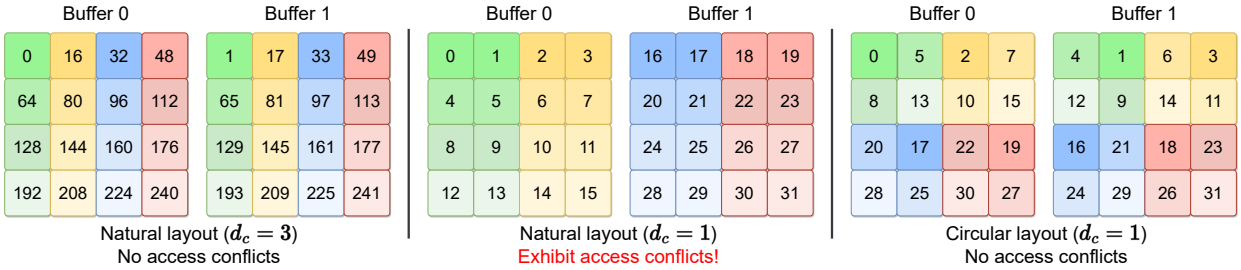

Fig. 5. On-chip buffer layout schemes in SAM, with $t = 2$ and $n = 4$. The numbers in the blocks denote the global element indices. Each two elements with the same color are fetched from the off-chip DDR together in the same cycle. Left: The natural layout works well with no buffer conflicts when $d_c > 1$. Middle: The natural layout incurs buffer conflicts when $d_c = 1$. Two elements fetched in the same cycle need to be stored to the same buffer. Right: The circular layout resolves all buffer conflicts for $d_c = 1$, by circulating and distributing elements across all buffers.

in different buffers, so that we can read out one element from each of the t rows/columns in parallel for the t NTT pipelines.

Our circular layout scheme satisfies the above requirements in the following way. For an element at the position of row r, column c, and plane p (all starting from 0) in the natural layout, e.g., 24 is at position $(r = 2, c = 0, p = 1)$, we put it to the position (r', c', p') in the circular layout, where

$$k = (p + r) \bmod t \tag{1}$$
$$r' = (p \cdot n + r)/t \tag{2}$$
$$c' = c \tag{3}$$
$$p' = (c + k) \bmod t \tag{4}$$

For example, 24 with $(r = 2, c = 0, p = 1)$ has $k = 1$, and is put to $(r' = 3, c' = 0, p' = 1)$. The idea is to distribute the t elements (with consecutive column indices c) to the t buffers (with consecutive p') as in Equation (4), following a certain circular shift distance k. The shift distance k is determined by the original row index r in Equation (1), so elements in consecutive rows are also distributed to different buffers.

We use a circular shift network-on-chip (NoC) across the multiple compute lanes, to realize the mapping from c to p' for the t elements in each DDR access. It calculates the required k value for shifting the elements. Since k can only have t possible values (usually 4 to 16 in our design), we use pipelined multiplexers to implement the shift network in a straightforward way. We implement a simple buffer address generator at each buffer to calculate r' and c', which determines the location in the buffer to write the data to.

E. Additional Hardware Optimizations

NTT pipelines. SAM adopts its NTT pipeline design from PipeZK [16]. As a size-n NTT unit, the pipeline has $\log n$ stages. There are input/output FIFO buffers before/after each stage, whose FIFO depth is equal to the desired access stride, i.e., $\frac{n}{2^i}$ for stage i. Each stage of the pipeline receives one input element per cycle, and outputs one result per cycle. The FIFO buffers organize the data to realize the expected strided accesses. This design has several advantages that match our multi-dimensional decomposition. First, the pipeline reads and writes one element per cycle, which reduces the data bandwidth requirement and is very friendly to non-consecutive data accesses. Second, it is easy to support the incomplete dimension (if any) $m < n$ for general power-of-2 values of N, by skipping the first few stages in the pipeline. Third, this design has good scalability on FPGA, especially for the way to support strided data selection, where FIFOs are relatively easy to instantiate than a large number of multiplexers [10].

