ZAMA

INCREMENTALLY VERIFIABLE COMPUTATION AND VERIFIABLE FHE

FEASIBILITY AND APPLICABILITY

Michael Walter

January 31, 2024

1 Introduction

This document is part of our effort to develop a SNARK prototype that is able to prove the correctness of a PBS evaluation in reasonable time. For an overview of our approach we refer to [Wal23a, Wal23b]. In this document we evaluate the practicality of applying Incrementally Verifiable Computation (IVC) to prove the correct evaluation of the blind rotation.

2 Incrementally Verifiable Computation

The concept of IVC goes back to the work of [Val08]. The idea is to prove the correctness of a long computation step by step and be able to verify this computation at any intermediate point, even before the entire computation is completed. Since most SNARKs prove computation in the circuit model, one may also think of IVC as a way to prove loops more efficiently than rolling out the entire loop into a large circuit. This is particularly useful when the prover time is super-linear as it allows to break up the proof of the large circuit of size nC into n proofs for small circuits of size C. In our setting, we target the blind rotation for the application of IVC as it is essentially a large loop.

In the literature, there are two main approaches discussed to achieve IVC, one based on recursive proofs and one based on folding. We will describe the high level idea of each of them below and then discuss their advantages and disadvantages for our setting.

3 Recursion

The concept of recursion is based on the observation that the computation of the verifier is just a circuit that can be the input to a SNARK itself. So after evaluating a given circuit, the prover may generate the proof for the circuit using the SNARK and then either send this proof directly, or prove knowledge of the proof by evaluating and proving the verifier circuit. Note that the two SNARKs applied here do not necessarily have to be the same, so the prototypical application of recursion is to combine the best

properties of two different SNARKs.

In our setting, we are more interested in using recursion in order to prove a loop, i.e. IVC. Let F be the function describing the step function of the loop, i.e. we want to proof $y = F^n(x)$, where $F^n(x)$ corresponds to applying F successively n times to x. We may augment F to obtain a function F' that takes as input a value y_i and a proof π_i and outputs y_{i+1} and a proof π_{i+1} attesting to the correctness of the statement:

- $y_{i+1} = F(y_i)$ and
- the verifier accepts π_i as a proof of $y_i = F(x)$.

The prover may now successively evaluate $(y_i, \pi_i) = F'(y_{i-1}, \pi_{i-1})$ and obtain the output y_n along with a succinct proof π_n.

While this approach allows to transform the problem of proving the computation of F^n to n computations of F', there is some overhead due to the fact that we prove F' instead of F for each step of the computation, meaning the circuit size for each step increased. This increase essentially corresponds to the size of the verifier circuit, which in turn depends on the used SNARK. In our setting, we use a FRI based PCS, which has a relatively large verifier circuit, but on the other hand avoids the need of cycles of elleptic curves plaguing other approaches.

4 Folding

The folding paradigm was more recently introduced in the work of [BGH19] and has spurred a flurry of new works [KST22, KS22, BC23, BCL+21, BCMS20, Moh23]. Recall that SNARKs are constructed by arithmetizing the computation to turn it into some intermediate format, e.g. R1CS or PLONK, which is then combined with a suitable polynomial commitment scheme. The intermediate format is some type of mathematical object (hence the term *arithmetization*), e.g. a contraint system or a polynomial, with specific properties that ensure that the computation is correct and allow to prove these properties. In the following we will denote a computation in the intermediate format by an *instance*. The general idea of a folding scheme is to take two instances of similar size and merge them into one instance of the same size. By using randomness provided by the verifier, the key observation is that the merged instance is with high probability not going to be a valid instance if one of the two initial instances was not valid. In turn

that means that it is sufficient to prove the merged instance in order to ensure that the two initial instances were correct.

Note that such an approach allows to prove loops by constructing an instance for each loop iteration and successively folding them together into an instance of size comparable to one loop iteration. Finally, the idea is to apply the proof only to the merged instance.

In contrast to IVC based on recursion, folding schemes do not require to compute an entire proof for each loop iteration — it is sufficient to compute the arithmetization of each step. Accordingly, this has the potential of being significantly more efficient than the approach based on recursion. On the other hand, folding does have its drawbacks, mostly hidden by the drastically simplified description above. In particular, it introduces cross terms (or error terms) that need to be handled. This requires to relax the conditions on the instances, introduces homomorphic commitments that need to be included in the circuit, and generally complicates the method significantly.

5 Discussion

In principle, both approaches seem to bear the potential of reducing the cost of proving the blind rotation. IVC based on recursion might be less efficient, but the concept has been known for a longer time and thus is more mature. In particular, there are efficient implementations of recursive SNARKs, for example plonky2 [Pol22], which is implemented in Rust and shares many properties of our prototype (a SNARK based on PLONK and FRI), but is optimized for recursion. Although documentation is scarce, preliminary experimentation suggests that it should be possible to realize an IVC-based blind rotation using plonky2, which has the potential to obtain results in a relatively short timeframe.

On the other hand, IVC based on folding seems to be more promising with regards to performance, but, as far as we know, there is at this point only one implementation of a folding scheme, namely Nova [Mic23, KST22]. Nova diverges significantly from our current approach by using the R1CS arithmetization. So if we wanted to continue our current approach based on PLONK, we would presumably have to implement the folding scheme ourselves, which will likely be more time consuming than using an efficient implementation like plonky2.

In summary, the approach based on recursion promises con-

crete results in the short term, while the one based on folding is likely to produce better results in the long term. Accordingly, we plan on exploring folding further in the near future with the goal of including an implementation in our prototype.

References

[BC23] Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special sound protocols. Cryptology ePrint Archive, Paper 2023/620, 2023. https://eprint.iacr.org/2023/620. URL: https://eprint.iacr.org/2023/620.

[BCL+21] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 681–710, Virtual Event, August 2021. Springer, Heidelberg. doi:10.1007/978-3-030-84242-0_24.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 1–18. Springer, Heidelberg, November 2020. doi:10.1007/978-3-030-64378-2_1.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. https://eprint.iacr.org/2019/1021.

[KS22] Abhiram Kothapalli and Srinath Setty. SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Report 2022/1758, 2022. https://eprint.iacr.org/2022/1758.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*,

pages 359–388. Springer, Heidelberg, August 2022. doi:10.1007/978-3-031-15985-5_13.

[Mic23] Microsoft. Nova: Recursive snarks without trusted setup. https://github.com/microsoft/Nova, 2023. Accessed: 2023-09-26.

[Moh23] Nicolas Mohnblatt. Sangria: a folding scheme for PLONK. https://geometry.xyz/notebook/sangria-a-folding-scheme-for-plonk, 2023. Accessed: 2023-03-30.

[Pol22] Polygon. Plonky2. https://github.com/mir-protocol/plonky2, 2022. Accessed: 2023-09-26.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Heidelberg, March 2008. doi:10.1007/978-3-540-78524-8_1.

[Wal23a] Michael Walter. Towards verifiable FHE: Proving correct execution of the external product using Plonk. Technical report, Zama, April 2023. Available at https://github.com/zama-ai/publications/raw/main/techreps/2023/verif_comp.pdf.

[Wal23b] Michael Walter. Towards verifiable TFHE (part 2): Proving correct execution of the blind rotation using plonk. Technical report, Zama, June 2023. Available at https://github.com/zama-ai/publications/raw/main/techreps/2023/vfhe_eoq_23-q2.pdf.