Efficient Generic Arithmetic for KKW

Practical Linear MPC-in-the-Head NIZK on Commodity Hardware Without Trusted Setup

David Heath(✉), Vladimir Kolesnikov, and Jiahui Lu

Georgia Institute of Technology, Atlanta, GA, USA
{heath.davidanthony,kolesnikov,jlu355}@gatech.edu

Abstract. Katz et al., CCS 2018 (KKW) is a popular and efficient MPC-in-the-head non-interactive ZKP (NIZK) scheme, which is the technical core of the post-quantum signature scheme Picnic, currently considered for standardization by NIST. The KKW approach simultaneously is concretely efficient, even on commodity hardware, and does not rely on trusted setup. Importantly, the approach scales linearly in the circuit size with low constants with respect to proof generation time, proof verification time, proof size, and RAM consumption. However, KKW works with Boolean circuits only and hence incurs significant cost for circuits that include arithmetic operations.

In this work, we extend KKW with a suite of efficient arithmetic operations over arbitrary rings and Boolean conversions. Rings $\mathbb{Z}_{2^k}$ are important for NIZK as they naturally match the basic operations of modern programs and CPUs. In particular, we:

– present a suitable ring representation consistent with KKW,

– construct efficient conversion operators that translate between arithmetic and Boolean representations, and

– demonstrate how to efficiently operate over the arithmetic representation, including a vector dot product of length-n vectors with cost equal to that of a *single* multiplication.

These improvements substantially improve KKW for circuits with arithmetic. As one example, we can multiply 100×100 square matrices of 32 bit number using $3200\times$ smaller proof size than standard KKW ($100\times$ improvement from our dot product construction and $32\times$ from moving to an arithmetic representation).

We discuss in detail proof size and resource consumption and argue the practicality of running large proofs on commodity hardware.

Keywords: MPC-in-the-Head · Zero knowledge

1 Introduction

Zero-knowledge proofs of knowledge (ZKPoKs) enable a prover $\mathcal{P}$, given a public circuit C, to show that she holds a witness w, such that $C(w) = 1$. Recent research focuses on efficient ZK proofs of *arbitrary* statements. A special case of ZK is *non-interactive* ZK (NIZK). NIZK proofs can be transferred and verified without interacting with $\mathcal{P}$.

[KKW18] specified a powerful NIZK proof system over Boolean circuits that features linear scaling in proof size, in verifier time, and, critically, in proof generation time. In this work, we extend this system with efficient arithmetic and conversions between Boolean and arithmetic representations. Our contribution thus reduces both proof size and computation.

Motivation and Setting for Our Work. ZKPs, and especially NIZKs, have enjoyed immense research interest in recent years. The majority of such works prioritize small proof size and fast verification, important metrics in blockchain-related applications. However, optimizing these metrics comes at significant prover cost. In experiments reported in many works, provers are run on powerful servers with hundreds of GB of RAM. Asymptotically, proof times are typically super-linear in the size of the proof circuit, with costs $O(n \log n)$.

At the same time, moderate resource requirements, such as low memory utilization, are essential to a class of applications, such as those running a ZK prover on a mobile device. Modern flagship phones have 4 to 6 GB RAM, a portion of which can be made available to the NIZK application.

This leaves room for a *balanced approach* that prioritizes total proof time, and that takes into account the ability to run on commodity hardware, and the costs of proof generation, network transmission, and verification.

We argue that [KKW18] is a great fit for applications where only commodity hardware is available and where concretely efficient performance is demanded: [KKW18]'s linear scaling in communication, prover computation, and verifier computation mean that the approach remains tractable even for large proof statements. [KKW18]'s RAM consumption is low even for large proof functions due to the gate-by-gate proof generation, and [KKW18] uses only light-weight computational primitives. The technique also requires no trusted setup. Finally, [KKW18] is actively supported by the community, since it is under consideration standardization by NIST as part of the Picnic post-quantum signature scheme.

However, [KKW18] supports only Boolean circuits. When [KKW18] is used in contexts that require complex arithmetic, the circuit size grows significantly, increasing both proof computation and communication. In this work, we improve [KKW18] by extending it with efficient arithmetic operations and with conversions between Boolean and arithmetic representations.

1.1 A Use Case for Balanced ZKP

To illustrate and make more precise our motivation, we explicate one natural use case where [KKW18] would be a top ZKP system among prior work.

Consider a set of mobile phones on a local Wi-Fi or bluetooth broadcast network, e.g., in the context of a group event, private contact tracing, etc. Suppose one phone wishes to prove a statement to everyone on the broadcast channel. Note that even though interaction is available here, the interactive designated-verifier systems, such as [JKO13, HK20] must repeat the proof for each verifier. Thus interactive techniques are not well suited for convincing the entire network at once, and a NIZK proof, which can be broadcasted, is a better solution.

In such a setting, the broadcast network resource is surprisingly substantial: while somewhat slower than 1gbps LAN, Wi-Fi supports speeds up to many hundreds of Mbps. Bluetooth 5 supports bandwidth of up to 2Mbps on distances of up to 800 ft (240 m).

We wish to complete the proof, including its generation, transmission (which may overlap with the other two phases), and verification, as quickly as possible. One natural way to view this optimization space is to ask: "*given the available bandwidth, say 10Mbps, is the bottleneck proof generation, transmission, or verification?*" The answer to this question (cf. Sect. 2 discussion of ZK systems' costs) is: "Proof generation/verification." Thus [KKW18], a proof system with concretely efficient linear scaling in the proof size, is a top choice.

1.2 Our Contribution and Outline of the Work

We extend the [KKW18] proof system with efficient ring arithmetic. Ring (e.g., vs field) operations are a particularly useful primitive for ZK, since they naturally match basic steps of existing programs written in standard languages, such as `C`. A ring-based ZK system can thus be more naturally used in proving program properties (e.g. presence of bugs [HK20]) in ZK.

While [dSGMOS19, BN19] (cf. Related Work Sect. 2) considered adding arithmetic to KKW for highly tailored applications, we provide a generic construction, and additionally offer the following efficient arithmetic operations:

Consider a finite ring whose elements are l bits long.

