Proving Poseidons with GKR

1 Introduction

Inspired by the recent GKR implementation of V. Buterin (blogpost: [1]) to prove Poseidons [2], we suggest 2 modifications that, together, provide:

- A proving speed of 1.25M Poseidons / s on a M4 Max laptop (using also Poseidon for the merkle tree in thr PCS). This corresponds to $\approx$ 15x overhead versus plaintext.
- Using the degree 5 extension of KoalaBear (providing 128 bits of security on the sumchecks, without grinding)
- With a proof size of 225 KiB (using WHIR, "at capacity", without merkle pruning).
- Potentially better guarantees against the GKR attack [3] ?

2 Classical GKR

We want to prove validity of $N = 2^n$ Poseidon2 permutations over 16 field elements (typically KoalaBear). There are 4 initial full rounds, 20 partial rounds and 4 final full rounds (28 rounds in total). The sbox is $x \to x^3$.

Every GKR layer i is composed by 16 multilinear polynomials: $M_i^0, \dots, M_i^{15}$, each in n variables.

- The prover commits to the input: $M_0^0, \dots, M_0^{15}$.
- The verifier sends a random point $x_{28} \in \mathbb{F}_q^n$.
- The prover sends the 16 claimed output evaluations: $M_{28}^0(x_{28}), \dots, M_{28}^{15}(x_{28})$.

  Each of these 16 values has a nice "sumcheckable" expression that depends on the previous layer:

  - $M_{28}^0(x_{28}) = \sum_{i \in \{0,1\}^n} eq(i, x_{28}) \cdot F_{28}^0(M_{27}^0(i), \dots, M_{27}^{15}(i))$
  - $\dots$
  - $M_{28}^{15}(x_{28}) = \sum_{i \in \{0,1\}^n} eq(i, x_{28}) \cdot F_{28}^{15}(M_{27}^0(i), \dots, M_{27}^{15}(i))$

  Where $F_{28}^0, \dots, F_{28}^{15}$ are multivariate polynomials of degree 3 (Precisely, each $F_{28}^i(M_{27}^0, \dots, M_{27}^{15})$ is a linear combination of $(M_{27}^0 + c_{28}^0)^3, \dots, (M_{27}^{15} + c_{28}^{15})^3$, where $c_{28}^0, \dots, c_{28}^{15}$ are the round constants for the last full round).

  We don't run 16 times the sumcheck protocol. Instead, we sample a random batching challenge $\alpha_{28} \in \mathbb{F}_q$ and run a sumcheck for:

$$\sum_{k=0}^{15} \alpha^k \cdot M_{28}^k(x_{28}) = \sum_{i \in \{0,1\}^n} eq(i, x_{28}) \cdot F_{28}^{\alpha}(M_{27}^0(i), \dots, M_{27}^{15}(i))$$

  With $F_{28}^{\alpha} = \sum_{k=0}^{15} \alpha^k \cdot F_{28}^k$

  Let $x_{27} \in \mathbb{F}_q^n$ be the random challenges sent during sumcheck.
- The prover sends the 16 claimed evaluations: $M_{27}^0(x_{27}), \dots, M_{27}^{15}(x_{27})$, enabling to finish the verification of the previous sumcheck. The validity of these 16 fresh values are, again reduced via sumcheck to claims on the layer 26, and so on.

- ...
- Finally, the claims $M_0^0(x_0), \ldots, M_0^{15}(x_0)$ can be answered via the commitment to the inputs.

3 Suggestion A: Univariate skip

3.1 Description

Idea: Apply the univariate skip [4], at each layer of GKR.

- k is the number of "skips". In practice, $k \approx 3$ ($k = 1$ is equivalent to the classical GKR described above).
- Each M_i^j is now a polynomial in $n+1-k$ variables, of degree 2^k in the first variable, multilinear in the others.
- x_{28} is now sampled from $\mathbb{F}_q^{n+k-1}$.
- The first sumcheck we run is now:

$$\sum_{k=0}^{15} \alpha^k \cdot M_{28}^k(x_{28}) = \sum_{i \in D \times \{0,1\}^{n-k}} \delta(i, x_{28}) \cdot F_{28}^{\alpha}(M_{27}^0(i), \ldots, M_{27}^{15}(i))$$

  With:

  - $D = \{0, 1, 2, \ldots, 2^k - 1\}$
  - $\delta((u_1, \ldots, u_{n+1-k}), (v_1, \ldots, v_{n+1-k})) = eq_k(u_1, v_1) \cdot eq((u_2, \ldots, u_{n+1-k}), (v_2, \ldots, v_{n+1-k}))$
  - eq_k is the unique polynomial in 2 variables, each of degree $2^k - 1$, such that for $(u, v) \in D^2$ $eq_k(u, v) = 1$ if $u = v$, 0 otherwise.
- and so on...

3.2 Advantages

- Replaces multiplications in the extension field by multiplications in the base field.
- Reduced memory throughput (less memory reads / writes).
- Empirically: 50% faster on CPU.

