Concretely-Efficient Zero-Knowledge Arguments for Arithmetic Circuits and Their Application to Lattice-Based Cryptography

Carsten Baum[1] and Ariel Nof[2(✉)]

[1] Aarhus University, Aarhus, Denmark
carsten.baum@outlook.com
[2] Technion, Haifa, Israel
arie.nof@cs.biu.ac.il

Abstract. In this work we present a new interactive Zero-Knowledge Argument of knowledge for general arithmetic circuits. Our protocol is based on the "MPC-in-the-head"-paradigm of Ishai et al. (STOC 2009) and follows the recent "MPC-in-the-head with Preprocessing" as proposed by Katz, Kolesnikov and Wang (ACM CCS 2018). However, in contrast to Katz et al. who used the "cut-and-choose" approach for pre-processing, we show how to incorporate the well-known "sacrificing" paradigm into "MPC-in-the-head", which reduces the proof size when working over arithmetic circuits. Our argument system uses only lightweight symmetric-key primitives and utilizes a simplified version of the so-called SPDZ-protocol.

Based on specific properties of our protocol we then show how it can be used to construct an efficient Zero-Knowledge Argument of Knowledge for instances of the Short Integer Solution (SIS) problem. We present different protocols that are tailored to specific uses of SIS, while utilizing the advantages of our scheme. In particular, we present a variant of our argument system that allows the parties to sample the circuit "on the fly", which may be of independent interest.

We furthermore implemented our Zero-Knowledge argument for SIS and show that using our protocols it is possible to run a complete interactive proof, even for general SIS instances which result in a circuit with $>10^6$ gates, in *less than 0.5 s*.

1 Introduction

Zero-Knowledge Arguments of Knowledge (ZKAoK) are interactive protocols that allow a computationally bounded prover to convince a verifier that he knows a witness for a certain statement, without revealing any further information about the witness. Since their introduction in the 80's [GMR89] these protocols have been important building blocks for applications in cryptography. While solutions for very specific languages have been plentiful, many applications require the use of arbitrary (algebraic) circuits in order to prove complex relationships. For example, proving that two homomorphic commitments contain the same committed message is generally an easy task, while proving knowledge of a preimage of a SHA-256 hash requires more generic solutions. Recent years saw a variety of different techniques which aim at providing such ZKAoK (see [PHGR16,GMO16,AHIV17,BBHR19,WTS+18,BCR+19] to just name a few), varying in terms of argument size, prover/verification time, interaction and assumptions. While many of these systems perform very well with large witnesses and circuit sizes, none of them are a one-size-fits-all solution.

As an example, consider the so-called Short Integer Solution (SIS) problem. Here, a verifier has a matrix $\mathbf{A}$ and a vector $\mathbf{t}$ while the prover wants to prove knowledge of a secret $\mathbf{s}$ such that $\mathbf{t} = \mathbf{A}\mathbf{s} \bmod q$ and $||\mathbf{s}||_\infty \leq \beta$. SIS and related problems are crucial building blocks for post-quantum lattice-based cryptography and constructing efficient ZKAoK with a small communication complexity and low prover's running time has long been a problem: the soundness error of current special-purpose protocols is constant, meaning that the arguments have to be repeated many times to actually be convincing to a verifier. A particular, non-standard approach has been suggested by Bendlin & Damgård [BD10], who were the first to examine arguments of knowledge for SIS using generic proof systems. They observed that for certain argument schemes, the function that is computed to validate a SIS instance has both a very low multiplication depth and low total number of multiplications, if the secret $\mathbf{s}$ is a binary vector. However, many general ZKAoK systems only provide asymptotic efficiency, meaning that they require the circuit to be big before their strengths play out [BBC+18]. Moreover, many of these approaches achieve sub-linear communication complexity at the cost of high prover's running time [AHIV17,PHGR16]. Other approaches are insecure to post-quantum attacks [WTS+18,MBKM19,BCC+16,PHGR16] or rely on knowledge assumptions that are poorly understood.

1.1 'MPC-in-the-Head' and Preprocessing

The "MPC-in-the-head" paradigm is a general technique which is used in our construction. Before outlining our contributions, we first describe this paradigm.

MPC or Secure Multiparty Computation describes a type of interactive protocol which allows to securely compute functions on secret data. No information is leaked beyond the output of the function with correctness even in the presence of dishonest participants.

MPC-in-the-head was introduced by Ishai et al. [IKOS07] as a technique to construct generic zero-knowledge proofs from MPC protocols. Here, the statement to be proven is rewritten into a circuit C, which outputs y if and only if its input w is a correct witness for the statement to be proven. The prover simulates all parties of an MPC protocol as well as their interaction *in his head*. These parties obtain a secret-sharing of the witness w as their input, run a protocol to evaluate C and send the outputs to the verifier. Moreover, the prover commits to the inputs as well as randomness and exchanged messages of each party separately, and opens a verifier-chosen subset of these commitments to the verifier. The verifier then checks if these parties were simulated correctly by the prover and that the messages and the outputs are consistent. On a very high level, this is a proof of the statement if the MPC scheme is robust against active attacks, and it is zero-knowledge due to the privacy of it.

Preprocessing is a widely used optimization of practical MPC schemes. Here, each party begins the actual protocol with shares of correlated randomness, which is itself independent of the inputs of the protocol. This correlated randomness is then used to speed up the actual computation, and due to its independence of the inputs it can be computed ahead of time. To the best of our knowledge, the first MPC-in-the-head scheme that uses preprocessing was introduced in [KKW18].

1.2 Our Contributions

In this work, we construct a new practically efficient ZKAoK for arithmetic circuits together with a multitude of techniques and apply these to construct interactive arguments for SIS. Our scheme is based on the "MPC-in-the-head" approach and uses only symmetric-key primitives. It has an argument size that only depends on non-linear gates of the circuit C and low prover running time. We implemented our construction and report on its practicality. In more details:

'MPC-in-the-Head' with Preprocessing. We first generalize the idea of [KKW18] to work over arithmetic circuits using a variant of the SPDZ MPC protocol [DPSZ12, LN17] and provide a formal proof of security to their "cut-and-choose" preprocessing heuristic. Then, we present a new construction where we replace the "cut-and-choose" mechanism with a "sacrificing"-based approach. While both approaches have similar cost per MPC instance, our "sacrificing"-based approach yields a smaller cheating probability, which means that the number of MPC instances simulated in the proof can be significantly smaller, thus *reducing* the overall communication footprint. Our scheme is highly flexible in its choice of parameters. In particular, by changing the number of parties in the underlying MPC protocol, one can alternate between achieving low communication and low running time. Our construction only requires efficient standard symmetric primitives, and thus is plausibly post-quantum secure even in the non-interactive case [DFMS19]. The two constructions can be found in Sect. 3.

Application to SIS. The MPC scheme which we use in our construction allows to perform additions and multiplications with public values "for free", meaning those do not have an impact on the size of the argument. In the SIS problem the verification of the input of the prover consists of computing a product with a public matrix $\mathbf{A}$ *and* a proof that the secret $\mathbf{s}$ contains bounded values, so the first part comes essentially for free. We initially tweak the approach of [BD10] and only allow $\mathbf{s}$ to consist of bits, which allows for a very fast argument of size using one square gate per element of $\mathbf{s}$. Then, we show how to handle more general distributions of $\mathbf{s}$ and introduce some specific optimizations to reduce communication and computation. In particular, we show how to adapt advanced techniques such as rejection sampling into the MPC-in-the-head framework, which yields a circuit with only linear gates. This is described in Sect. 5.

Experimental Results. We implemented our zero-knowledge protocol for the Binary SIS problem (i.e., where the secret $\mathbf{s}$ is a vector of bits) and ran extensive experiments with various sets of parameters – both for the SIS problem and for the simulated MPC protocol. For a 61-bit field and a matrix $\mathbf{A}$ of size 1024×4096 (which suffices for many applications such as encryption or commitments), we are able to run our argument in 1.2 s for 40-bits of statistical security when working with a single thread. When utilizing 32 threads, this reduces to only 250 ms. This shows that general lattice-based ZK arguments (which do not rely on structured lattices) are practical and can be used in real-world applications. To the best of our knowledge, we are also the first to implement ZK arguments for general SIS. The results and all the details can be found in Sect. 6.

Sampling Circuits on the Fly. A major source of optimizations to our application is the fact that our "MPC-in-the-head" protocol allows the prover and the verifier to negotiate the circuit C *during the protocol*, under certain circumstances. This fact is used by us to construct circuits "on the fly" with fewer non-linear gates, which helps to reduce the argument size. Thus, as an additional contribution of this work, we provide formal definitions for an argument system where the circuit is sampled jointly by the prover and the verifier during the execution and show how to incorporate this into our protocols. This is described in Sect. 4.

Full Version. Due to space limitations, most proofs are deferred to the full version of this work [BN19] which can be found on eprint.

2 Preliminaries

Unless stated otherwise, operations in this work are carried out over the field $\mathbb{F} = \mathbb{F}_q$ for an odd prime q. $\mathbb{F}_q$-elements are identified by the interval $[-(q-1)/2, (q-1)/2]$. $\mathbb{B}$ denotes the set $\{0,1\}$ while $[n]$ stands for $\{1,\ldots,n\}$. We use λ as the computational and κ as the statistical security parameters, and generally assume that $q \approx \mathit{poly}(\lambda,\kappa)$. We use bold lower-case letters such as $\mathbf{s}$ to denote a vector and bold upper-case letters like $\mathbf{A}$ for matrices. We let $\mathbf{s}[i]$ denote the ith component of the vector $\mathbf{s}$.

2.1 Programming Model

Our notation for the circuits that we use in this paper will be similar to [BHR12]. The circuit $C = (n_{\mathtt{in}}, n_{\mathtt{out}}, n_C, L, R, F)$ is defined over $\mathbb{F}$, and each wire w will hold a value from $\mathbb{F}$ or $\perp$ initially. C has $n_{\mathtt{in}}$ input wires, $n_{\mathtt{out}}$ output wires and

$n_C \geq n_{\mathtt{in}} + n_{\mathtt{out}}$ wires in total. We define $\mathcal{I} = \{1, \ldots, n_{\mathtt{in}}\}, \mathcal{W} = \{1, \ldots, n_C\}$ and $\mathcal{O} = \{n_C - n_{\mathtt{out}} + 1, \ldots, n_C\}$. The circuit has $n_{\mathtt{gates}} = n_C - n_{\mathtt{in}}$ gates in total and we define the set $\mathcal{G} = \{n_{\mathtt{in}} + 1, \ldots, n_C\}$.