– We add an efficient operation that computes the dot-product of two arbitrary size vectors of ring elements for $2l$ proof bits (cf Sect. 5.1).
– We add efficient conversions between Boolean and arithmetic representations. Specifically, we add conversion operations between Boolean and rings $\mathbb{Z}_k$ for arbitrary k. Let $l = \log k$. Converting l Boolean values to an arithmetic value (or vice versa) costs l^2 bits in the proof (cf Sect. 5.2).

In Sect. 6, we formalize our approach as a p party semi-honest protocol in the preprocessing model, prove its security, and explain how it plugs into the honest-verifier ZK protocol of [KKW18]. Thus, via the Fiat-Shamir transform [FS87] our approach directly extends the NIZK [KKW18] proof system.

We provide a detailed account of the performance of our system, including individual gate costs and comparisons with standard [KKW18] (Sect. 7). We demonstrate that for arithmetic operations, our approach substantially improves [KKW18]. Of particular note is our improvement for linear arithmetic: as an example, our approach can multiply 100×100 square matrices of 32 bit number using $3200\times$ smaller proof size than standard [KKW18].

1.3 Intuition: MPC-in-the-Head, [KKW18] and Our Work

[KKW18] is a NIZK in the MPC-in-the-Head paradigm. In the MPC-in-the-head paradigm, the prover $\mathcal{P}$ simulates *in her head* a secure multi-party computation (MPC) protocol between several 'virtual players'. These players are given shares of $\mathcal{P}$'s witness as input and run the proof circuit C under MPC. $\mathcal{P}$ commits to the views of these players, and then the verifier $\mathcal{V}$ selects a subset of views to open. By checking that these views are consistent with the honest execution of the MPC protocol resulting in the output 1, $\mathcal{V}$ gains confidence that $\mathcal{P}$ did not cheat and indeed has a witness. Because $\mathcal{V}$ does not see the views of all players, the MPC protocol's security properties prevent him from learning $\mathcal{P}$'s witness. Therefore, such a protocol achieves Zero Knowledge. The players amplify soundness by repeating the protocol. Such systems can be transformed into NIZK proof systems using the classic Fiat-Shamir transform [FS87].

[KKW18] implements MPC-in-the-head with a protocol heavily based on *preprocessing*. Preprocessing fits elegantly with MPC-in-the-head because it can be easily prepared by $\mathcal{P}$ and checked by $\mathcal{V}$. As do [IKOS07, GMO16, CDG+17], [KKW18] allows efficient broadcast-based MPC, which allows $\mathcal{P}$ to simulate larger numbers of MPC players in her head. Because the protocol happens only in $\mathcal{P}$'s head, these broadcasts are efficient. By simulating more players, $\mathcal{P}$ reduces the number of repetitions needed to amplify soundness.

Our work notices inherent flexibility in this broadcast-based MPC protocol. We point out that broadcasts of Boolean values are easily generalized to broadcasts of elements of arbitrary finite *rings*. We show how this extension allows us to directly encode algebraic operations like addition and multiplication, significantly reducing cost. We further show how Boolean and k-bit integer operations can be mixed in the same circuit by including *conversion* operations.

2 Related Work

Zero Knowledge. ZKP [GMR85, GMW91] is a fundamental cryptographic primitive. ZK proofs of knowledge (ZKPoKs) [GMR85, BG93, DP92] allow a prover to convince a verifier, who holds a circuit C, that the prover knows an input, or *witness*, w for which $C(w) = 1$. Originally, practical ZK research focused on specific algebraic relations. More recently, ZK research has focused on practical proofs of arbitrary circuits. Our work is in this arbitrary circuit setting.

MPC-in-the-Head. Ishai et al. [IKOS07], introduced the powerful 'MPC-in-the-head' paradigm, outlined in Sect. 1.3. ZKBoo [GMO16] was the first implementation of MPC-in-the-head. Chase et al. [CDG+17] deprecated ZKBoo by building a more efficient system, ZKB++. They demonstrated that ZKB++ can implement an efficient signature scheme using only symmetric-key primitives.

Katz et al. [KKW18], the basis of our work, further improved this direction by using MPC with preprocessing. *Picnic* [ZCD+17], a signature scheme based on ZKB++, was submitted to the NIST post-quantum standardization effort. The Picnic submission was since updated and is now based on [KKW18].

Ligero [AHIV17] is another MPC-in-the-head protocol that diverges from our work's lineage. Ligero offers sublinear proof size ($O(\sqrt{|C|})$), but incurs superlinear prover computation ($O(|C| \log |C|)$). It is estimated that Ligero constructs smaller proofs than [KKW18] for circuits with more than approximately 100K gates. Thus, a choice between [KKW18] and Ligero should be based on the desired application and on performance requirements.

SNARKs. Succinct non-interactive arguments of knowledge (SNARK) [GGPR13, PHGR13, BCG+13, CFH+15, Gro16] build proofs that are particularly efficient, both in communication and verification time. They construct proofs that are shorter than the input itself. Prior work demonstrated the feasibility of sublinear ZK proofs [Kil92, Mic94], but were concretely inefficient. Early SNARKs required a semi-trusted party. This disadvantage led to the development of STARKs (succinct transparent arguments of knowledge) [BBHR18]. STARKs do not require trusted setup and rely on more efficient primitives. STARKs are succinct ZKP, and thus are SNARKs. In this work, we do not separate them; rather we see them as a body of work focused on sublinear proofs.

Recent SNARKs include Libra [XZZ+19] and Virgo [ZXZS19]. SNARKs [MBKM19, CHM+20] rely on trusted setup, which we wish to avoid. SPARKs [EFKP20] parallelize expensive prover time, but total CPU consumption (our metric) is superlinear. Supersonic's prover [BFS20] is quazilinear with high constants. Fractal [COS20] runs its concretely expensive prover on a high-end Intel Xeon 6136 CPU at 3.0 GHz with 252 GB of RAM (no more than 32 GB of RAM were used in any experiment).

Interactive ZK. In this work, we focus on concretely efficient non-interactive ZK. Another direction forgoes non-interactivity in exchange for very fast proofs. Interactivity allows private-coin ZK protocols, such as those based on [JKO13] and garbled circuits (GC). In [JKO13], $\mathcal{V}$ garbles the evaluated circuit, then $\mathcal{P}$ evaluates and thus obtains the random encoding of the output. The GC authenticity property guarantees that $\mathcal{P}$ is unable to obtain a requisite output label without evaluating with a valid witness w. Recently, [HK20] showed that *conditional branches* in the proof circuit can be evaluated for free.

