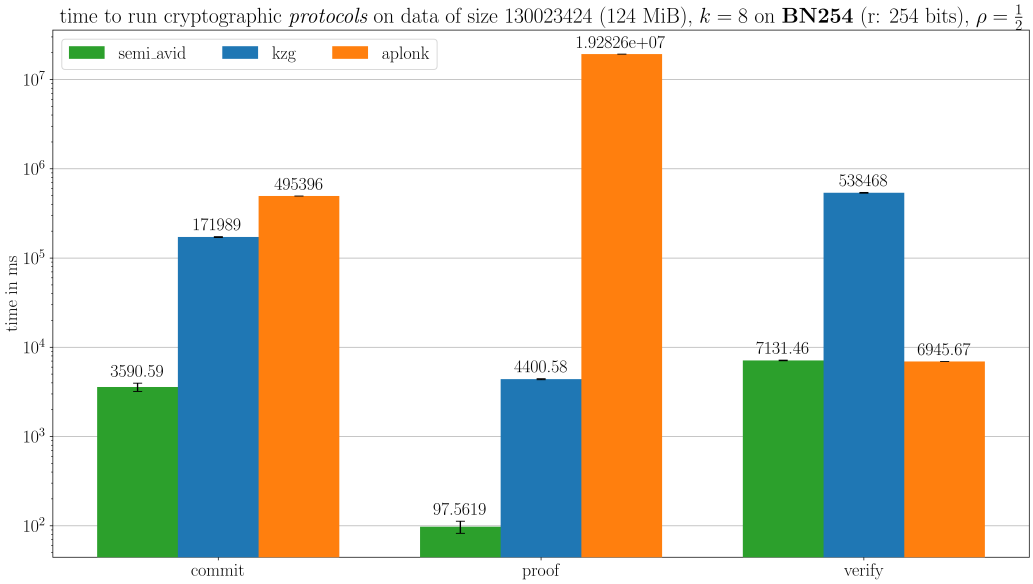

Figure 2: Performance for large files. Average over 10 runs for **KZG+** and **Semi-AVID**. Only 1 run for **aPlonK**.

Figure 1 shows that **Semi-AVID** is the best for committing, proving and verifying small files. Figure 2 shows that **aPlonK** is slightly better for verifying large files but still suffers from performance orders of magnitude worst than **Semi-AVID** for committing and proving. Both figures show that **KZG+** is neither good nor too bad.

[3] results repository: https://gitlab.isae-supaero.fr/dragoon/komodo-benchmark-results
[4] pre-compiled results figures: https://gitlab-pages.isae-supaero.fr/dragoon/komodo-benchmark-results/

Additional information

Komodo is fully written in *Rust* and thus all dependencies are taken care of by *Cargo* and `Cargo.toml`.

Contact

- by email: `firstname.lastname@isae-supaero.fr`
- ticket tracker: https://github.com/dragoon-rs/komodo/issues
- contributions: https://github.com/dragoon-rs/komodo/pulls

Acknowledgements

This work was supported by the Defense Innovation Agency (AID) of the French Ministry of Defense through the Research Project DRAGOON: Dependable distRibuted storAGe fOr mObile Nodes (2022 65 0082).

References

Al-Bassam, M., Sonnino, A., & Buterin, V. (2019). *Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities*. https://doi.org/10.48550/arXiv.1809.09044

Ambrona, M., Beunardeau, M., Schmitt, A.-L., & Toledo, R. R. (2023). *aPlonK: Aggregated PlonK from multi-polynomial commitment schemes*. 195–213. https://doi.org/10.1007/978-3-031-41326-1_11

arkworks contributors. (2022). *arkworks zkSNARK ecosystem*. https://arkworks.rs

Boneh, D., Drake, J., Fisch, B., & Gabizon, A. (2020). Efficient polynomial commitment schemes for multiple points and polynomials. *Cryptology ePrint Archive*. https://eprint.iacr.org/2020/081

Gabizon, A., Williamson, Z. J., & Ciobotaru, O. (2019). Plonk: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*. https://eprint.iacr.org/2019/953

Kate, A., Zaverucha, G. M., & Goldberg, I. (2010). Constant-size commitments to polynomials and their applications. *International Conference on the Theory and Application of Cryptology and Information Security*, 177–194. https://doi.org/10.1007/978-3-642-17373-8_11

Nazirkhanova, K., Neu, J., & Tse, D. (2022). Information dispersal with provable retrievability for rollups. *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, 180–197. https://doi.org/10.1145/3558535.3559778

rsmt2d contributors. (2026). Go implementation of two dimensional Reed-Solomon Merkle tree data availability scheme. In *GitHub repository*. GitHub. https://github.com/celestiaorg/rsmt2d

rust-rse contributors. (2021). Rust implementation of Reed-Solomon erasure coding. In *GitHub repository*. GitHub. https://github.com/rust-rse/reed-solomon-erasure

Stevan, A., Lavaur, T., Lacan, J., Detchart, J., & Pérennou, T. (2025). Assessing the efficiency of polynomial commitment schemes in erasure code-based data distribution. *International Conference on Information Systems Security and Privacy*, 274–300. https://doi.org/10.1007/978-3-031-89518-0_13

Stevan, A., Lavaur, T., Lacan, J., Detchart, J., & Pérennou, T. (2024). Performance

evaluation of polynomial commitments for erasure code based information dispersal. *10th International Conference on Information Systems Security and Privacy*. https://doi.org/10.5220/0012377900003648

What is proto-danksharding? (2024). https://ethereum.org/roadmap/danksharding/#what-is-protodanksharding