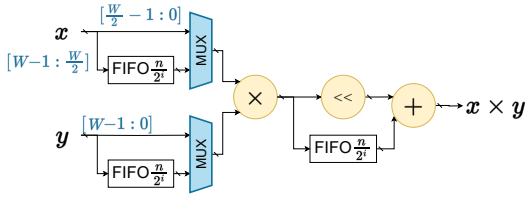

Fig. 6. Using a $\frac{W}{2}$-bit $\times$ W-bit modular multiplier to perform W-bit $\times$ W-bit multiplications with half of the throughput.

However, this NTT pipeline has a critical issue, where the fully pipelined modular multiplier at each stage is only 50% utilized. This is because only one element enters the stage in each cycle, but it requires two elements to compute. The stage i works for $\frac{n}{2^i}$ cycles and then sits idle for another $\frac{n}{2^i}$ cycles.

To tackle this issue, we propose an improved design that uses a half-throughput modular multiplier at each pipeline stage. We reduce the modular multiplier width from W-bit $\times$ W-bit to $\frac{W}{2}$-bit $\times$ W-bit, and add additional operand and result FIFOs before and after it, as shown in Fig. 6. Specifically, for input operands x and y, we first compute $x_L \times y$ (we omit modulo in this paragraph) and store to the result FIFO, while buffering both x_H and y in the operand FIFOs. After all input data of this phase are fed in and during the idle time, x_H and y are popped from the operand FIFOs for computing, and the result is shifted and added to the corresponding product $x_L \times y$ from the result FIFO, to obtain the final result $x \times y$. In many cases y is always a constant, and the second operand FIFO can be removed. This optimization reduces the DSP consumption by roughly 1/3 without any negative performance impact.

Data prefetching. So far, SAM uses t compute lanes to process t 2-D data planes in parallel. In general, these 2-D planes have non-consecutive elements along the rows and columns, while Section III-B ensures that the elements along the plane dimension are always consecutive, and thus fetching multiple

planes together increases the DDR burst access granularity and correspondingly the bandwidth utilization. However, t is limited by the available FPGA logic resources. On a typical FPGA, DSP units are more precious than RAM blocks. If we only fetch and buffer t planes, we would underutilize the RAM space, while the DDR access burst length is still insufficient.

We hence introduce a buffer capacity extension factor b to decouple data processing and data buffering. SAM actually fetches b groups of t 2-D planes together, with the buffer in each lane also enlarged by $b\times$. These b groups of planes are processed sequentially. All the $b \times t$ planes within one fetch have consecutive base addresses, making each DDR access of btW bits. Thus a sufficiently large b can fully utilize the on-chip RAM space to achieve high DDR bandwidth utilization.

Having $b\times$ more available work on-chip also helps hide the long NTT pipeline latency. When switching from column NTTs to row NTTs, we must wait until the output from the last column NTT to be written back to the buffer, before we can read out the first row to start the first row NTT. In our multi-stage NTT pipeline [16], such dependencies result in long pipeline stalls. With b groups of planes, we adjust the execution order, to first perform the row NTTs on all b groups, and then start the column NTTs.

IV. Design Space Exploration

SAM contains three main design parameters, the size of each NTT pipeline n, the number of parallel lanes t, and the buffer capacity extension factor b. They exhibit various tradeoffs. A larger n consumes more on-chip resources, but also supports larger NTT kernels natively and reduces the decomposition overheads. Increasing t improves parallelization and thus performance, but is limited by both the on-chip DSP resources and the off-chip DDR bandwidth. Finally, b should be large enough to better utilize the memory bandwidth, subject to the FPGA RAM space limit.