We achieve performance similar to the above works (linear with low concrete overhead), but we work with algebraic values and in the non-interactive setting.

Prior work on Arithmetic. [KKW18] tailors [KKW18] for AES-based signatures and Short Integer Solution problem. In contrast, we propose a more general KKW suite of tools. Namely:

Motivated by ZKP of AES (whose S-boxes use $\mathbb{F}_{2^8}$ arithmetic), [dSGMOS19] adapt [KKW18] to operate in the field $\mathbb{F}_{2^8}$. Our approach similarly improves

[KKW18] by adding operations, but we take a more general approach and integrate *ring* operations, efficient dot product, and conversions.

Baum and Nof [BN19] consider an arithmetic field-only version of [KKW18], focusing on interactive instances of ZK arguments of knowledge for instances of the Short Integer Solution problem. [BN19] do not provide conversions between representations. We offer ring arithmetic (matching basic steps of existing programs) and additional efficient operations: dot product and conversions.

We are not aware of other arithmetic ZK constructions that work with KKW.

Balanced NIZK. In Sect. 1, we motivate a setting that prioritizes total proof time, taking into account the ability to run on commodity hardware and the cost of proof generation, transmission, and verification. Among the many recent ZKP systems (cf. Sect. 2), several works belong to this balanced niche, among them Libra [XZZ+19] and Virgo [ZXZS19] (concretely expensive but with linear prover time), Ligero [AHIV17] and [KKW18]. We improve this balanced setting. Among the above works, Libra and Virgo are the most recent, and enjoy linear proof time with reasonable proof size. However, Libra requires trusted setup that we wish to avoid. While Libra and Virgo report faster proof times than Ligero and [KKW18], they were tested on vastly more powerful machines: Libra: Amazon EC2 c5.9xlarge with 70 GB of RAM and Intel Xeon Platinum 8124 m CPU with 3 GHz virtual core, and Virgo: server with 512 GB of DDR3 RAM (1.6 GHz) and 16 3.2 GHz cores (2 threads/core). While Ligero runs on modest hardware, its proof time is super-linear: $O(n \log n)$. Finally, [KKW18] enjoys both linear scaling and concretely efficient proof time, but its proof size is linear as well. Because of [KKW18]'s linear scaling, it is a strong fit for balanced-cost NIZK.

MPC Arithmetic Protocols. We highlight some related works that address arithmetic protocols with properties similar to our own improvements.

[CGH+18] used threshold secret sharing to construct an efficient arithmetic MPC protocol. Like our approach, their protocol provides an efficient vector dot operation. However, their protocol works with fields (and we are interested in supporting efficient ring arithmetic), and further, is not compatible with Boolean circuits.

BLAZE [PS20] proposed a fast three-server privacy-preserving machine learning framework. Their protocol allows both vector dot product operations and conversions between Boolean and arithmetic values. BLAZE is a 3-PC protocol, and does not generalize to arbitrary numbers of parties. Additionally, their arithmetic to Boolean conversions require the use of garbled circuits and are expensive. Our MPC-in-the-head protocol supports similar operations, but allows any number of virtual parties and leverages the ZK prover to efficiently instantiate conversions.

3 Notation

- Let p denote the number of parties: there are p parties P_i for $i \in \{1, \dots, p\}$.
- We consider finite rings $\mathcal{R}$. We denote the bit-length of an $\mathcal{R}$ element by l.
- We write $a \leftarrow_\$ \mathcal{R}$ to denote that a is a uniform element drawn from $\mathcal{R}$.
- λ denotes a uniform Boolean mask.
- x, y, z, etc. denote cleartext Boolean bits that appear in the proof.
- $\widetilde{x}, \widetilde{y}, \widetilde{z}$, etc. denote encrypted Boolean bits. I.e., $\widetilde{x} = x \oplus \lambda$ for some mask λ.
- Capitalized variables are used for ring values: Λ is a uniform ring element mask, X, Y, Z are cleartext ring elements, and $\widetilde{X}, \widetilde{Y}, \widetilde{Z}$ are encrypted ring elements. That is, $\widetilde{X} = X + \Lambda_X$.
- Suppose a is a Boolean value (resp. A is a ring element). Then let $[\![a]\!]$ denote a secret sharing of a (resp. $[\![A]\!]$ of A). That is, suppose each player P_i holds an additive share a_i such that $\bigoplus_{i=1}^{p} a_i = a$ (resp. $\sum_{i=1}^{p} A_i = A$). Then $[\![a]\!]$ is the vector $(a_1, a_2, \dots, a_p)$ (resp. $[\![A]\!] = (A_1, A_2 \dots A_p)$).
- We refer to Boolean (resp. arithmetic) wires with lowercase (resp. uppercase), e.g., value A on wire A. Context disambiguates this slight abuse of notation.

Although our circuits discuss arbitrary rings, we also provide concrete conversion operators between particular rings. Specifically, we construct conversion operations from the Booleans to rings $\mathbb{Z}_k$ for arbitrary k. We also provide the dual conversion from $\mathbb{Z}_k$ to the Booleans.

4 [KKW18] Background

As discussed in Sect. 1, [KKW18] is a powerful MPC-in-the-head NIZKPoK system that takes advantage of a preprocessing-based protocol to achieve efficiency. [KKW18] NIZKPoK's relevant (to us) features are non-interactivity and its low-constant linear scaling in all proof costs, including proof generation, proof transmission, and proof verification.

The proof system is built from two protocols:

1. An MPC protocol with preprocessing secure against up to $p - 1$ semi-honest corruptions. It is this protocol that we improve in our work.
2. An honest-verifier Zero Knowledge (HVZK) protocol. This HVZK protocol uses the above MPC protocol as a black-box. In particular, $\mathcal{P}$ runs the MPC protocol in her head many times (i.e., there are many *instances*) and among many players. This MPC protocol includes a preprocessing and online phase. The verifier $\mathcal{V}$ challenges $\mathcal{P}$ to open the views of players. In some instances, $\mathcal{V}$ inspects the preprocessing given to all players to check it was correctly constructed. In the other instances, $\mathcal{V}$ inspects the preprocessing and online views of all but one MPC party and checks that the views are consistent with the MPC protocol. Thus, $\mathcal{V}$ becomes convinced that $\mathcal{P}$ could not have cheated in either the preprocessing or the online phases.
   Because the MPC protocol is used as a black-box, we can substitute in our own improved protocol.