We define functions $L : \mathcal{G} \to \mathcal{W}$, $R : \mathcal{G} \to \mathcal{W} \cup \{\perp\}$ such that $L(x) < x$ as well as $L(x) < R(x) < x$ if $R(x) \neq \perp$ (i.e., the function $L(x)$ returns the index of the left input wire of the gate whereas the function $R(x)$ returns the index of the right input wire if it exists). The function $F : \mathcal{G} \times \mathbb{F} \times (\mathbb{F} \cup \{\perp\}) \to \mathbb{F}$ determines the function which is computed by each gate.

The algorithm $\mathtt{eval}(C, \mathbf{x})$ with $\mathbf{x} \in \mathbb{F}^{n_{\mathtt{in}}}$ is defined as follows:

1. For $i \in [n_{\mathtt{in}}]$ set $w_i \leftarrow \mathbf{x}[i]$.
2. For each $g \in \mathcal{G}$ set $l \leftarrow L(g), r \leftarrow R(g)$ and then $w_g \leftarrow F(g, l, r)$.
3. Output $\mathbf{y} = (w_{n_C - n_{\mathtt{out}}+1}, \ldots, w_{n_C})^\top$.

We further restrict F to compute certain functions only: (i) **Add:** On input x_1, x_2 output $x_1 + x_2$, (ii) **Mult:** On input x_1, x_2 output $x_1 \times x_2$, (iii) **CAdd:** On input x and for the hard-wired a output $x + a$, (iv) **CMult:** On input x and for the hard-wired a output $x \times a$; and (v) **Square:** On input x output x^2. We say that $C(\mathbf{x}) = \mathbf{y}$ if $\mathtt{eval}(C, \mathbf{x})$ returns the value $\mathbf{y} \in \mathbb{F}^{n_{\mathtt{out}}}$. We denote by n_{mul} and n_{sq} the number of multiplication and square gates in the circuit.

2.2 Zero-Knowledge Arguments of Knowledge

Let TM be an abbreviation for Turing Machines. An iTM is defined to be an interactive TM, i.e. a Turing Machine with a special communication tape and a PPT TM is a probabilistic polynomial-time TM. Let $L_R \subseteq \mathbb{B}^*$ be an NP language and R be its related NP-relation for circuits over $\mathbb{F}$. Thus $(x = (C, \mathbf{y}), \mathbf{w}) \in R$ iff $(C, \mathbf{y}) \in L_R$ and $\mathtt{eval}(C, \mathbf{w}) = \mathbf{y}$. We write $R_x = \{\mathbf{w} \mid (x, \mathbf{w}) \in R\}$ for the set of witnesses for a fixed x.

Honest Verifier Zero Knowledge Argument of Knowledge (HVZKAoK). Assume $(\mathcal{P}, \mathcal{V})$ is a pair of PPT iTMs and let $\xi : \mathbb{B}^* \to [0, 1]$ be a function. We say that $(\mathcal{P}, \mathcal{V})$ is a *zero-knowledge argument of knowledge* for the relation R with *knowledge error* ξ if the following properties hold:

Completeness: If $\mathcal{P}$ and $\mathcal{V}$ follow the protocol on input $x \in L_R$ and private input $\mathbf{w} \in R_x$ to $\mathcal{P}$, then $\mathcal{V}$ always outputs 1.

Knowledge Soundness: There exists a probabilistic algorithm $\mathcal{E}$ called the *knowledge extractor*, such that for every interactive prover $\hat{\mathcal{P}}$ and every $x \in L_R$, the algorithm $\mathcal{E}$ satisfies the following condition: let $\delta(x)$ the probability that $\mathcal{V}$ accepts on input x after interacting with $\hat{\mathcal{P}}$. If $\delta(x) > \xi(x)$, then upon input $x \in L_R$ and oracle access to $\hat{\mathcal{P}}$, the algorithm $\mathcal{E}$ outputs a vector $\mathbf{w} \in R_x$ in expected number of steps bounded by $O(\frac{1}{\delta(x) - \xi(x)})$.

Honest Verifier Zero-Knowledge: Let $\mathsf{view}_{\mathcal{V}}^{\mathcal{P}}(x, \mathbf{w})$ be a random variable describing the random challenge of $\mathcal{V}$ and the messages $\mathcal{V}$ receives from $\mathcal{P}$ with input $\mathbf{w}$ during the joint computation on common input x. Then, there exists a PPT simulator $\mathcal{S}$, such that for all $x \in L_R, \mathbf{w} \in R_x$: $\mathcal{S}(x) \approx_c \mathsf{view}_{\mathcal{V}}^{\mathcal{P}}(x, \mathbf{w})$.

This definition suffices, since public-coin protocols like the protocols we consider in this work, which satisfy the above properties, can be easily transformed to protocols which are zero-knowledge in general by having the verifier commit to its challenges at the beginning of the protocol. As is well known, it is possible to obtain a non-interactive zero-knowledge argument of knowledge (NIZKAoK) from any HVZKAoK via the Fiat-Shamir transformation [FS86].

2.3 Commitments and Collision-Resistant Hash Functions

We use *Commitments* and *Collision-Resistant* Hash Functions (CRHF) as buildings blocks in our constructions and thus introduce them now briefly. A commitment scheme allows one party to commit to a message m by sending a commitment value which satisfies the following two properties: (i) *Hiding:* the commitment reveals nothing about m.; and (ii) *Binding:* it is (computationally) infeasible for the committing party to open a committed message m to different message $m' \neq m$. In this work, we assume that the commitment scheme is instantiated using a cryptographic hash function such as e.g. SHA-256, which we model as a Random Oracle[1] for the purpose of giving a proof of security.

A Collision-Resistant Hash Function (CRHF) is an efficiently computable function H for which it is "hard" to find x, x' such that $H(x) = H(x')$. Usually, CRHFs are used to compress a long message into a short digest and thus for almost all messages a collision exists. CRHFs require that a collision is hard to find for any PPT algorithm.

2.4 The Short Integer Solution Problem

We will now formalize the SIS problem, which was already informally introduced in the introduction. $\mathbb{F}_q$ is the base field of the argument system. At the same time, the characteristic q will also be the modulus of the SIS instance which is defined over $\mathbb{Z}_q$. To define the Short Integer Solution problem, let $n, m \in \mathbb{N}$ be such that $n \ll m$. We naturally embed $\mathbb{Z}_q$ into $\mathbb{Z}$ by identifying each $\mod q$-number with an element from the interval $[-\frac{q-1}{2}, \frac{q-1}{2}] \subset \mathbb{Z}$. We thereby let $||\mathbf{s}||_\infty$ be the ∞-norm of the embedding of $\mathbf{s} \in \mathbb{Z}_q^m$ into the module $\mathbb{Z}^m$. Define $S_\beta^m \subset \mathbb{Z}_q^m$ as the subset of m-element vectors with ℓ_∞-norm $\leq \beta$.

Definition 1 (Short Integer Solution (SIS)). *Let m, n, q be as above and $\beta \in \mathbb{N}$. Given $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{t} \in \mathbb{Z}_q^n$, the (inhomogeneous) SIS-problem is to find $\mathbf{s} \in \mathbb{Z}_q^m$ such that $\mathbf{t} = \mathbf{A}\mathbf{s} \mod q$ and $\mathbf{s} \in S_\beta^m$.*

We collect such $(\mathbf{A}, \mathbf{s}, \mathbf{t})$ that fulfill Definition 1 in an NP-relation

$$R_{\mathtt{SIS}}^{m,n,q,\beta} = \{(x, w) = ((\mathbf{A}, \mathbf{t}), \mathbf{s}) \,|\, \mathbf{s} \in S_\beta^m \wedge \mathbf{A} \in \mathbb{F}_q^{n \times m} \wedge \mathbf{t} = \mathbf{A}\mathbf{s}\}.$$

[1] This is to obtain the smallest possible argument size while avoiding attacks such as [DN19]. A generalization of our scheme without this assumption can be obtained by generating the randomness for commitments independent of the message.

In practice, one often encounters proofs that do not show exactly that $\mathbf{s} \in S_\beta^m$ even though the prover has such a value as witness. Instead, they guarantee that the bound might be a bit bigger, by a factor at most ω which usually is called *slack*. We have that $R_{\mathtt{SIS}}^{m,n,q,\beta} \subseteq R_{\mathtt{SIS}}^{m,n,q,\omega \cdot \beta}$ if $\omega \geq 1$, so any honest prover will still make the verifier accept if it proves $R_{\mathtt{SIS}}^{m,n,q,\omega \cdot \beta}$ instead. For simplicity, we also consider an instance of SIS where $\mathbf{s}$ is binary.

Definition 2 (Binary-SIS). *Let m, n, q be defined as above. Given $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{t} \in \mathbb{Z}_q^n$, the (inhomogeneous) Binary SIS-problem is to find $\mathbf{s} \in \mathbb{B}^m$ such that $\mathbf{t} = \mathbf{A}\mathbf{s} \mod q$.*

This Binary-SIS problem is not uncommon and e.g. [BD10,KTX08] used it. Its relation $R_{\mathtt{B-SIS}}^{m,n,q}$ can be defined similarly as $R_{\mathtt{SIS}}^{m,n,q,\beta}$.

3 Honest Verifier Arguments of Knowledge for Arithmetic Circuits

In this section, we introduce our honest verifier zero-knowledge argument of knowledge (HVZKAoK) protocols for satisfiability of arithmetic circuits. We begin by describing the underlying MPC protocol to securely compute an arithmetic circuit. Then, we present two HVZKAoKs based on the MPC protocol - one that relies on the "cut–and–choose" paradigm and one that uses "sacrificing". While the first is a direct extension of a recent work of [KKW18], the second one is completely new to the best of our knowledge.