To determine these parameters, we consider latency matching between computations and data accesses for double buffering. SAM uses one DDR channel for data reads and another for writes. We empirically find on our platform that the DDR burst length per channel must be no less than $L = 1\,\text{kB}$ to achieve the maximum sustainable bandwidth of $B = 11\,\text{GB/s}$. When $W = 256\,\text{bits}$, we can achieve a frequency of $f = 100\,\text{MHz}$. For the compute latency, there are $\frac{N\log N}{2}$ butterfly operations, and our NTT pipeline needs 2 cycles for each. So

$$\text{Latency}_{\text{comp}} = \frac{N}{tf\log m} + \frac{N(\log N - 1)}{tf\log n}$$

where the first term specially considers the incomplete dimension m. For the memory latency, we multiply the data size NW by the number of rounds, and divide by the bandwidth.

$$\text{Latency}_{\text{mem}} = \frac{NW}{B}\left\lceil\frac{\log N}{2\log n}\right\rceil$$

Letting the two latencies equal, and simplifying by omitting the ceiling function and assuming $m = n$, we get $t = \frac{2B}{Wf} \approx 6.9$. Because t must be a power of 2 as required by the buffer layout in Section III-D, it may be 4 or 8. We find that for large N values, there are many large strided accesses that decrease the effective bandwidth B, which in turn lowers the optimal value of t. We thus choose $t = 4$. Accordingly, $b = \frac{L}{Wt} = 8$.

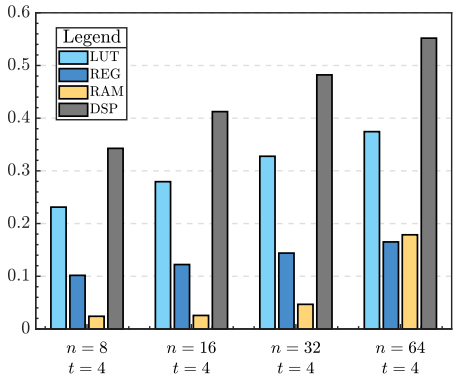

Fig. 7. Resource consumption of various n values for $t = 4$ at 256 bits.

Finally, to decide n, we see both the compute and memory access latencies above decrease with $\log n$, making us use an n as large as possible. However, we are limited by the on-chip FPGA resources. As shown in Fig. 7, the DSP count eventually restricts SAM to use $n = 64$.

Similarly, we calculate another configuration of SAM for $W = 64\,\text{bits}$ with a frequency $f = 165\,\text{MHz}$. Repeating the above procedure, we have $t = 16$, $b = 8$, $n = 64$. Narrower multipliers use fewer DSPs and allow for more parallel lanes.

V. Evaluation

In this section, we evaluate SAM against the CPU baseline and previous NTT accelerators on FPGAs.

A. Experimental Methodology

We implement the RTL design of SAM in Verilog, and synthesize it using Xilinx Vitis 2021.1, targeting the Alveo U250 card with 12284 DSP slices, 2547 instances of 36 kbit BRAM, and 1280 instances of 288 kbit URAM. We set the target frequency to 100 MHz (256 bits) or 165 MHz (64 bits) for the core logic, and 300 MHz for the I/O and memory interface. We have extensively verified the correctness using random tests and comparing with the CPU baseline [30]. The host server has an Intel Xeon Gold 5120 CPU with 14 physical cores at 2.2 GHz, and 252 GB of DDR4 memory.

We compare SAM with several state-of-the-art FPGA-based NTT accelerators [8], [12], [14]–[16], as well as a CPU baseline that uses libsnark [30] optimized for large-scale 256-bit NTTs in ZKP. We evaluate NTT sizes in the range of 2^{16} to 2^{28}. This range is commonly used for HE and ZKP applications. It not only covers the problem sizes in previous work [8]–[13], [17]–[19], but also extends to larger scales rarely supported before. For each bitwidth W, the same hardware configuration can work for different N values. All reported results of SAM are from real on-board executions. The input data of the desired size N are initially stored in

TABLE I
OVERALL COMPARISON RESULTS.