Due to lack of space, we defer a formal review of [KKW18]'s MPC protocol to a full version of this paper. We present our extensions to their protocol in full detail such that the specific details of their protocol are not essential background.

5 Adding Arithmetic to Boolean Circuits

Our core contribution is an extension to the concretely efficient NIZK proof system of [KKW18]. In particular, we add efficient arithmetic operations as well as conversions between arithmetic and Boolean representations. In this section, we explain how our protocol implements these operations.

We first discuss a pure algebraic version of [KKW18] and explain the relative efficiency of our arbitrary ring operations. Then, we show how to *mix* arithmetic and Boolean representations in a single circuit by adding conversion operations.

5.1 Ring Circuits with Efficient Dot Product

Consider circuits where the gates perform ring operations: i.e. circuits with addition, multiplication, and subtraction gates. Ring [KKW18] is a natural generalization of the Boolean protocol, and we leverage similar preprocessing and online phases. The phases are primarily concerned with propagating the following invariants gate-by-gate through the circuit:

- **Preprocessing invariant.** During the preprocessing phase, $\mathcal{P}$ ensures that each virtual player has a uniformly random additive share of a random mask. For each ring wire A the p virtual players hold the random sharing $[\![\Lambda_A]\!]$ where Λ_A is a uniform element of a finite ring $\mathcal{R}$.
- **Online invariant.** In the online phase, each virtual player holds the value $\widetilde{A} = A + \Lambda_A$. I.e., they each hold the same encryption of A.

These invariants support *correctness*, because they imply that on each output wire A the players hold $A + \Lambda_A$ and $[\![\Lambda_A]\!]$. Thus, the players can broadcast their mask shares and locally reconstruct A. The invariants support *security* against up to $p - 1$ semi-honest corruptions, because they ensure that each cleartext value A is *masked* by Λ_A, and thus no strict subset of players, who together have only a uniform additive *share* of Λ_A, can reconstruct Λ_A. We prove these facts formally in the full version of this paper.

We next step through the supported algebraic operations, showing how our representation propagates the preprocessing and online invariants.

Inputs. Suppose the wire A is an input wire. Our goal is to provide a uniform sharing $[\![\Lambda_A]\!]$ in the preprocessing phase and to provide the encryption $\widetilde{A} = A + \Lambda_A$ to each player in the online phase.

$\mathcal{P}$ sets up the preprocessing invariant by choosing p uniform values and sending one to each virtual player. She distributes $[\![\Lambda_A]\!]$. The sum of these p values is the uniform mask Λ_A. In practice, $\mathcal{P}$ draws these values according to per-player pseudorandom seeds. Thus, P_i's view of all input mask messages (as well as all other pseudo-randomly generated masks, as we discuss for subsequent gates) can be computed from a short seed. The online phase is also straight-forward: $\mathcal{P}$ sends the value $\widetilde{A} = A + \Lambda_A$ to each virtual player.

To use our protocol to construct a proof, $\mathcal{P}$ sends to $\mathcal{V}$ the views of all virtual players save one. For an input wire, this costs l bits of communication where l is the bit-length of an element in $\mathcal{R}$. Thus, the preprocessing phase is communication-free due to seeds and the online phase requires only a single broadcast of a ring element.

Addition. Consider an addition gate with inputs A and B and output C that computes $C \leftarrow A+B$. By induction, the players hold uniform sharings $[\![\Lambda_A]\!]$ and $[\![\Lambda_B]\!]$ in the preprocessing phase and encryptions $\widetilde{A}, \widetilde{B}$ in the online phase. Our goal is to propagate a sharing $[\![\Lambda_C]\!]$ in the preprocessing phase and an encryption $\widetilde{C} = C + \Lambda_C$ such that $C = A + B$.

In the preprocessing phase, we let the preprocessing mask of the output wire be $\Lambda_C = \Lambda_A + \Lambda_B$. Accordingly, the virtual players locally compute their mask shares by adding their respective input shares: together they compute $[\![\Lambda_C]\!] \leftarrow [\![\Lambda_A]\!] + [\![\Lambda_B]\!]$. In the online phase, the players locally add together the masked input values: $\widetilde{C} \leftarrow \widetilde{A} + \widetilde{B}$. The preprocessing and online local computations propagate the respective invariants:

$$\widetilde{A} + \widetilde{B} = (A + \Lambda_A) + (B + \Lambda_B) = (A + B) + (\Lambda_A + \Lambda_B) = C + \Lambda_C = \widetilde{C}$$

Because addition gates do not require the virtual players to communicate, addition gates are communication-free in the proof.

Subtraction. Subtraction is performed in the same manner as addition. Consider a gate with inputs A and B and output C that computes $C \leftarrow A - B$. We let $\Lambda_C = \Lambda_A - \Lambda_B$. During the online phase, the virtual players locally subtract:

$$\widetilde{A} - \widetilde{B} = (A + \Lambda_A) - (B + \Lambda_B) = (A - B) + (\Lambda_A - \Lambda_B) = C + \Lambda_C = \widetilde{C}$$

Like addition gates, subtraction gates are communication-free.

Public Constants. Public constants and multiplication by public constants can be easily encoded in our representation. Due to lack of space, we defer this explanations of these encodings to the full version of this paper.

Multiplication. Consider a multiplication gate with inputs A, B and output C that computes $C \leftarrow AB$. Unfortunately, multiplication cannot be computed as easily as addition. In particular, $\mathcal{P}$ must distribute auxiliary ring elements to the players in the preprocessing phase, and the players must communicate via broadcast in the online phase.

In the preprocessing phase, $\mathcal{P}$ generates a fresh uniform mask Λ_C by drawing uniform ring elements and sending one to each player. In practice, these values are drawn according to the per-player pseudorandom seed, and hence are communication-free in the proof.

Additionally, $\mathcal{P}$ computes the product $\Lambda_{A,B} = \Lambda_A \Lambda_B$ and distributes a uniform sharing $[\![\Lambda_{A,B}]\!]$ to the players. Note that because $\Lambda_{A,B}$ is a fixed value, one virtual player's uniform share cannot be generated from a seed, and must instead be set according to $\Lambda_{A,B}$. Therefore, that player's preprocessing incurs communication in the proof.