3.1 The MPC Protocol

Our MPC protocol is a simplified version of the SPDZ[2] protocol [DPSZ12]. Let N denote the number of parties and let $P_1, \ldots, P_N$ denote the parties participating in the protocol.

Secret Sharing Scheme. Let $[\![x]\!]$ denote an additive sharing of x, i.e. a sharing of x consists of random $x_1, \ldots, x_N \in \mathbb{F}_q$ such that $x = \sum_{i \in [N]} x_i$, where P_i holds x_i. We define the following operations on shares:

- $\mathsf{open}([\![x]\!])$**:** To reveal the secret x each party broadcasts its share x_i. Upon receiving x_j from each P_j, P_i sets $x = \sum_{j \in [N]} x_j$.
- $[\![x]\!] + [\![y]\!]$**:** Given two shares x_i and y_i of x and y, each party P_i defines $x_i + y_i$ as its share of the result.
- $\sigma + [\![x]\!]$**:** Given a sharing $[\![x]\!]$ and a public constant σ, P_1 defines $x_1 + \sigma$ as its new share while other parties' shares remain the same.
- $\sigma \cdot [\![x]\!]$**:** Given a sharing $[\![x]\!]$ and a public constant σ, each party P_i defines $\sigma \cdot x_i$ as its share of the product.

[2] It works like SPDZ in the sense that it considers dishonest majority, uses an additive secret sharing and multiplication triples, but without the information-theoretic MAC.

Multiplications. We say that $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ is a random multiplication triple if a and b are random and $c = a \cdot b$. To multiply $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$ using a preprocessed random triple $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$, the parties do the following:

1. The parties compute $\llbracket \alpha \rrbracket = \llbracket x \rrbracket - \llbracket a \rrbracket$ and $\llbracket \beta \rrbracket = \llbracket y \rrbracket - \llbracket b \rrbracket$.
2. The parties run $\mathsf{open}(\llbracket \alpha \rrbracket)$ and $\mathsf{open}(\llbracket \beta \rrbracket)$ to obtain α and β.
3. Each party computes $\llbracket z \rrbracket = \llbracket c \rrbracket - \alpha \cdot \llbracket b \rrbracket - \beta \cdot \llbracket a \rrbracket + \alpha \cdot \beta$.

The above is a well-known [Bea91] technique and works because

$$\begin{aligned}\llbracket z \rrbracket &= \llbracket c \rrbracket - \alpha \cdot \llbracket b \rrbracket - \beta \cdot \llbracket a \rrbracket + \alpha \cdot \beta \\ &= \llbracket ab \rrbracket - (x-a) \cdot \llbracket b \rrbracket - (y-b) \cdot \llbracket a \rrbracket + (x-a) \cdot (y-b) \\ &= \llbracket xy \rrbracket\end{aligned}$$

Squaring. We say that $(\llbracket b \rrbracket, \llbracket d \rrbracket)$ is a random square if b is random and $d = b^2$. To compute $\llbracket x^2 \rrbracket$ given $\llbracket x \rrbracket$ using a preprocessed $(\llbracket b \rrbracket, \llbracket d \rrbracket)$, the parties do the following:

1. The parties compute $\llbracket \alpha \rrbracket = \llbracket x \rrbracket - \llbracket b \rrbracket$.
2. The parties run $\mathsf{open}(\llbracket \alpha \rrbracket)$ to obtain α.
3. Each party computes $\llbracket z \rrbracket = \alpha \cdot (\llbracket x \rrbracket + \llbracket b \rrbracket) + \llbracket d \rrbracket$.

Note that the above holds since

$$\begin{aligned}\llbracket z \rrbracket = \alpha \cdot (\llbracket x \rrbracket + \llbracket b \rrbracket) + \llbracket d \rrbracket &= (x-b) \cdot (\llbracket x \rrbracket + \llbracket b \rrbracket) + \llbracket b^2 \rrbracket \\ &= \llbracket x^2 - b^2 + b^2 \rrbracket = \llbracket x^2 \rrbracket.\end{aligned}$$

The Protocol. The above building blocks can easily be combined to securely run $\mathtt{eval}(\cdot)$ on a circuit C: after the inputs are secret-shared using $\llbracket \cdot \rrbracket$, the parties apply G as defined in Sect. 2.1 consecutively to the shares. That is, addition gates and multiplication/addition by-a-public-constant gates are computed locally, whereas multiplication and square gates are computed using the above sub-protocols.

Security. For our purpose of using a MPC protocol to establish our zero-knowledge argument, the used protocol only needs to be secure in the presence of a semi-honest adversary. Furthermore, it suffices for the protocol to be secure in the client-server broadcast model, i.e., when the parties who run the protocol (the servers) do not hold input and do not see the final output, but rather receive shares of the inputs from the clients, perform the computation by only local computation as well as sending broadcast messages to each other, and then send the output shares back to the clients.

Formally, let $\mathcal{F}_{\mathtt{tr}}$ and $\mathcal{F}_{\mathtt{sq}}$ be ideal functionalities that provide the parties with random multiplication triples and squares. We define $\mathsf{view}_{I,\pi}^{\mathcal{F}_{\mathtt{tr}},\mathcal{F}_{\mathtt{sq}}}(C)$ to be the view of a subset of parties I during the execution of a protocol π on the circuit C, in the $(\mathcal{F}_{\mathtt{tr}}, \mathcal{F}_{\mathtt{sq}})$-hybrid model and in the client-server model. The view consists of the input shares, the correlated randomness they receive from the functionalities and the messages they obtain from the other parties while evaluating C. The security of π is stated in Theorem 1, which is proven in the full version.

Theorem 1. *Let C be an arithmetic circuit over the field $\mathbb{F}$ and let π be the protocol described above. Then, for every subset of parties $I \subset \{P_1, \ldots, P_N\}$ with $|I| \le N-1$, there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ such that $\{\mathcal{S}(I, C)\} \equiv \{\mathsf{view}_{I,\pi}^{\mathcal{F}_{\mathtt{tr}},\mathcal{F}_{\mathtt{sq}}}(C)\}$.*

3.2 HVZKAoK Protocol Using Cut and Choose

We now explain our first HVZKAoK protocol $\Pi_{\mathsf{c\&c}}$, which is based on the MPC protocol from Sect. 3.1, and which relies on the cut–and–choose technique to generate correct random multiplication triples and squares. The formal description appears in the full version.

The idea behind the protocol is that the prover $\mathcal{P}$ proves its knowledge of $\mathbf{w}$ such that $C(\mathbf{w}) = \mathbf{y}$ by simulating a secure N-party computation of the circuit over an additive sharing of $\mathbf{w}$, using the MPC protocol described above. Since $\mathcal{P}$ knows the input and thus the values on each wire of the circuit, it can simulate the execution "in the head". Since our MPC protocol uses random triples and squares supplied by the ideal functionalities $\mathcal{F}_{\mathtt{tr}}$ and $\mathcal{F}_{\mathtt{sq}}$, the prover $\mathcal{P}$ needs to play their role as well. Clearly, $\mathcal{P}$ may try to cheat in the simulated computation, aiming to cause the verifier $\mathcal{V}$ to accept false statements. This is prevented by having $\mathcal{V}$ challenging $\mathcal{P}$ in two ways. First, after $\mathcal{P}$ has committed to M sets of random triples and squares, $\mathcal{V}$ randomly selects τ of them, which are then opened to it. The remaining $M - \tau$ sets of the pre-processed data are used to support $M - \tau$ circuit computations - each with different randomness. The prover $\mathcal{P}$ performs these computations and commits to the views of the parties, to be then challenged for the second time by $\mathcal{V}$. In this second challenge, the verifier chooses a random subset of $N-1$ parties in each execution, whose views are opened and tested for consistency. If these two tests pass successfully and the output of the circuit is $\mathbf{y}$, then $\mathcal{V}$ outputs acc. Observe that $\mathcal{V}$ cannot learn any information about the witness $\mathbf{w}$ during the protocol: the opened pre-processing executions reveal only random data which is thrown away afterwards, and the $N-1$ views that are opened do not reveal anything since the MPC protocol is resilient to $N-1$ semi-honest parties. In more details, in Round 1, $\mathcal{P}$ commits to M pre-processing executions. A major source of saving here is using pseudo-randomness instead of pure randomness. Specifically, $\mathcal{P}$ chooses a seed sd_e for each execution e, from which it derives the seeds $\mathsf{sd}_{e,i}$ for each party P_i. These seeds are used to generate all the random shares held by P_i throughout the computation. Now, if execution e is selected to be tested by $\mathcal{V}$ in Round 2, then $\mathcal{P}$ can just send sd_e to $\mathcal{V}$, thereby saving communication. For the $M - \tau$ preprocessings which are used in the on-line execution in Round 3, $\mathcal{P}$ cannot send the master seed but rather will have to send $N-1$ seeds of the $N-1$ parties chosen to be opened by $\mathcal{V}$ in Round 4. Thereby the data of one of the parties is kept secret. Observe, however, that not all the data held by the parties is random. In particular, when generating a multiplication triple $[\![a_{e,k}]\!], [\![b_{e,k}]\!], [\![c_{e,k}]\!]$ (e is the execution index and k is the index of the gate for which this triple is consumed), one can use the seeds of the parties to generate the sharing of $a_{e,k}$ and $b_{e,k}$, but once these are fixed, so is $c_{e,k} = a_{e,k} \cdot b_{e,k}$. Therefore, when generating the sharing of $c_{e,k}$, it is necessary to "fix" the initial sharing derived from the random seeds. To obtain this, the prover also commits to the offset $\Delta_{e,k}$ for each execution e and multiplication gate g_k, which is added to the initial random sharing $[\![c_{e,k}]\!]$. The same applies when generating random squares. Similarly, when the sharings of the inputs are generated in Round 3, $\mathcal{P}$ can use the seeds of the parties to derive their shares, and then adjust this initial sharing by adding the offset (denoted by $\phi_{e,k}$) to obtain a correct sharing of the given input. Thus, $\mathcal{P}$ must commit to the offset on each input wire as well. To further reduce communication, we hash all the commitments together and send only the hash value to $\mathcal{V}$.