Design	Platform	W (bits)	Freq (MHz)	Data	LUT / REG / RAM (kb) / DSP	N	Latency (ms)
SAM (**Ours**)	XCU250	256	100 & 300	Off-chip	593073 / 533675 / 81684 / 6776	2^{16} 2^{20} 2^{24} 2^{28}	1.24 12.61 183.56 4023.49
CPU [30]	Intel Xeon	256	2200	Off-chip	-	2^{16} 2^{20} 2^{24}	511.47 1244.68 19970.75
PROTEUS [14]	XCU250	256	125	***On-chip***	356000 / - / 55620 / 2640	2^{16}	1.05
PipeZK-Scaled [16]	-	256	100 (scaled)	Off-chip	-	2^{20}	33.00
SAM (**Ours**)	XCU250	64	165 & 300	Off-chip	267132 / 328486 / 76536 / 2736	2^{16} 2^{17} 2^{18} 2^{20} 2^{24} 2^{28}	0.38 0.56 0.99 2.84 34.12 750.14
HEPCloud [8]	XC6VLX240T	30	100 & 200	***On-chip***	72163 / 63086 / 3420 / 250	2^{16}	0.48
FCCM'20 [12]	XCVU190	62	200	***On-chip***	365000 / - / 81304 / 1332	2^{17}	0.98
PROTEUS [14]	XCVX485T	64	135	***On-chip***	31300 / - / 9180 / 300	2^{16}	0.44
CNTT-4 [15]	VU55P	64	300	***HBM***	53026 / 46026 / 13824 / 74	2^{20} 2^{24}	2.12 42.57
CNTT-6 [15]	VU55P	64	161	***HBM***	563677 / 319104 / 82944 / 691	2^{18} 2^{24}	0.10 8.08
CNTT-6-Scaled [15]	-	64	161	Off-chip	-	2^{18} 2^{24}	0.88 71.44

the FPGA board memory. The latency measurements do not include data transfers between the host and the FPGA board.

B. Results

Table I summarizes the overall comparison between SAM and the baselines. First, for $W = 256$ bits, SAM uses $t = 4$, $b = 8$, $n = 64$. It achieves $376\times, 99\times, 109\times$ speedups at $N = 2^{16}, 2^{20}, 2^{24}$, respectively, compared to the CPU baseline. Among previous FPGA-based NTT accelerators, only PROTEUS [14] reported performance for 256 bits, but it assumed all data fit on-chip and only supported up to $N = 2^{16}$. Nevertheless, SAM almost matches this on-chip performance with 18% slowdown despite that its data are from off-chip, demonstrating that our design parameter selection in Section IV effectively hides most memory access latencies. We also compare with a scaled version of PipeZK [16]. PipeZK was an ASIC design and had a much higher frequency. We scale down the frequency to be the same as SAM, and optimistically assume linear performance scaling. SAM outperforms PipeZK-Scaled by $2.6\times$ at $N = 2^{20}$, the largest size PipeZK could support.

For the 64-bit version of SAM, we use $t = 16$, $b = 8$, $n = 64$. HEPCloud [8], FCCM'20 [12], and PROTEUS [14] were all on-chip designs and only supported small NTT sizes up to 2^{16} or 2^{17}. Nevertheless, SAM is able to perform slightly faster than all of them, even though it needs to access off-chip data. CNTT-4 and CNTT-6 were two different configurations in CycloneNTT [15], which used an HBM-capable FPGA board. CNTT-4 utilized 6 HBM channels, while CNTT-6 utilized 24 HBM channels; but they only supported $\log N$ as multiplies of 4 and 6, with limited flexibility. It is not surprising the much higher memory bandwidth helps CycloneNTT perform better. However, if we scale down their bandwidth to match SAM, e.g., by $0.11\times$ from CNTT-6, SAM would achieve a $2.1\times$ speedup at $N = 2^{24}$. Furthermore, CycloneNTT was specially optimized for the Goldilocks field, which reduced the resource consumption and boost its frequency. SAM could also be specialized to a certain field, but we opt for generality.