In the online phase, the virtual players hold $[\![\Lambda_A]\!]$, $[\![\Lambda_B]\!]$, $[\![\Lambda_{A,B}]\!]$, $[\![\Lambda_C]\!]$, $\widetilde{A}$, and $\widetilde{B}$. They locally compute the following intermediate sharing:

$$[\![S]\!] \leftarrow [\![\Lambda_{A,B}]\!] + [\![\Lambda_C]\!] - \widetilde{A}[\![\Lambda_B]\!] - [\![\Lambda_A]\!]\widetilde{B}$$

Recall that we do not assume ring multiplication is commutative, so we take care to order multiplicands properly. The players broadcast these shares, reconstruct S, and set $\widetilde{C} \leftarrow \widetilde{A}\widetilde{B} + S$. Note that it is safe to reconstruct S, because S is masked by the uniform element Λ_C. This computation properly calculates an encryption of AB:

$$\begin{aligned}\widetilde{A}\widetilde{B} + S &= \widetilde{A}\widetilde{B} + (\Lambda_A\Lambda_B + \Lambda_C - \widetilde{A}\Lambda_B - \Lambda_A\widetilde{B})\\ &= \widetilde{A}\widetilde{B} + (\Lambda_A\Lambda_B + \Lambda_C - (A\Lambda_B + \Lambda_A\Lambda_B) - (\Lambda_A B + \Lambda_A\Lambda_B))\\ &= AB + \Lambda_C = \widetilde{C}\end{aligned}$$

Altogether, this arithmetic product costs $2l$ bits in the proof where l is the bit-size of ring elements: l bits to add the last player's share of $\Lambda_{A,B}$ to the proof message and l bits to send the unopened player's broadcast ($\mathcal{V}$ can compute the opened players' broadcasts himself, so they need not be sent).

Dot Product. In this section, we generalize from multiplication to vector dot product without increasing cost. Without our optimization, such a dot product of n element vectors costs $2ln$ bits in the proof, because the dot product involves n multiplications each costing $2l$ bits. We show that only $2l$ total bits are needed.

Note, a particular player P_i's received messages $\Lambda^i_{A,B}$ and S are only used in an *additive* manner to compute the product. Therefore, if our intent is to add together n products, then we can sum the per-player messages for all products before sending them, avoiding sending all of the summands.

Consider a dot product gate with input vectors $(A_1, \ldots A_n), (B_1, \ldots, B_n)$ and output C. The gate computes:

$$C \leftarrow A_1B_1 + \ldots + A_nB_n$$

By the circuit invariants, players have mask shares for all input vector elements in the preprocessing phase and encryptions all vector elements in the online phase. The players receive auxiliary masks/communicate to evaluate the dot product.

For simplicity, we argue that our improvement works for the sum of two products $A_1B_1 + A_2B_2$, but our argument generalizes to the sum of any number of products. Let S_1, S_2 respectively be the broadcasted terms reconstructed by virtual players when computing A_1B_1 and A_2B_2. Recall that to compute an encryption of A_1B_1 and A_2B_2, the players locally compute $\widetilde{A_1}\widetilde{B_1}+S_1$ and $\widetilde{A_2}\widetilde{B_2}+S_2$. Thus, to compute the overall sum, the players compute:

$$(\widetilde{A_1}\widetilde{B_1} + S_1) + (\widetilde{A_2}\widetilde{B_2} + S_2) = (\widetilde{A_1}\widetilde{B_1} + \widetilde{A_2}\widetilde{B_2}) + (S_1 + S_2)$$

Thus, in the online phase, the players need not broadcast $[\![S_1]\!]$ and $[\![S_2]\!]$ separately: instead they more efficiently broadcast $[\![S_1 + S_2]\!]$. This reduces the communication cost of the online phase. The preprocessing communication cost is similarly improved: to compute $[\![S_1 + S_2]\!]$, the players compute shares of the following expression (where $\Lambda_{C_1}, \Lambda_{C_2}$ are uniformly random):

$$\begin{aligned} S_1 + S_2 &= (\Lambda_{A_1,B_1} + \Lambda_{C_1} - \widetilde{A_1}\Lambda_{B_1} - \Lambda_{A_1}\widetilde{B_1}) + (\Lambda_{A_2,B_2} + \Lambda_{C_2} - \widetilde{A_2}\Lambda_{B_2} - \Lambda_{A_2}\widetilde{B_2}) \\ &= (\Lambda_{A_1,B_1} + \Lambda_{A_2,B_2}) + (\Lambda_{C_1} + \Lambda_{C_2}) - \widetilde{A_1}\Lambda_{B_1} - \Lambda_{A_1}\widetilde{B_1} - \widetilde{A_2}\Lambda_{B_2} - \Lambda_{A_2}\widetilde{B_2} \end{aligned}$$

Thus, in the preprocessing phase it suffices for $\mathcal{P}$ to distribute uniform shares $[\![\Lambda_{A_1,B_1} + \Lambda_{A_2,B_2}]\!]$ instead of distributing both $[\![\Lambda_{A_1,B_1}]\!]$ and $[\![\Lambda_{A_2,B_2}]\!]$. Again, this improves communication. Other values are known to the players a priori or can be generated from seeds. Altogether, the vector dot product of length n vectors costs $2l$ bits in the proof.

To illustrate the importance of this optimization, we compare the communication cost to multiply an $M \times N$ matrix by a $N \times P$ matrix where each matrix entry is an l bit ring element. Without vector dot product, such a multiplication costs $2M \cdot N \cdot P \cdot l$ bits of communication, because the resulting $M \cdot P$ matrix entries are each l bit sums of N products. Our optimization removes the factor N: the total cost is $2M \cdot P \cdot l$ bits.

5.2 Converting Between Boolean and Arithmetic

We have shown how we construct an arithmetic version of the [KKW18] protocol for arbitrary rings. However, many functions (e.g., comparisons and bitwise operations) are more efficiently expressed in a Boolean representation. To get the best of both worlds, we now introduce efficient conversion operations between Boolean and arithmetic representations. We stress that our conversions are not for *arbitrary* rings, but only rings of the form $\mathbb{Z}_k$ for arbitrary $k > 2$.