Cheating Error (Soundness). We compute the probability that $\mathcal{V}$ outputs acc when $C(\mathbf{w}) \neq \mathbf{y}$. Let c be the number of pre-processing emulations where $\mathcal{P}$ cheats (i.e., by generating incorrect squares or multiplication triples). Since τ emulations out of M are opened and tested by the verifier, we have that this step is passed without the cheating being detected with probability $\frac{\binom{M-c}{\tau}}{\binom{M}{\tau}}$. After this step, $M - \tau$ circuit computations are being simulated by the prover. In order to make the output of the protocol be $\mathbf{y}$, $\mathcal{P}$ must cheat (i.e., deviate from the specification of the MPC protocol) in $M - \tau - c$ emulations. Since $N - 1$ views are being opened in each such emulation, $\mathcal{P}$ clearly will not sabotage the view of more than one party. Thus, the probability that this is not detected is $\frac{1}{N^{M-\tau-c}}$. The overall success cheating probability is therefore

$$\xi_{\mathsf{c\&c}}(M, N, \tau) = \max_{0 \leq c \leq M-\tau} \left\{ \frac{\binom{M-c}{\tau}}{\binom{M}{\tau} \cdot N^{M-\tau-c}} \right\}$$

Formal Proof. As mentioned before, the above protocol has appeared already in [KKW18] (for Boolean circuits, but extending it to Arithmetic circuits is straightforward). However, there it was described as an optimization to their baseline protocol and so was not formally proven. In the full version we therefore provide a proof that the described protocol $\Pi_{\mathsf{c\&c}}$ is an honest verifier zero-knowledge argument of knowledge.

Theorem 2. *Let H be a collision-resistant hash function and let* $\mathtt{com}$ *be the Random Oracle-based commitment scheme. Then, the protocol $\Pi_{\mathsf{c\&c}}$ is an HVZKAoK with knowledge error (soundness) $\xi_{\mathsf{c\&c}}(M, N, \tau)$.*

3.3 HVZKAoK Protocol Using Imperfect Preprocessing and Sacrificing

We now present our second HVZKAoK protocol Π_{sac}. In this protocol, we rely on a method where one "sacrifices" random multiplication triples and squares in order to verify the correctness of multiplication and square operations. The idea of this protocol is that $\mathcal{P}$ does not simulate the execution of a protocol to *compute* multiplication and square gates, but rather simulates an execution of a protocol to *verify* that the shares on the output wires of these gates are correctly defined. This means that now $\mathcal{P}$ will first define and commit to sharings of the values on each wire of the circuit and then will simulate an execution of a verification protocol for multiplication and square gates (recall that for other gates the computation is local and thus no verification is required). We begin by describing the verification methods used in our protocol.

Verification of a Multiplication Triple Using Another. This procedure is reminiscent to the recent work of [LN17]. Given a random triple $([\![a]\!], [\![b]\!], [\![c]\!])$, it is possible to verify the correctness of a triple $([\![x]\!], [\![y]\!], [\![z]\!])$, i.e., that $z = x \cdot y$, without revealing any information on either of the triples, in the following way:

1. The parties generate a random $\epsilon \in \mathbb{F}$.
2. The parties locally set $[\![\alpha]\!] = \epsilon[\![x]\!] + [\![a]\!]$, $[\![\beta]\!] = [\![y]\!] + [\![b]\!]$.
3. The parties run $\mathsf{open}([\![\alpha]\!])$ and $\mathsf{open}([\![\beta]\!])$ to obtain α and β.
4. The parties locally set $[\![v]\!] = \epsilon[\![z]\!] - [\![c]\!] + \alpha \cdot [\![b]\!] + \beta \cdot [\![a]\!] - \alpha \cdot \beta$.
5. The parties run $\mathsf{open}([\![v]\!])$ to obtain v and accept iff $v = 0$.

Observe that if both triples are correct multiplication triples (i.e., $z = xy$ and $c = ab$) then the parties will always accept since

$$\begin{aligned} v &= \epsilon \cdot z - c + \alpha \cdot b + \beta \cdot a - \alpha \cdot \beta \\ &= \epsilon \cdot xy - ab + (\epsilon \cdot x + a)b + (y + b)a - (\epsilon \cdot x + a)(y + b) = 0 \end{aligned}$$

In contrast, if one (or both) of the triples are incorrect, then the parties will accept with probability at most $1/|\mathbb{F}|$ as shown in Lemma 1 whose proof appears in the full version.

Lemma 1. *If $([\![a]\!], [\![b]\!], [\![c]\!])$ or $([\![x]\!], [\![y]\!], [\![z]\!])$ is an incorrect multiplication triple then the parties output acc in the sub-protocol above with probability $\frac{1}{|\mathbb{F}|}$.*

Verification of a Square Pair Using Another. Similarly, one can use a random square $([\![b]\!], [\![d]\!])$ to verify the correctness of a given square $([\![x]\!], [\![z]\!])$ as follows:

1. The parties generate a random $\epsilon \in \mathbb{F}$.
2. The parties locally compute $[\![\alpha]\!] = [\![x]\!] - \epsilon[\![b]\!]$.
3. The parties run $\mathsf{open}([\![\alpha]\!])$ to obtain α.
4. Each party locally computes $[\![v]\!] = [\![z]\!] - \alpha \cdot ([\![x]\!] + \epsilon[\![b]\!]) - \epsilon^2[\![d]\!]$.
5. The parties run $\mathsf{open}([\![v]\!])$ to obtain v and accept iff $v = 0$.

As before, if the squares are correct, i.e., $z = x^2$ and $d = b^2$, then the parties will accept, since

$$\begin{aligned} v &= z - \alpha \cdot (x + \epsilon \cdot b) - \epsilon^2 \cdot d \\ &= x^2 - (x - \epsilon \cdot b) \cdot (x + \epsilon \cdot b) - \epsilon^2 \cdot b^2 \\ &= x^2 - x^2 + \epsilon^2 b^2 - \epsilon^2 b^2 = 0 \end{aligned}$$

In contrast, if one of the random squares (or both) is incorrect, then the parties will accept with probability $\frac{2}{|\mathbb{F}|}$. This is shown in Lemma 2, which is proven in the full version.

Lemma 2. *If* $([\![x]\!],[\![z]\!])$ *or* $([\![b]\!],[\![d]\!])$ *is an incorrect square, then the parties output* acc *in the above protocol with probability* $\frac{2}{|\mathbb{F}|}$.

The Protocol. Our AoK protocol is formally described in Figs. 1a and 1b. In this protocol, the prover $\mathcal{P}$ first commits in Round 1 to sharings of the values on each wire of the circuit and to sharings of random multiplication triples and squares for M independent executions. As in the previous protocol, we save communication by deriving all the random shares from a single seed. Then, in Round 2, $\mathcal{V}$ challenges $\mathcal{P}$ by choosing the randomness required for the verification procedure, i.e., an ϵ value for each multiplication and square gate. Upon receiving the challenge from $\mathcal{V}$, $\mathcal{P}$ simulates M executions of the verification protocol in Round 3 and commits to the view of the parties in each execution. Then, in Round 4, $\mathcal{V}$ picks its second challenge by choosing, for each execution, $N-1$ parties whose view will be opened and tested. In Round 5, $\mathcal{P}$ sends to $\mathcal{V}$ the seeds from which the randomness of the $N-1$ parties was derived and all the messages sent to these parties from the remaining party $P_{\bar{i}_e}$. As in $\Pi_{\mathsf{c\&c}}$, for values that are fixed, i.e., inputs, multiplications and squares, $\mathcal{P}$ sends also an offset (which was committed in the first round) to "fix" the sharing to the correct value. As before, we further reduce the communication cost by hashing the commitments together and sending only the hash value. Finally, $\mathcal{V}$ accepts if and only if all commitments are correct, the view of each party was computed correctly, the verification procedures conclude with the parties holding a sharing of 0 for each multiplication/square gate and the output of the circuit is $\mathbf{y}$.

Cheating Probability (Soundness). We compute the probability that $\mathcal{V}$ outputs acc when $C(\mathbf{w}) \neq \mathbf{y}$. Observe that all the M executions are independent of each other. When considering a single instance, $\mathcal{P}$ can cheat in either computing the view of one of the parties or cheat by changing the shares on the output wire of a multiplication/square gate. In the former case, it will succeed with probability $\frac{1}{N}$ whereas in the latter case it will succeed with probability $\frac{1}{|\mathbb{F}|}$ or $\frac{2}{|\mathbb{F}|}$ (note that if there are gates of both types in the circuit, it will be more beneficial for $\mathcal{P}$ to cheat in square gates since $\frac{2}{|\mathbb{F}|} > \frac{1}{|\mathbb{F}|}$). Furthermore, the best strategy for the prover is to first cheat in multiplication/square gates and then if it didn't receive the desired challenge that will cause the verification process to end successfully, it can manipulate one of the parties' view. Thus, if there are square gates in the circuit, then the overall cheating probability is

$$\xi_{\mathsf{sac}}(M,N) = \left(\frac{2}{|\mathbb{F}|} + \left(1 - \frac{2}{|\mathbb{F}|}\right)\cdot\frac{1}{N}\right)^M = \left(\frac{2N+|\mathbb{F}|-2}{|\mathbb{F}|\cdot N}\right)^M.$$

Similarly, if there are multiplication gates in the circuit (and no square gates), then the cheating probability is

$$\xi_{\mathsf{sac}}(M,N)=\left(\frac{1}{|\mathbb{F}|}+\left(1-\frac{1}{|\mathbb{F}|}\right)\cdot\frac{1}{N}\right)^{M}=\left(\frac{N+|\mathbb{F}|-1}{|\mathbb{F}|\cdot N}\right)^{M}.$$

It can be seen that the impact of $|\mathbb{F}|$ on the cheating probability in practice is not important, as the $1/N$-term will dominate the expression since $N \ll |\mathbb{F}|$.

In the full version we will give the full proof of the following.