VI. CONCLUSION

In this paper, an FPGA-accelerated NTT architecture, SAM, is designed based on the recursive multi-dimensional decomposition of the NTT algorithm. SAM can flexibly support a wide range of NTT kernel sizes commonly seen in HE and ZKP algorithms, up to 2^{28}. It has specific optimizations for its on-chip data layout and off-chip data accesses under the execution flow of multi-dimensional decomposition. SAM achieves significant speedups over previous NTT accelerators.

ACKNOWLEDGMENT

The authors thank the anonymous reviewers for their valuable suggestions, and the Tsinghua IDEAL group members for constructive discussion. This work was supported by the National Natural Science Foundation of China (62072262).

REFERENCES

[1] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC)*, 2009, p. 169–178.

[2] J. Groth, "On the Size of Pairing-Based Non-interactive Arguments," in *Proceedings of the 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2016, pp. 305–326.

[3] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) Fully Homomorphic Encryption without Bootstrapping," *ACM Transactions on Computation Theory*, vol. 6, no. 3, Jul 2014.

[4] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption," Cryptology ePrint Archive, 2012. [Online]. Available: https://eprint.iacr.org/2012/144

[5] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *Proceedings of the 23rd International Conference on the Theory and Applications of Cryptology and Information Security*, 2017, pp. 409–437.

[6] J. Groth and M. Maller, "Snarky Signatures: Minimal Signatures of Knowledge from Simulation-Extractable SNARKs," in *Annual International Cryptology Conference*, 2017, pp. 581–612.

[7] H. Wu, W. Zheng, A. Chiesa, R. A. Popa, and I. Stoica, "DIZK: A Distributed Zero Knowledge Proof System," Cryptology ePrint Archive, 2018. [Online]. Available: https://eprint.iacr.org/2018/691

[8] S. Sinha Roy, K. Järvinen, J. Vliegen, F. Vercauteren, and I. Verbauwhede, "HEPCloud: An FPGA-Based Multicore Processor for FV Somewhat Homomorphic Function Evaluation," *IEEE Transactions on Computers*, vol. 67, no. 11, pp. 1637–1650, 2018.

[9] S. Sinha Roy, F. Turan, K. Jarvinen, F. Vercauteren, and I. Verbauwhede, "FPGA-Based High-Performance Parallel Architecture for Homomorphic Computing on Encrypted Data," in *Proceedings of the 25th International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 387–398.

[10] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "HEAX: An Architecture for Computing on Encrypted Data," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020, p. 1295–1309.

[11] N. Zhang, Q. Qin, H. Yuan, C. Zhou, S. Yin, S. Wei, and L. Liu, "NTTU: An Area-Efficient Low-Power NTT-Uncoupled Architecture for NTT-Based Multiplication," *IEEE Transactions on Computers*, vol. 69, no. 4, pp. 520–533, 2020.

[12] S. Kim, K. Lee, W. Cho, Y. Nam, J. H. Cheon, and R. A. Rutenbar, "Hardware Architecture of a Number Theoretic Transform for a Bootstrappable RNS-based Homomorphic Encryption Scheme," in *Proceedings of the 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020, pp. 56–64.

[13] A. C. Mert, E. Karabulut, E. Öztürk, E. Savaş, M. Becchi, and A. Aysu, "A Flexible and Scalable NTT Hardware: Applications from Homomorphically Encrypted Deep Learning to Post-Quantum Cryptography," in *Proceedings of the 23rd Conference on Design, Automation and Test in Europe (DATE)*, 2020, pp. 346–351.

[14] F. Hirner, A. C. Mert, and S. S. Roy, "PROTEUS: A Tool to Generate Pipelined Number Theoretic Transform Architectures for FHE and ZKP Applications," Cryptology ePrint Archive, 2023. [Online]. Available: https://eprint.iacr.org/2023/267

[15] K. Aasaraai, E. Cesena, R. Maganti, N. Stalder, J. Varela, and K. Bowers, "CycloneNTT: An NTT/FFT Architecture Using Quasi-Streaming of Large Datasets on DDR- and HBM-based FPGA Platforms," Cryptology ePrint Archive, 2022. [Online]. Available: https://eprint.iacr.org/2022/1657

[16] Y. Zhang, S. Wang, X. Zhang, J. Dong, X. Mao, F. Long, C. Wang, D. Zhou, M. Gao, and G. Sun, "PipeZK: Accelerating Zero-Knowledge Proof with a Pipelined Architecture," in *Proceedings of the 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 416–428.