Single Bit Conversion. Consider a conversion gate with Boolean input wire a and arithmetic output wire A. By induction, the players together hold the mask sharing $[\![\lambda_a]\!]$ in the preprocessing phase and each hold the encryption $a \oplus \lambda_a$ in the online phase. We show how added communication allows the players to propagate the invariant such that they hold $[\![\Lambda_A]\!]$ and $\widetilde{A} = A + \Lambda_A$. In other words, we convert the Boolean encoding to a valid arithmetic encoding.

We start by giving the players preprocessing material. First of all, $\mathcal{P}$ pseudorandomly generates from seeds $[\![\Lambda_A]\!] \in \mathbb{Z}_k$ and distributes it to the players. This new value Λ_A serves only as a mask that ensures security. Additionally, $\mathcal{P}$ deals a uniform sharing $[\![\Lambda_a]\!] \in \mathbb{Z}_k$ such that $\Lambda_a = \lambda_a$ (λ_a is a Boolean value and

$\varLambda_a$ is an arithmetic value)[1]. The role of this auxiliary mask is different than $\varLambda_A$. In particular, the auxiliary mask algebraically eliminates the Boolean mask λ_a. Note that, similar to multiplication, one player's share of $\varLambda_a$ cannot be pseudorandomly generated, because the shares must sum to $\varLambda_a = \lambda_a$, and λ_a is a fixed value. Thus, the translation preprocessing costs l proof bits. We emphasize that although $\lambda_a = \varLambda_a$, the players hold `XOR` shares of λ_a and additive shares of $\varLambda_a$.

In the online phase, we use the following two properties of arbitrary values $x, y \in \{0, 1\}$ when computing modulo $k > 2$:

$$x \oplus y = x + y - 2xy \tag{1}$$

$$x^2 = x \tag{2}$$

To convert, the virtual players locally compute the following intermediate share:

$$[\![S]\!] \leftarrow [\![\varLambda_a]\!](1 - 2\widetilde{a}) + [\![\varLambda_A]\!]$$

Each virtual player then broadcasts her share, reconstructs S, and computes $\widetilde{A} \leftarrow \widetilde{a} + S$. That is, each player outputs a correct arithmetic representation $\widetilde{A} = A + \varLambda_A$. We now show that this computation is correct:

$$\begin{aligned}
\widetilde{a} + S &= (a \oplus \lambda_a) + \varLambda_a(1 - 2(a \oplus \lambda_a)) + \varLambda_A && \widetilde{a} = a \oplus \lambda_a \\
&= (a + \lambda_a - 2a\lambda_a) + \varLambda_a(1 - 2(a + \lambda_a - 2a\lambda_a)) + \varLambda_A && \text{Equation (1)} \\
&= (A + \lambda_a - 2A\lambda_a) + \varLambda_a(1 - 2(A + \lambda_a - 2A\lambda_a)) + \varLambda_A && a = A \\
&= (A + \varLambda_a - 2A\varLambda_a) + \varLambda_a(1 - 2(A + \varLambda_a - 2A\varLambda_a)) + \varLambda_A && \lambda_a = \varLambda_a \\
&= (A + \varLambda_a - 2A\varLambda_a) + \varLambda_a - 2A\varLambda_a - 2\varLambda_a^2 + 4A\varLambda_a^2 + \varLambda_a && \text{distribute} \\
&= A + \varLambda_a - 2A\varLambda_a + \varLambda_a - 2A\varLambda_a - 2\varLambda_a + 4A\varLambda_a + \varLambda_A && \text{Equation (2)} \\
&= A + \varLambda_A = \widetilde{A}
\end{aligned}$$

This conversion costs l bits of communication in the preprocessing phase, because one virtual player is given a non-pseudorandomly chosen value for her share of $\varLambda_a$. Therefore $\mathcal{P}$ sends this non-pseudorandom value to $\mathcal{V}$ to open the view of this player (if this player is not opened, preprocessing is free). In the online phase, we incur l bits of communication, because $\mathcal{P}$ must send the broadcast of the unopened player to $\mathcal{V}$ to convey the views of all opened players.

Multi-bit Conversion. Often, it is useful to convert an entire vector of Boolean values together into a single arithmetic value. Specifically, a Boolean vector is often used as the binary representation of an arithmetic value. Suppose we have a vector of Boolean wires $(a_1, a_2, \ldots, a_l)$ that we would like to convert into an arithmetic value A:

$$A = a_1 + 2a_2 + \ldots + 2^{l-1}a_l$$

Of course, we can use l single-bit conversions, as described above, to construct this sum. However, there are optimizations available.

[1] Here and elsewhere, equality between a Boolean value and an arithmetic value simply indicates that both values are either both 0 or both 1 in their respective ring.

In particular, this naïve translation costs l^2 bits in the online phase. We now reduce this cost to l bits. Recall that each bitwise translation requires the broadcast of an l bit value S. Let S_i be this broadcast value for bit i. The idea is to simply have the players compute and broadcast their shares of the following single value:

$$[\![S_A]\!] = [\![S_1 + 2S_2 + \ldots + 2^{l-1}S_l]\!]$$

The players then reconstruct S_A and locally compute the following:

$$A + \Lambda_A \leftarrow S_A + \widetilde{a_1} + 2\widetilde{a_2} + \ldots + 2^{l-1}\widetilde{a_l}$$

One further optimization is available for integer rings $\mathbb{Z}_{2^n}$ for some n, a particularly useful set of rings for modeling common cleartext computations. Looking again at the definition of $[\![S_A]\!]$, notice that the summand $2^{l-1}S_l$ overflows the ring by 2^{l-1} bits. That is, this summand carries only 1 bit of information in $\mathbb{Z}_{2^n}$. Therefore, at preprocessing time $\mathcal{P}$ need not give the players l bit shares Λ_a, but instead need only send 1 bit shares. In general, for a given vector index i, $\mathcal{P}$ sends $l - i - 1$ bits of preprocessing. In sum, the preprocessing costs $\frac{l^2+l}{2}$ bits.

Altogether, an l bit conversion costs l bits in the online phase and (at most) l^2 bits in the preprocessing phase.