Theorem 3. *Let H be a collision-resistant hash function and let* $\mathtt{com}$ *be the Random Oracle-based commitment scheme. Then the protocol Π_{sac} is a HVZKAoK with knowledge error (soundness) $\xi_{\mathsf{sac}}(M,N)$.*

3.4 Optimizations

The following optimizations can directly be made to our protocols:

1. The prover is required to send $N-1$ seeds for each execution e that was not chosen to be opened. Each of these seeds is used to generate the randomness of one party throughout the execution. As in [KKW18], we can reduce the number of seeds that are sent from $N-1$ to $\log N$ by using a binary tree.
2. We can reduce communication by verifying the correctness of the circuit's output in a batched manner, i.e., take a random linear combination of all outputs, where the randomness is chosen (as an additional challenge) by $\mathcal{V}$. Then, only the shares of this linear combination result are sent to $\mathcal{V}$.
3. Each multiplication in Π_{sac} is being verified separately. In order to save communication it is possible to batch-verify of them by opening a random linear combination of all $[\![v]\!]$-sharings.

3.5 Computation and Communication Cost

By inspecting both Π_{sac} and $\Pi_{\mathsf{c\&c}}$ one sees that for each multiplication gate $O(M\cdot N)$ multiplications in $\mathbb{F}$ must be computed. In practice, their runtime dominates those of the additions in $\mathbb{F}$ which can be optimized by carrying out multiple $\mathbb{F}$-additions over the integers before applying a modular reduction. For large enough $\mathbb{F}$ we have that $\xi_{\mathsf{sac}}(M,N)\approx(1/N)^{M}$, and so for statistical security parameter κ we have $M\cdot\log N=\kappa$ which means that we will approximately have to perform $O(\kappa\cdot(N/\log N)\cdot|C|)$ multiplications both at proving and verification time, but only over the field over which C is actually defined.

Next, we estimate both the practical and asymptotic communication cost of the Π_{sac} protocol[3]. Denote by $|\mathsf{hash}|$, $|\mathsf{sd}|$ and $|\mathtt{com}|$ the length in bits of the hash values, seeds and commitments. The communication cost of messages sent from $\mathcal{P}$ to $\mathcal{V}$ in each round is: (i) **Round 1**: $|\mathsf{hash}|$; (ii) **Round 3**: $|\mathsf{hash}|$; and

[3] The analysis for $\Pi_{\mathsf{c\&c}}$ is described in the full version.

Let H be a CRHF and $\texttt{com}$ be the Random Oracle-based commitment scheme.

Inputs: Both $\mathcal{P}$ and $\mathcal{V}$ hold $\mathbf{y} \in \mathbb{F}^{n_{\text{out}}}$, a description C over $\mathbb{F}$ and parameters M, N; $\mathcal{P}$ additionally holds $\mathbf{w} \in \mathbb{F}^{n_{\text{in}}}$ such that $C(\mathbf{w}) = \mathbf{y}$.

Round 1:

1. $\mathcal{P}$ chooses a salt $\texttt{salt} \leftarrow \mathbb{B}^{\lambda}$ and does the following for each $e \in [M]$:
   - (a) Initialize empty strings $\mathsf{st}_e, \{\mathsf{st}_{e,i}\}_{i\in[N]}$.
   - (b) Choose seeds $\mathsf{sd}_e, \{\mathsf{sd}_{e,i}\}_{i\in[N]}$ and set $\mathsf{st}_{e,i} \leftarrow \mathsf{sd}_{e,i}$ for $i \in [N]$.
   - (c) Prepare the pre-processing data:
     - – For each multiplication gate $g_k \in \mathcal{G}$:
       - i. For each $i \in [N]$, use $\mathsf{sd}_{e,i}$ to generate $a_{e,k,i}, b_{e,k,i}, c_{e,k,i}$. These shares define the random sharings $[\![a_{e,k}]\!], [\![b_{e,k}]\!]$ and $[\![c_{e,k}]\!]$, where $a_{e,k} = \sum_{i=1}^{N} a_{e,k,i}$, $b_{e,k} = \sum_{i=1}^{N} b_{e,k,i}$ and $c_{e,k} = \sum_{i=1}^{N} c_{e,k,i}$.
       - ii. Set $\Delta_{e,k} = a_{e,k} \cdot b_{e,k} - c_{e,k}$ and $\mathsf{st}_e \leftarrow \mathsf{st}_e \parallel \Delta_{e,k}$.
       - iii. Define the random triple for g_k to be $([\![a_{e,k}]\!], [\![b_{e,k}]\!], [\![c_{e,k}]\!] + \Delta_{e,k})$.
     - – For each square gate $g_k \in \mathcal{G}$:
       - i. For each $i \in [N]$ use $\mathsf{sd}_{e,i}$ to generate $b_{e,k,i}$ and $d_{e,k,i}$. These shares define the random sharings $[\![b_{e,k}]\!]$ and $[\![d_{e,k}]\!]$, where $b_{e,k} = \sum_{i=1}^{N} b_{e,k,i}$ and $d_{e,k} = \sum_{i=1}^{N} d_{e,k,i}$.
       - ii. Set $\Delta_{e,k} = (b_{e,k})^2 - d_{e,k}$ and $\mathsf{st}_e \leftarrow \mathsf{st}_e \parallel \Delta_{e,k}$.
       - iii. Define the random square for g_k to be $([\![b_{e,k}]\!], [\![d_{e,k}]\!] + \Delta_{e,k})$.
   - (d) Choose a random sharing of the inputs:
     - i. For each $i \in [N]$, use $\mathsf{sd}_{e,i}$ to generate $w_{e,1,i}, \ldots, w_{e,n_{\text{in}},i}$. These shares define the random sharings $[\![w_{e,1}]\!], \ldots, [\![w_{e,n_{\text{in}}}]\!]$, where $w_{e,k} = \sum_{i=1}^{N} w_{e,k,i}$.
     - ii. For each input wire $k \in \mathcal{I}$ set $\phi_{e,k} = w_k - \sum_{i=1}^{N-1} w_{e,k,i}$ and $\mathsf{st}_e \leftarrow \mathsf{st}_e \parallel \phi_{e,k}$. The sharing on this wire then is $[\![w_{e,k}]\!] + \phi_{e,k}$.
   - (e) Simulate the computation of C gate-by-gate in topological order:
     - – For each linear gate, compute the parties' output shares via the local operation described in Section 3.1.
     - – For each multiplication gate $g_k \in \mathcal{G}$ with $[\![x_k]\!], [\![y_k]\!]$ as inputs:
       - i. For each $i \in [N]$, use $\mathsf{sd}_{e,i}$ to generate $z_{e,k,i}$ which define the random sharing $[\![z_{e,k}]\!]$ where $z_{e,k} = \sum_{i=1}^{N} z_{e,k,i}$.
       - ii. Set: $\varphi_{e,k} = x_k \cdot y_k - \sum_{i=1}^{N} z_{e,k,i}$ and $\mathsf{st}_e \leftarrow \mathsf{st}_e \parallel \varphi_{e,k}$. The sharing on the output wire is defined to be $[\![z_{e,k}]\!] + \varphi_{e,k}$.
     - – For each square gate $g_k \in \mathcal{G}$ with sharing $[\![x_k]\!]$ on its input wire:
       - i. For each $i \in [N]$ use $\mathsf{sd}_{e,i}$ to generate $z_{e,k,i}$. These shares define the random sharing $[\![z_{e,k}]\!]$ where $z_{e,k} = \sum_{i=1}^{N} z_{e,k,i}$.
       - ii. Set: $\varphi_{e,k} = (x_k)^2 - \sum_{i=1}^{N} z_{e,k,i}$ and $\mathsf{st}_e \leftarrow \mathsf{st}_e \parallel \varphi_{e,k}$. The sharing on the output wire is defined to be $[\![z_{e,k}]\!] + \varphi_{e,k}$.
   - (f) Use sd_e to generate $r_e \in \mathbb{B}^{\lambda}$ and compute $\Gamma_e = \texttt{com}(\mathsf{st}_e, r_e, \texttt{salt})$.
   - (g) For each $i \in [N]$ use $\mathsf{sd}_{e,i}$ to generate $r_{e,i} \in \mathbb{B}^{\lambda}$ and then compute $\Gamma_{e,i} = \texttt{com}(\mathsf{st}_{e,i}, r_{e,i}, \texttt{salt})$. Then set $h_e = H(\Gamma_e \parallel \Gamma_{e,1} \parallel \cdots \parallel \Gamma_{e,N})$.
2. Compute $h_\Gamma = H(h_1 \parallel \cdots \parallel h_M)$ and send it to $\mathcal{V}$.

Fig. 1a. The "Sacrificing" based argument Π_{sac} (Part 1)

Round 2: $\mathcal{V}$ chooses sd_ι and for each $e \in [M]$ uses sd_ι to generate random coefficients $\epsilon_{e,k}$ for each multiplication/square gate g_k. $\mathcal{V}$ then sends sd_ι to $\mathcal{P}$.

Round 3: $\mathcal{P}$ performs the following steps:

1. Choose a random seed sd_E. Use sd_ι to generate random $\epsilon_{e,k}$ as $\mathcal{V}$ would do.
2. For each $e \in [M]$:
   - (a) Initialize an empty string view_e.
   - (b) For each multiplication gate g_k (in topological order) simulate the verification procedure described in the text using $\epsilon_{e,k}$. In addition, set: $\mathsf{view}_e \leftarrow \mathsf{view}_e \parallel \alpha_{e,k,1} \parallel \cdots \parallel \alpha_{e,k,N} \parallel \beta_{e,k,1} \parallel \cdots \parallel \beta_{e,k,N}$.
   - (c) For each square gate g_k (in topological order) simulate the verification procedure described in the text using $\epsilon_{e,k}$. In addition, sets $\mathsf{view}_e \leftarrow \mathsf{view}_e \parallel \alpha_{e,k,1} \parallel \cdots \parallel \alpha_{e,k,N}$.
   - (d) Let $v_{e,k,i}$ be the sharing held by party P_i at the end of the verification procedure of gate g_k. Then, for each $i \in [N]$ set: $\mathsf{view}_e \leftarrow \mathsf{view}_e \parallel v_{e,k,1} \parallel \cdots \parallel v_{e,k,N}$.
   - (e) Let $o_{e,1,i}, \ldots, o_{e,n_{\mathtt{out}},i}$ be the shares on the output wires of C held by P_i. Then, for output wire $k \in \mathcal{O}$ set: $\mathsf{view}_e \leftarrow \mathsf{view}_e \parallel o_{e,k,1} \parallel \cdots \parallel o_{e,k,N}$.
3. Generate $g_e \in \{0,1\}^\lambda$ from sd_E and set $\Pi_e = \mathtt{com}(\mathsf{view}_e, g_e, \mathtt{salt})$.
4. Compute $h_\pi = H(\Pi_1 \parallel \cdots \parallel \Pi_M)$ and send it to $\mathcal{V}$.