4 Suggestion B: Commit the cubes in the partial rounds

4.1 Describtion

This suggestion comes from the observation the prover work in this GKR is (at least on a M4 mac) memory bounded: the sumcheck for the full rounds are almost as costly as the sumchecks for the partial rounds.

The idea here is to replace the 20 GKR layers for the partial rounds by a single sumcheck. But doing so, we would end up with an impractical expression of degree 3^{20}. The partial rounds only apply the sbox (cube map) to the first element of the state. Let's denote by $M_5^{cube}, \ldots, M_{23}^{cube}$ the corresponding cubes:

- $M_5^{cube} := (M_4^0 + c_5)^3$
- ...
- $M_{23}^{cube} := (M_{22}^0 + c_{23})^3$

Where $c_5, \ldots, c_{23}$ denote the partial round constants (the last one c_{24} doesn't appear here, instead it will be in the sumcheck bellow).

- These 19 polynomials are committed at the beginning, alongside the input $M_0^0, \dots, M_0^{15}$.

  The key observation is that, once the cubes are committed, the constraints become linear in the partial rounds (This idea is not new, for instance it is implemented in Plonky3 AIR arithmetization for Poseidon).

- The first 4 sumchecks for the final full rounds are performed without modification.

- Now comes the partial rounds, for which we have the sumcheckable expression:

$$\sum_{k=0}^{15} \alpha^k \cdot M_{24}^k(x_{24}) = \sum_{i \in \{0,1\}^n} eq(i, x_{24}) \cdot F^{\alpha}_{partial}(M_4^0(i), \dots, M_4^{15}(i), M_5^{cube}(i), \dots, M_{23}^{cube}(i)) \quad (1)$$

  Where $F^{\alpha}_{partial}$ has degree 3.

  But we must also ensure that committed cubes are correctly constructed. The following expressions should be verified:

  - $M_5^{cube}(X) = \sum_{i \in \{0,1\}^n} eq(i, X) \cdot F_5^{cube}(M_4^0(i), \dots, M_4^{15}(i))$
  - $M_6^{cube}(X) = \sum_{i \in \{0,1\}^n} eq(i, X) \cdot F_6^{cube}(M_4^0(i), \dots, M_4^{15}(i), M_5^{cube})$
  - $\dots$
  - $M_{23}^{cube}(X) = \sum_{i \in \{0,1\}^n} eq(i, X) \cdot F_{23}^{cube}(M_4^0(i), \dots, M_4^{15}(i), M_5^{cube}, \dots, M_{22}^{cube})$

  With $F_5^{cube}, \dots, F_{23}^{cube}$ 19 multivariate polynomials of degree 3 (For instance: $F_5^{cube} = (M_4^0 + c_5)^3$, $\dots$).

  To ensure correctness of the expressions, we evaluate them at the random challenge x_{24}. Finally, we can batch altogether these sums with (1) to obtain a single sumcheck. To conclude, the only sumcheck we run to ensure correctness of the partial rounds has the form:

$$\sum_{k=0}^{15} \alpha^k \cdot M_{24}^k(x_{24}) + \sum_{k=0}^{18} \alpha^{k+16} \cdot M_{5+k}^{cube}(x_{24})$$
$$= \sum_{i \in \{0,1\}^n} eq(i, x_{24}) \cdot F^{\alpha}_{partial/cubes}(M_4^0(i), \dots, M_4^{15}(i), M_5^{cube}(i), \dots, M_{23}^{cube}(i))$$

  Where $F^{\alpha}_{partial/cubes}$ has degree 3.

  (For soundness, α must be sampled after the prover has sent the claimed evaluations on the cubes: $M_5^{cube}(x_{24}), \dots, M_{23}^{cube}(x_{24})$)

4.2 Advantages

This suggestion trades 20 sumchecks (of degree 3, for the partial rounds) against one, bigger sumcheck (also of degree 3) and 19 committed field elements per Poseidon. Empirically this brings $\approx 25\%$ performance improvement, on a M4 mac.

One second advantage of this idea, is that committing to intermediate values in the GKR circuit intuitively increases the defense against the known GKR attack [3]? (To be confirmed)

References

[1] V. Buterun, "A gkr tutorial," 2025. [Online]. Available: https://vitalik.eth.limo/general/2025/10/19/gkr.html

[2] L. Grassi, D. Khovratovich, and M. Schofnegger, "Poseidon2: A faster version of the poseidon hash function," Cryptology ePrint Archive, Paper 2023/323, 2023. [Online]. Available: https://eprint.iacr.org/2023/323

[3] D. Khovratovich, R. D. Rothblum, and L. Soukhanov, "How to prove false statements: Practical attacks on fiat-shamir," Cryptology ePrint Archive, Paper 2025/118, 2025. [Online]. Available: https://eprint.iacr.org/2025/118

[4] A. Gruen, "Some improvements for the PIOP for ZeroCheck," 2024. [Online]. Available: https://eprint.iacr.org/2024/108