Converting Arithmetic to Boolean. Suppose we wish to convert an arithmetic value A to its binary decomposition $(a_1, a_2, \ldots, a_l)$. Our construction for this conversion is based on a simple observation about Zero Knowledge. In the ZK setting, we can "compute backwards" and then prove what was computed is correct. More precisely, $\mathcal{P}$ simply gives the virtual players encryptions and masks corresponding to $(a_1, a_2, \ldots, a_l)$ as Boolean inputs. Then, the virtual parties use the multi-bit conversion described above to translate $(a_1, a_2, \ldots, a_l)$ to an arithmetic value A'. Note that if $\mathcal{P}$ provides the correct inputs, then $A = A'$. Therefore, the parties compute $A - A'$ and reconstruct the output by broadcasting their mask shares of this result; i.e., they reconstruct 0 if $\mathcal{P}$ did not cheat. By inspecting this output value, $\mathcal{V}$ is convinced that $\mathcal{P}$ provided a correct binary decomposition of the value A.

Altogether, converting an arithmetic value to its binary decomposition costs (1) at most l^2 preprocessing bits for the Boolean to arithmetic conversion, (2) l online bits for the Boolean to arithmetic conversion, (3) l online bits for the input bits given the virtual parties, and (4) l online bits for the unopened player's broadcast of her mask share.

6 Our Semi-honest MPC Protocol

We first explain the 3-round honest-verifier ZK (HVZK) protocol of [KKW18] so that our core theorems can be understood. Our protocol is plugged directly into this HVZK protocol.

$\mathcal{P}$ constructs a large number M (e.g., 500) of commitments to full instances of our protocol. That is, she commits to the views of all p (e.g., 64) players, both in the preprocessing and online protocol phases. Then, $\mathcal{V}$ challenges $\mathcal{P}$ to open a small number τ (e.g., 25) of instances. For each of these τ instances, $\mathcal{P}$ sends to $\mathcal{V}$ the compactly represented views of all except for one player chosen by $\mathcal{V}$. $\mathcal{V}$ checks that the views of these opened players are consistent with the protocol, and thus is convinced (his confidence depends on τ) that $\mathcal{P}$ could not have cheated in the online phase. Because $\mathcal{V}$ only obtains $p-1$ views and because our protocol is secure against $p-1$ semi-honest corruptions, he does not learn $\mathcal{P}$'s witness. In the remaining $M-\tau$ instances, $\mathcal{P}$ opens all players' preprocessing views, where each preprocessing instance is compactly represented as a single master seed. $\mathcal{V}$ checks that these preprocessing views are consistent with the protocol, and thus is convinced (depending on M and τ) that $\mathcal{P}$ could not have cheated in the preprocessing phase either. By configuring M, τ, and p, $\mathcal{P}$ can construct a ZK proof with high soundness.

By plugging into [KKW18]'s HVZK protocol, we achieve a ZK protocol by specifying our protocol as a semi-honest protocol in a preprocessing model. The crucial pieces of our protocol Π are specified in Sect. 5, where we give the individual actions taken by the virtual parties on gate types. While we stress that the discussion given in Sect. 5 is sufficient to understand our approach, we give also a more formal construction in the full version of this paper.

Construction 1. *Π is the p party protocol defined in Sect. 5.*

Our protocol Π makes use of two oracle functionalities. In particular, it first uses the functionality $\mathcal{F}_{\mathtt{pre}}$ to instantiate preprocessing material for the p virtual players and $\mathcal{F}_{\mathtt{inp}}$ which broadcasts $\mathcal{P}$'s masked input to the p players.

Theorems proved in the full version of this paper imply the following:

Theorem 1. *Π correctly implements the semantics of ring circuits and is secure against up to $p-1$ semi-honest corruptions in the $\mathcal{F}_{\mathtt{pre}}$, $\mathcal{F}_{\mathtt{inp}}$ hybrid model.*

This fact, combined with Theorem 2.2 of [KKW18], implies the following:

Theorem 2. *Let M be the total number of repetitions of the proof, τ be the number of proofs checked by $\mathcal{V}$ (hence $M-\tau$ preprocessings are checked), and p be the number of virtual players. Assuming the existence of a collision-resistant hash function and of a secure commitment scheme, the 3-round honest-verifier Zero Knowledge proof protocol of [KKW18] instantiated with Π is an honest-verifier zero-knowledge proof of knowledge with soundness/knowledge error ϵ where:*

$$\epsilon = \max_{M-\tau \leq k \leq M} \left\{ \frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau} \cdot p^{k-M-\tau}} \right\}$$

We point out that our soundness error is equal that of standard [KKW18], because the cheating $\mathcal{P}$'s chances of being caught remain the same. If $\mathcal{P}$ cheats in any preprocessing phase, then she is caught if $\mathcal{V}$ inspects the preprocessing. If $\mathcal{P}$ cheats in any online phase, then she is caught if $\mathcal{V}$ opens the cheating player.

Finally, by applying the Fiat-Shamir transform [FS87] (and assuming a random oracle), this instantiated 3-round HVZK protocol becomes a non-interactive Zero Knowledge Proof of Knowledge (NIZKPoK).

7 Performance Estimation

We improve the [KKW18] approach by adding a suite of efficient ring operations and Boolean conversions. It is immediate from our constructions that we inherit the performance characteristics of [KKW18], including computation and communication costs. Namely, our computation costs are approximately the same per gate (whether arithmetic or Boolean) as [KKW18]. This is because the computations supporting arithmetic or Boolean operations are extremely lightweight, and the main costs involve memory manipulations, which are of similar scope. Our communication cost is the same as [KKW18] for Boolean operations, and is correspondingly increased for arithmetic operations. We outline this cost below, and compare it with that of [KKW18].

Gate Kind	`ADD`	`SUB`	`DOT`	`INPUT`	`OUTPUT`	`B2A`
Preprocessing	0	0	l	0	0	l^2
Online	0	0	l	l	l	l

Fig. 1. The per-instance proof size cost of each gate for an l-bit finite ring. Note that to construct a NIZK, multiple instances must be completed. Realistic numbers of instances vary between $20 - 40$ depending on the parameterization of the NIZK [KKW18].

Gate Costs. Recall, multiple proof *instances* are required to increase soundness. Figure 1 tabulates the per-instance communication cost for each gate. The number of instances needed to achieve a certain security parameter depends on the number of simulated parties. As one practical example, for 128 bits of security with 64 simulated parties, 23 instances are required [KKW18]. Thus, if $p = 64$, the total communication cost of, e.g., a `DOT` gate is $46l$ bits where l is the bit size of the finite ring elements. We emphasize that proof instances that are used only to check preprocessing incur essentially no communication cost because the entire preprocessing is regenerated from a single master seed.