Round 4: For each $e \in [M]$: $\mathcal{V}$ sends a random $\bar{i}_e \in [N]$ to $\mathcal{P}$.

Round 5: For each $e \in [M]$:
Let $I_e = [N] \setminus \{\bar{i}_e\}$. Then, $\mathcal{P}$ sends the following to $\mathcal{V}$: $\mathtt{salt}$, sd_E, sd_e, $\{\mathsf{sd}_{e,i}\}_{i \in I_e}$, $\Gamma_{e,\bar{i}_e}$, $\{\phi_{e,k}\}_{k=1}^{n_{\mathtt{in}}}$, the tuple $\left(\Delta_{e,k}, \varphi_{e,k}, \alpha_{e,k,\bar{i}_e}, \beta_{e,k,\bar{i}_e}, v_{e,k,\bar{i}_e}\right)$ for each multiplication or square gate g_k, and $o_{e,1,\bar{i}_e}, \ldots, o_{e,n_{\mathtt{out}},\bar{i}_e}$.

Output: $\mathcal{V}$ outputs acc iff all the following checks succeed:

1. For each $e \in [M]$, $\mathcal{V}$ uses $\{\mathsf{sd}_{e,i}\}_{i \in I_e}$ and the tuple received for each multiplication and square gate to compute the shares of the parties in I_e on each wire and their shares of each random triple and square. Then, it uses sd_e to compute Γ_e and uses $\{\mathsf{sd}_{e,i}\}_{i \in I_e}$ to compute $\{\Gamma_{e,i}\}_{i \in I_e}$ as an honest prover would do. Then, using $\Gamma_{e,\bar{i}_e}$ received from $\mathcal{P}$, the verifier $\mathcal{V}$ computes h_e.
   Then, $\mathcal{V}$ checks that $h_\Gamma = H(h_1 \parallel \cdots \parallel h_M)$.
2. For each $e \in [M]$, $\mathcal{V}$ computes view_e by going gate-by-gate in topological order and simulating the verification procedure using the tuple received from $\mathcal{P}$ for each multiplication and square gate, and using $\{o_{e,k,\bar{i}_e}\}_{k=1}^{n_{\mathtt{out}}}$. Then, it computes Π_e as a honest prover would do. Finally, $\mathcal{V}$ checks that $h_\pi = H(\Pi_1 \parallel \cdots \parallel \Pi_M)$.
3. For each $e \in [M]$ and multiplication/square gate g_k, $\mathcal{V}$ checks if $\sum_{i=1}^N v_{e,k,i} = 0$.
4. For each $e \in [M]$, for each $k \in \mathcal{O}$, $\mathcal{V}$ checks that $\sum_{i=1}^N o_{e,k,i} = y_k$.

Fig. 1b. The "Sacrificing" based argument Π_{sac} (Part 2)

(iii) **Round 5**: $|\mathsf{sd}| + M \cdot (|\mathsf{sd}| + \log N \cdot |\mathsf{sd}| + |\mathtt{com}| + 4\log_2(|\mathbb{F}|) \cdot n_{mul} + 3\log_2(|\mathbb{F}|) \cdot n_{sq} + \log_2(|\mathbb{F}|) + \log_2(|\mathbb{F}|) \cdot n_{\mathtt{in}} + \log_2(|\mathbb{F}|)$.

Let $\mathtt{base}(\mathsf{hash}, \mathsf{sd}, \mathtt{com}, M, N) = 2 \cdot |\mathsf{hash}| + |\mathsf{sd}| \cdot (2 + M \log N) + |\mathtt{com}| \cdot M$ for which we only write $\mathtt{base}$ when the context is clear. We obtain that the overall amount of bits sent from $\mathcal{P}$'s side is $\mathtt{base} + \log_2(|\mathbb{F}|) \cdot M(4n_{mul} + 3n_{sq} + n_{\mathtt{in}} + 2)$.

Asymptotically, by setting $|\mathsf{hash}| = |\mathsf{sd}| = |\mathtt{com}| = O(\lambda)$, $\log_2(|\mathbb{F}|) = O(\log(\lambda))$ and M, N as above we get that the communication cost of $\mathcal{P}$ is $O(\log(\lambda) \cdot \kappa \cdot (|C|/\log(N)))$.

4 Sampling Circuits on the Fly

At the end of the previous section we briefly mentioned an optimization where $\mathcal{V}$ checks output correctness by looking only at a linear combination of the outputs instead of checking each output separately. In particular, this is done by having $\mathcal{V}$ choosing random coefficients which will be used to compute the linear combination *after* $\mathcal{P}$ fixes the inputs and (correlated) randomness of the simulated parties. This process can also be viewed as an interaction where the parties determine the final circuit's structure during the execution, as here the challenge chosen by $\mathcal{V}$ adds a layer on top of the initial circuit which consists of 'multiplication-by-a-constant' and addition gates. This idea, which we call "sampling the circuit on the fly" will be also used in some of the optimizations suggested for the application presented in Sect. 5. We therefore now formally establish this idea, so that security of optimizations of this kind can be derived easily without the need to re-prove security of the whole ZKAoK each time.

Although in the above example only $\mathcal{V}$ chooses the circuit that will be evaluated, we consider a broader definition where both $\mathcal{P}, \mathcal{V}$ sample the circuit together. The sampling process must begin only after $\mathcal{P}$ has committed and fixed the witness and randomness that will be used. This means that from this point on any form of cheating is possible only during the simulation of the MPC protocol to compute the sampled circuit, as the witness cannot be tailored anymore to the actually chosen circuit. We remark that although the circuit will be jointly sampled by both parties, we restrict the sampling done by $\mathcal{V}$ to be independent of the messages of $\mathcal{P}$ and to not require him to keep a secret state so that the overall protocol stays public-coin. $\mathcal{P}$, in contrast, will be allowed to make his choice depending on the witness that it committed or on other messages. At the same time, the choice of $\mathcal{P}$ should neither allow him to break the soundness nor the zero-knowledge property.

In this section, we first provide a formal definition for the notion of circuit sampling. Then, we show how to incorporate it into our argument system and finally explain (as an example) how the output linear combination optimization described above is an instantiation of the general notion and how it fits into the framework. We want to mention that, independently, Badetscher et al. [BJM19] introduced a similar concept but in an unrelated context.

4.1 Definition of Circuit Sampling

First, we define the notion of circuit sampling for an NP relation.

Definition 3. (R-circuit Sampler). *Let R be an NP relation and $S_{\mathcal{P}}, S_{\mathcal{V}}$ be two non-empty sets that can be described with a string of polynomial length (in the security parameter λ). We say that* $\mathsf{Sample} = (\mathsf{ExtWitness}, \mathsf{Response}, \mathsf{SampCircuit})$ *is an R-circuit sampler for $(x, w) \in R$ if*

$\mathsf{ExtWitness}$ *is a PPT algorithm which on input* (x, w) *outputs an extended witness* $\widehat{w}$.
$\mathsf{Response}$ *is a PPT algorithm which on input* $(x, w, \widehat{w}, \tau_{\mathcal{V}})$ *outputs* $\tau_{\mathcal{P}}$.
$\mathsf{SampCircuit}$ *is a deterministic polynomial-time algorithm which on input* $(x, \tau_{\mathcal{V}}, \tau_{\mathcal{P}})$ *outputs a circuit* C *as well as a description of a set* Y.

Furthermore, we require that membership in Y can be decided in polynomial time. We next define a security game which follows the way we embed these algorithms into our protocols. Consider the following game, which we denote by $\mathsf{Game}_{R,\mathcal{P}}\left((x, w), S_{\mathcal{P}}, S_{\mathcal{V}}, \lambda\right)$, executed with $\mathcal{P}$:

1. $\mathcal{P}$ outputs $\widehat{w}$.
2. Choose a random $\tau_{\mathcal{V}} \leftarrow S_{\mathcal{V}}$ and hand it to $\mathcal{P}$.
3. $\mathcal{P}$ outputs $\tau_{\mathcal{P}} \in S_{\mathcal{P}}$.
4. Compute $(C, Y) \leftarrow \mathsf{SampCircuit}(x, \tau_{\mathcal{P}}, \tau_{\mathcal{V}})$.
5. Output 1 iff $C(\widehat{w}) \in Y$.

To understand the game, observe that Step 1 emulates the commitment to the witness, made by $\mathcal{P}$ in the first step of our protocols, in Step 2 a challenge is chosen which is followed by the configuration chosen by $\mathcal{P}$ in Step 3. Once all the input for SampCircuit is gathered, (C, Y) are being determined, and $\mathcal{P}$ wins if computing the circuit C on $\widehat{w}$ yields a valid output. In the above definition there is no validation ensuring that $\tau_{\mathcal{P}}$ used in the game is valid. This can be done by SampCircuit outputting $Y = \emptyset$ for an *invalid* $\tau_{\mathcal{P}}$.

We have three requirements from the circuit sampler. First, an obvious requirement is that if $\mathcal{P}$ uses the correct w and chooses $\tau_{\mathcal{P}}$ honestly, then the output of the game should be 1 (except for a negligible probability).