[17] N. Samardzic, A. Feldmann, A. Krastev, S. Devadas, R. Dreslinski, C. Peikert, and D. Sanchez, "F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption," in *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021, p. 238–252.

[18] N. Samardzic, A. Feldmann, A. Krastev, N. Manohar, N. Genise, S. Devadas, K. Eldefrawy, C. Peikert, and D. Sanchez, "CraterLake: A Hardware Accelerator for Efficient Unbounded Computation on Encrypted Data," in *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*, 2022, p. 173–187.

[19] S. Kim, J. Kim, M. J. Kim, W. Jung, J. Kim, M. Rhu, and J. H. Ahn, "BTS: An Accelerator for Bootstrappable Fully Homomorphic Encryption," in *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*, 2022, p. 711–725.

[20] N. K. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli, "High Performance Discrete Fourier Transforms on Graphics Processors," in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC)*, 2008, pp. 1–12.

[21] S. Kim, W. Jung, J. Park, and J. H. Ahn, "Accelerating Number Theoretic Transformations for Bootstrappable Homomorphic Encryption on GPUs," in *Proceedings of the 2020 IEEE International Symposium on Workload Characterization (IISWC)*, 2020, pp. 264–275.

[22] O. Özerk, C. Elgezen, A. C. Mert, E. Öztürk, and E. Savaş, "Efficient Number Theoretic Transform Implementation on GPU for Homomorphic Encryption," *Journal of Supercomputing*, vol. 78, no. 2, p. 2840–2872, Feb 2022.

[23] J.-C. Bajard, J. Eynard, M. A. Hasan, and V. Zucca, "A Full RNS Variant of FV Like Somewhat Homomorphic Encryption Schemes," in *Selected Areas in Cryptography (SAC)*, 2017, pp. 423–442.

[24] S. Halevi, Y. Polyakov, and V. Shoup, "An Improved RNS Variant of the BFV Homomorphic Encryption Scheme," in *Topics in Cryptology (CT-RSA)*, 2019, pp. 83–105.

[25] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A Full RNS Variant of Approximate Homomorphic Encryption," in *Selected Areas in Cryptography (SAC)*, 2019, pp. 347–368.

[26] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized Anonymous Payments from Bitcoin," in *Proceedings of the 35th IEEE Symposium on Security and Privacy (S&P)*, 2014, pp. 459–474.

[27] "Filecoin: A Decentralised Market for Storage," https://filecoin.io/filecoin.pdf, 2022.

[28] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. H. Ahn, "ARK: Fully Homomorphic Encryption Accelerator with Runtime Data Generation and Inter-Operation Key Reuse," in *Proceedings of the 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 1237–1254.

[29] R. Agrawal, L. de Castro, G. Yang, C. Juvekar, R. Yazicigil, A. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "FAB: An FPGA-based Accelerator for Bootstrappable Fully Homomorphic Encryption," in *Proceedings of the 29th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 882–895.

[30] "libsnark: a C++ library for zkSNARK proofs," https://github.com/scipr-lab/libsnark, 2022.

[31] "bellperson: GPU parallel acceleration for zk-snark," https://github.com/filecoin-project/bellperson, 2022.

[32] "bellman: a crate for building zk-snark circuits," https://github.com/zkcrypto/bellman, 2022.

[33] "jsnark: A java library for building snarks," https://github.com/akosba/jsnark, 2022.

[34] PolygonZero, "Plonky2: Fast Recursive Arguments with PLONK and FRI." https://github.com/mir-protocol/plonky2/blob/main/plonky2/plonky2.pdf, 2022.