Arithmetic Improvement. We compare our communication with that of [KKW18] on several functions. To understand the performance of classic [KKW18], it suffices to look at Fig. 1 with $l = 1$, recalling that classic [KKW18] does not support vector dot product, only simple Boolean `AND` (costing 2 bits total).

- **Addition.** Suppose that we wish to add numbers in the ring $\mathbb{Z}_{2^l}$ for some l. Boolean circuits can encode this addition efficiently using a ripple-carry adder that costs $l - 1$ `AND` gates or $2l - 2$ proof bits. In contrast, our addition is a free homomorphic operation.
  Adding in most rings other than $\mathbb{Z}_{2^l}$ is extremely costly for the Boolean approach. For example, to compute in the field $\mathbb{Z}_p$ for prime p, the Boolean approach must compute the costly mod$p$ operation which uses l^2 `AND` gates. In contrast, our approach adds elements of this field for free.

- **Multiplication.** Computing a multiplication in the ring $\mathbb{Z}_{2^l}$ consumes l^2 `AND` gates by the schoolbook method. In contrast, we use only l bits. Furthermore, if we consider other rings, the situation skews further in our favor. For example, to multiply in a field $\mathbb{Z}_p$, a Boolean circuit must multiply in a ring large enough that the product does not overflow (i.e. twice the number of bits in p), and then compute mod$p$.
- **Matrix multiplication.** As discussed in Sect. 5, our `DOT` gate excels as efficiently computing matrix multiplications. In sum, multiplying a $M \times N$ matrix by a $N \times P$ matrix of l-bit elements requires $2M \cdot P \cdot l$ bits. That is, computing a matrix multiplication requires only twice as many bits as are needed to *represent* the output matrix. To compare the performance of classic [KKW18], assume that the matrix elements are elements of $\mathbb{Z}_{2^l}$. Thus, Boolean algebra can encode a multiplication using l^2 `AND` gates or $2l^2$ proof bits. The total proof cost for a matrix multiplication in the specified dimensions is thus $2l^2 \cdot M \cdot N \cdot P$ bits. Thus, our approach improves by factor $N \cdot l$. When concrete parameters are considered, it becomes clear that this improvement is substantial. For example, to multiply 100×100 square matrices of 32 bit integers, we require $3200\times$ less communication.

Acknowledgements. This work was supported in part by NSF award #1909769, by a Facebook research award, and by Georgia Tech's IISP cybersecurity seed funding (CSF) award.

References

[AHIV17] Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: lightweight sublinear arguments without a trusted setup. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 2087–2104. ACM Press, October/November 2017

[BBHR18] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018). https://eprint.iacr.org/2018/046

[BCG+13] Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_6

[BFS20] Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 677–706. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_24

[BG93] Bellare, M., Goldreich, O.: On defining proofs of knowledge. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 390–420. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_28

[BN19] Baum, C., Nof, A.: Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. Cryptology ePrint Archive, Report 2019/532 (2019). https://eprint.iacr.org/2019/532

[CDG+17] Chase, M.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 1825–1842. ACM Press, October/November 2017

[CFH+15] Costello, C., et al.: Geppetto: versatile verifiable computation. In 2015 IEEE Symposium on Security and Privacy, pp. 253–270. IEEE Computer Society Press, May 2015

[CGH+18] Chida, K., et al.: Fast large-scale honest-majority MPC for malicious adversaries. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10993, pp. 34–64. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96878-0_2

[CHM+20] Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 738–768. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_26

[COS20] Chiesa, A., Ojha, D., Spooner, N.: Fractal: post-quantum and transparent recursive proofs from holography. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 769–793. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_27

[DP92] De Santis, A., Persiano, G.: Zero-knowledge proofs of knowledge without interaction (extended abstract). In: 33rd FOCS, pp. 427–436. IEEE Computer Society Press, October 1992

[dSGMOS19] de Saint Guilhem, C.D., De Meyer, L., Orsini, E., Smart, N.P.: BBQ: using AES in picnic signatures. Cryptology ePrint Archive, Report 2019/781 (2019). https://eprint.iacr.org/2019/781.pdf

[EFKP20] Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: SPARKs: succinct parallelizable arguments of knowledge. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 707–737. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_25

[FS87] Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12

[GGPR13] Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_37

[GMO16] Giacomelli, I., Madsen, J., Orlandi, C.: ZKBoo: faster zero-knowledge for Boolean circuits. In: Holz, T., Savage, S. (eds.) USENIX Security 2016, pp. 1069–1083. USENIX Association, August 2016

[GMR85] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: 17th ACM STOC, pp. 291–304. ACM Press, May 1985

[GMW91] Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. J. ACM **38**(3), 690–728 (1991)

[Gro16] Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11

[HK20] Heath, D., Kolesnikov, V.: Stacked garbling for disjunctive zero-knowledge proofs. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12107, pp. 569–598. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45727-3_19

[IKOS07] Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Johnson, D.S., Feige, U. (eds.) 39th ACM STOC, pp. 21–30. ACM Press, June 2007

[JKO13] Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In: Sadeghi, A.-R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013, pp. 955–966. ACM Press, November 2013

[Kil92] Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th ACM STOC, pp. 723–732. ACM Press, May 1992

[KKW18] Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018, pp. 525–537. ACM Press, October 2018

[MBKM19] Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 2111–2128. ACM Press, November 2019

[Mic94] Micali, S.: CS proofs (extended abstracts). In: 35th FOCS, pp. 436–453. IEEE Computer Society Press, November 1994

[PHGR13] Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy, pp. 238–252. IEEE Computer Society Press, May 2013

[PS20] Patra, A., Suresh, A.: BLAZE: blazing fast privacy-preserving machine learning. In: NDSS 2020. The Internet Society (2020)

[XZZ+19] Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: succinct zero-knowledge proofs with optimal prover computation. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 733–764. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_24

[ZCD+17] Zaverucha, G.: Picnic. Technical report, National Institute of Standards and Technology (2017). https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions

[ZXZS19] Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. Cryptology ePrint Archive, Report 2019/1482 (2019). https://eprint.iacr.org/2019/1482