Definition 4. (Correct R-circuit Sampler). *Let* Sample *be an* R*-circuit sampler. If when* $\mathcal{P}$ *on input* $(x, w) \in R$ *computes* $\widehat{w} \leftarrow \mathsf{ExtWitness}(x, w)$ *and* $\tau_{\mathcal{P}} \leftarrow \mathsf{Response}(x, w, \widehat{w}, \tau_{\mathcal{V}})$, *with probability negligibly close to 1 it holds that* $\mathsf{Game}_{R,\mathcal{P}}\left((x, w), S_{\mathcal{P}}, S_{\mathcal{V}}, \lambda\right) = 1$ *then we say that* Sample *is correct.*

We furthermore require soundness. Similarly to the standard definition of it, here if $\mathcal{P}$ wins in the above game with probability $> \alpha$, then a correct witness for R can be extracted.

Definition 5. (α-sound R-circuit Sampler). *Let* Sample *be an* R*-circuit sampler. If given* $\Pr[\mathsf{Game}_{R,\mathcal{P}}\left((x, w), S_{\mathcal{P}}, S_{\mathcal{V}}, \lambda\right) = 1] > \alpha$ *(where the distribution is over* $\tau_{\mathcal{V}} \in S_{\mathcal{V}}$*), there exists a deterministic polynomial-time extractor* $\mathcal{E}(\widehat{w})$ *which outputs* $(x, w') \in R$ *then we say that* Sample *is* α*-sound.*

The definition may look similar to knowledge soundness as defined in Sect. 2.2, but there are crucial differences: $\mathcal{E}$ runs on $\widehat{w}$ in polynomial time and with probability 1. This is because extracting some w' from $\widehat{w}$ is an "easy" task (as we will see in all our circuit sampling uses) and so the only question is whether w' is valid for R or not. The definition thus says that if $\mathcal{P}$ wins with probability higher than α, then it must have used the correct witness w to compute $\widehat{w}$ which can be obtained.

Finally, we also need to ensure that the additional interaction does not leak any information about w. This is formalized in the standard way of requiring the existence of a simulator who can output an indistinguishable transcript without knowing w. Clearly, the message $\tau_{\mathcal{P}}$ should not reveal any information about $\widehat{w}$ to an outsider. However, we additionally need simulatability of $C(\widehat{w})$: the sampled circuit may enforce the relation R in different ways than a static circuit would do, which could potentially leak information.

Definition 6. (Simulatable R-circuit Sampler). *Let $(x, w) \in R$ and* Sample *be an R-circuit sampler. Then we say that* Sample *is simulatable if there exists a PPT algorithm $\mathcal{S}$ such that*

$$\{(\tau_{\mathcal{P}}, C(\widehat{w})) \leftarrow \mathcal{P}(x, w, \tau_{\mathcal{V}})\} \approx_s \{(\tau_{\mathcal{P}}, C(\widehat{w})) \leftarrow \mathcal{S}(x, \tau_{\mathcal{V}})\}$$

where $\mathcal{P}$ acts honestly as in Definition 4.

4.2 Circuit Sampling and Our ZKAoK

We now include the above approach into our second protocol Π_{sac}. The modified protocol $\Pi_{\mathsf{sac}}^{\mathrm{samp}}$ works as follows, where we highlight the additional steps:

Round 1: For each $e \in [M]$ (i.e. each MPC instance), $\mathcal{P}$ computes $\widehat{w}_e \leftarrow$ $\mathsf{ExtWitness}(x, w)$. Then, it chooses the randomness used for the execution e (i.e., the seeds used to derive all randomness as well as the salt). Finally, $\mathcal{P}$ commits to the extended witness and the randomness and sends it to $\mathcal{V}$.

Round 2: For each $e \in [M]$, $\mathcal{V}$ samples $\tau_{\mathcal{V},e}$ as in Step 2 of the above game. It then sends $\tau_{\mathcal{V},1}, \ldots, \tau_{\mathcal{V},M}$ to $\mathcal{P}$.

Round 3: $\mathcal{P}$ locally computes $\tau_{\mathcal{P},e} \leftarrow \mathsf{Response}(x, w, \widehat{w}_e, \tau_{\mathcal{V},e})$ for each $e \in [M]$ as well as $(C_e, Y_e) \leftarrow \mathsf{SampCircuit}(x, \tau_{\mathcal{P},e}, \tau_{\mathcal{V},e})$. It uses C_e in MPC protocol instance e and sends the remaining first round messages together with $\tau_{\mathcal{P},e}$ to $\mathcal{V}$.

Round 4–Round 7: Run rounds 2–5 as in the regular protocol.

Output: Upon receiving the last message, for each $e \in [M]$ $\mathcal{V}$ recomputes $(C_e, Y_e) \leftarrow \mathsf{SampCircuit}(x, \tau_{\mathcal{P},e}, \tau_{\mathcal{V},e})$, verifies the MPC transcripts for the individual C_e and then tests that each output lies in Y_e.

The following Lemma, whose proof appears in the full version, shows that $\Pi_{\mathsf{sac}}^{\mathrm{samp}}$ is an HVZKAoK.

Lemma 3. *Let* Sample *be a correct, α-sound and simulatable R-circuit sampler for the NP-relation R. Then Π_{sac}^{samp} is a statistically complete HVZKAoK for R with knowledge error $(\alpha + (1 - \alpha) \cdot \xi_{sac}(1, N))^M$.*

To prove the lemma, the main change compared to the proof of Π_{sac} is that here the circuits are not identical throughout all instances. Fortunately, it turns out that this assumption can be relaxed without hurting the runtime of the extractor $\mathcal{E}$. Completeness and the zero-knowledge property, on the other hand, follow directly from the original proof.

Impact on the Argument Size and Runtime. As one can see from the above description, adding Circuit Sampling to the protocol Π_{sac} adds another two rounds of communication. In terms of argument size, we essentially split **Round 1** into two different parts and make two commitments instead of one which commit to preprocessed data and evaluation, but now separately. The extra cost is to send two extra commitments (thus $\mathtt{base}$ increases by $2 \cdot |\mathtt{com}|$), which is negligible in comparison to the rest of the argument.

Furthermore, it is possible to cut away the extra two rounds of communication by running the simulation C in **Round 3** only, at the expense of introducing more communication. This can be done by switching from the verification-based approach of Π_{sac} to the standard forward circuit evaluation of π where we check the triples/squares while we use them, which is possible because now evaluation is fully deterministic. This allows to perform evaluation and checking in one round in parallel. We leave a detailed analysis as future work.

4.3 Output Correctness as a Circuit Sampler

We now revisit the (briefly sketched) idea of output compression in the context of circuit sampling. Here $\mathcal{V}$ chooses random coefficients that are used to compute the linear combination of the outputs, so that only one value is eventually opened by the resulting circuit instead of $n_{\mathtt{out}}$.

We first define the three algorithms of the circuit sampler for this optimization: $\mathsf{ExtWitness}$ receives $((C, \mathbf{y}), \mathbf{w})$ as an input and returns the extended witness $\widehat{w}$, which in this case is just $\mathbf{w}$. $\mathsf{Response}$ receives as an input the tuple $((C, \mathbf{y}), \mathbf{w}, \widehat{w}, \tau_{\mathcal{V}})$, but note that in this optimization, the verifier's challenge $\tau_{\mathcal{V}}$ fully defines the circuit and thus the output of $\mathsf{Response}$ is just 1. Finally, $\mathsf{SampCircuit}$ receives $((C, \mathbf{y}), \tau_{\mathcal{V}}, \tau_{\mathcal{P}})$ as its input and returns the circuit C' and the set Y defined in the following way: The circuit C' consists of the original circuit C and the following layers which are added on top of it: (i) subtraction gates for subtracting each value on an output wire $\mathbf{y}'[k]$ by the expected public value $\mathbf{y}[k]$; (ii) 'multiplication-by-a-constant' gates for each result of the previous layer, where the constants are defined by $\tau_{\mathcal{V}}$; and (iii) addition gates for summing the results of the previous layer. The set Y consists of one value only. We summarize the construction in Fig. 2.

The three algorithms defined above satisfy the properties of the Circuit Sampler. Correctness is straightforward. Soundness of the sampler is $\frac{1}{|\mathbb{F}|}$, since if w is incorrect, then $C(w) \in Y$ with probability $\frac{1}{|\mathbb{F}|}$ because the random coefficients are uniform (see Lemma 1). Simulation follows since both $\tau_{\mathcal{P}}$ and Y are fixed.

Let $C = (n_{\mathtt{in}}, n_{\mathtt{out}}, n_C, L, R, F)$ be a circuit over $\mathbb{F}$.

ExtWitness: On input $(x = (C, \mathbf{y}), \mathbf{w})) \in R$ set $\widehat{w} := \mathbf{w}$.

SampCircuit: On input $\tau_{\mathcal{V}} = (\gamma) \in \mathbb{F}^{n_{\mathtt{out}}}$ output the circuit C' doing the following:
1. Compute $\mathbf{y}' = C(\widehat{w})$ where $\mathbf{y}' \in \mathbb{F}^{n_{\mathtt{out}}}$ and $y_1 = \sum_{i=1}^{n_{\mathtt{out}}} \gamma[i] \cdot (\mathbf{y}'[i] - \mathbf{y}[i])$.
2. Output y_1.

Furthermore output the set $Y = \{(0)\}$.

Response: Output 1.

Fig. 2. Batching the output check as a circuit sampler.

5 Proving Knowledge of SIS Instances

The protocols from Sect. 3 are *asymptotically* less communication-efficient than previous argument systems such as [AHIV17, BBC+18] as can be seen in the analysis. However, they have advantages when the circuit size is not too big or when there are many linear gates in the circuit, because the communication is dominated by the number of non-linear operations in the circuit C and has very small circuit-independent cost. In this section, we exploit this fact to implement communication-efficient arguments of knowledge for different versions of the so-called Short-Integer Solution (SIS) problem.

The section is organized as follows. We begin by presenting an interactive argument for binary secrets which does not allow any slack, which is the same as in [BD10]. The approach can be simply generalized to secrets from a larger interval, but only at the expense of vastly increasing the communication. Then, we introduce some optimizations that allow us to reduce the communication for the suggested arguments and then further squeeze down their size by introducing a slack factor. Throughout the section, for each approach that we present, we will mention what is the resulting size of the argument, based on the analysis of $\Pi_{\mathsf{sac}}^{\mathrm{samp}}$ (which is the same as that of Π_{sac}).

5.1 The Baseline Proof for SIS

We start by presenting an argument for the Binary SIS problem as introduced in Sect. 2.4. The reason behind that is because general range proofs are hard using a circuit over $\mathbb{F} = \mathbb{F}_q$ whereas they are very simple for binary values. Moreover, the protocol we design for this problem will serve as a starting point for constructions supporting secrets from larger intervals.

There are two main tasks that the protocol has to achieve, which is to show that the secret $\mathbf{s}$ is a binary vector and the correctness of the product $\mathbf{t} = \mathbf{A}\mathbf{s}$. The matrix multiplication uses a publicly known matrix, and since linear operations are free in our used MPC scheme computing $\mathbf{t}$ can be done without increasing the proof size. What remains to show is that the witness consists of bits. This test is easy to perform because $\mathbf{s}[i] \in \{0, 1\}$ is equivalent to $\mathbf{s}[i]^2 - \mathbf{s}[i] = 0$. We can therefore let the circuit C compute the square of each element of $\mathbf{s}$ and then perform a linear test. The obtained circuit is described in Fig. 3.

```
Witness: $\mathbf{w} = (\mathbf{s}[1], \dots, \mathbf{s}[m]) \in \mathbb{F}^m$
Computation:
 1. $\forall i \in [m]$ compute $r_i \leftarrow \mathbf{s}[i]^2$
 2. $\forall j \in [n]$ compute $y_j \leftarrow \sum_{i \in [m]} a_{j,i}\mathbf{s}[i]$
 3. $\forall i \in [m]$ compute $y_{i+n} \leftarrow r_i - \mathbf{s}[i]$
Output: $\hat{\mathbf{y}} \leftarrow (y_1, \dots, y_{m+n})$
```

Fig. 3. A circuit representation of $R^{m,n,q}_{\mathtt{B-SIS}}$; The circuit contains m square gates, has m inputs and $m+n$ outputs.

For ease of notation we let $a_{i,j} \in \mathbb{F}$ be the element in the ith row and the jth column of $\mathbf{A}$. The circuit can be evaluated using one of the protocols from Sect. 3, with $\mathcal{V}$ testing that the circuit's output $\hat{\mathbf{y}}$ equals $(\mathbf{t}[1], \dots, \mathbf{t}[n], 0, \cdots, 0)$. This yields a highly efficient protocol, as there are only m non-linear gates in the circuit that require communication, and all of them are square gates. Using the cost analysis from Sect. 3.5, we conclude that the total communication by $\mathcal{P}$ is $\mathtt{base} + \log q \cdot M(4m+2)$ bits. It is immediate to extend the construction from Fig. 3 to full SIS instances (which we do in the full version) by taking the *bit-decomposition* of each input. There we show that, for secrets $\mathbf{s}$ of ∞-norm $\leq \beta$ we will have to expand the witness to contain $(\lfloor \log_2(\beta) \rfloor + 2) \cdot m$ elements and we furthermore have to evaluate as many square gates. $\mathcal{P}$ then sends $\mathtt{base} + \log q \cdot M(4m \cdot (\lfloor \log_2(\beta) \rfloor + 2) + 2)$ bits in the argument.

5.2 Amortizing Bit Tests

We now discuss an optimization which aims at reducing the argument size for the Binary SIS problem by reducing the number of non-linear gates in the circuit. Recall that in Fig. 3, we defined a circuit for this problem that has m square gates. Each of the gates was used to verify that one of m inputs is a bit. We now show how the number of square gates can be reduced to 1, at the cost of adding elements to the witness. This reduces the overall communication since adding an element to the witness increases the size of the argument per MPC instance by one field element, whereas evaluating a square gate requires sending at least two field elements (secret-shared random square, messages during evaluation of the gate etc.). The optimization uses circuit sampling where only $\mathcal{V}$ has a challenge and so only $\mathcal{V}$ is actually sampling the circuit alone.

Assume that we want to check if m input sharings $\mathbf{s}[1], \dots, \mathbf{s}[m]$ indeed are bits, and let $|\mathbb{F}| \gg 2m$. We can implicitly define the polynomial $D(X) \in \mathbb{F}[X]$ of degree at most $m-1$ such that $\forall i \in [m]: \ D(i) = \mathbf{s}[i]$. Furthermore, we know that there exists a polynomial $B(X) = D(X) \cdot D(X)$ of degree at most $2m-2$ such that $\forall i \in \mathbb{F}: \ D(i)^2 = B(i)$. We thus can say that $\forall i \in [m]: \ \mathbf{s}[i] \in \mathbb{B}$ if and only if $\forall i \in [m]: \ B(i) = D(i)$.

This allows us to construct a new circuit-sampling procedure. Instead of testing all $\mathbf{s}[i]$ separately for being bits, we let the prover $\mathcal{P}$ secret-share the predetermined $B(X)$ as part of the witness. Here, by our above observation that $\forall i \in [m] : \; B(i) = D(i)$ it is only necessary to share the points $B(m+1), \ldots, B(2m-1)$ (in addition to sharing all $\mathbf{s}[i]$). Then, using the fact that Lagrange-interpolation requires only linear operations (so it is entirely local in the underlying MPC scheme) we let $\mathcal{V}$ send a challenge $x \in \mathbb{F}$ that is the point at which we will evaluate D, B and test that $B(x) - D(x)^2 = 0$. By the Schwartz-Zippel-Lemma, we then must have identity of $D(X)^2$ and $B(X)$ except with probability $\frac{2m-2}{|\mathbb{F}|}$. In the full version we formalize the above intuition which we show yields a circuit sampler:

Theorem 4. *The aforementioned approach yields a perfectly correct, $\frac{2m-2}{|\mathbb{F}|}$-sound and perfectly simulatable circuit sampler for the relation $R^{m,n,q}_{\mathtt{B-SIS}}$.*

Applying this optimization and using $\Pi^{\mathrm{samp}}_{\mathsf{sac}}$, we obtain that the total communication is $\mathtt{base} + \log q \cdot M(2m+4)$ bits which is approximately $\log q \cdot M(3m)$ bits smaller than the baseline approach.

5.3 Trading Argument Size for Slack

So far we have considered only arguments for SIS-instances where the gap between the norm of correct witnesses and the norm that the argument guarantees is small: if we start with $((\mathbf{A}, \mathbf{y}), \mathbf{s}) \in R^{m,n,q,\beta}_{\mathtt{SIS}}$ (i.e., $||\mathbf{s}||_\infty \leq \beta$) then the soundness guarantee is that a witness $\mathbf{s}'$ with $((\mathbf{A}, \mathbf{y}), \mathbf{s}') \in R^{m,n,q,\omega\beta}_{\mathtt{SIS}}$ could be extracted (i.e., $||\mathbf{s}||_\infty \leq \omega\beta$) where ω is a small constant. However, the argument size depends on $M \cdot m \cdot \log_2(q) \cdot \log_2(\beta)$ as we have to perform non-linear computations for the bit-decomposition of each input $\mathbf{s}[i]$. The goal of this subsection is to give an *approximate* argument of size for the $\mathbf{s}[i]$ without having to resort to bit-decomposition for each $\mathbf{s}[i]$. This would allow for a smaller number of square- or multiplication-gates as well as a more compact witness. On the other hand, the arguments will have a larger slack ω which will now also depend on the number of inputs m.

To achieve a more compact argument, we will ask the prover to show that random linear combinations of elements from $\mathbf{s}$ are small. For this we use a Lemma from [BL17] who showed that random linear combinations mod q of elements from $\mathbf{s}$ are with certain probability not much smaller than $||\mathbf{s}||_\infty$:

Lemma 4. *For all $\mathbf{s} \in \mathbb{F}_q^k$ it holds that*

$$\Pr_{\mathbf{c} \leftarrow \mathbb{B}^k}\left[|\langle \mathbf{c}, \mathbf{s} \rangle| < \frac{||\mathbf{s}||_\infty}{2} \right] \leq \frac{1}{2} \;\; \& \;\; \Pr_{\mathbf{C} \leftarrow \mathbb{B}^{\ell \times k}}\left[||\mathbf{C} \cdot \mathbf{s}||_\infty < \frac{||\mathbf{s}||_\infty}{2} \right] \leq 2^{-\ell}.$$

Proof. See [BL17, Lemma 2.3 & Corollary 2.4].

The above Lemma only talks about the chance of detecting a vector of high norm by seeing *one* large element in the result of the product with a random binary matrix. In the full version we extend it to the case where we always see that lots such large elements in the product $\mathbf{C} \cdot \mathbf{s}$. This is summed up as follows:

Corollary 1. *Let* $\kappa, r \in \mathbb{N}^+, \mathbf{s} \in \mathbb{F}_q^k, \beta = ||\mathbf{s}||_\infty$ *and define* $S_\kappa^\beta = \{\mathbf{h} \in \mathbb{F}_q^{r \cdot \kappa} \mid \exists T \subseteq [r \cdot \kappa] \wedge |T| > \kappa \wedge \forall i \in T: \ |\mathbf{h}[i]| \geq \frac{1}{2} \cdot \beta\}$. *If* $r \geq 5$ *then*

$$\Pr_{\mathbf{C} \leftarrow \mathbb{B}^{(r \cdot \kappa) \times k}} \left[\mathbf{C} \cdot \mathbf{s} \notin S_\kappa^\beta\right] \leq 2^{-\kappa}.$$

The above statements can directly be implemented in our argument system by the means of circuit sampling. Unfortunately, this results in a new problem, which is that we cannot output the product of $\mathbf{s}$ with a random binary matrix to $\mathcal{V}$ without necessarily leaking information about $\mathbf{s}$.

We resolve this problem using circuit sampling on the side of the prover and give two different solutions. The first idea is that $\mathcal{P}$ can compute $\mathbf{u} = \mathbf{Cs}$ and output $\mathbf{u}+$ "small" where "small" is a value of small norm. To achieve good soundness guarantees we let "small" only be polynomially bigger than $||\mathbf{u}||_\infty$ and use Rejection Sampling to hide the information from the product. Alternatively, we can allow $\mathcal{P}$ to prove knowledge of the bit decomposition of each value of $\mathbf{u} = \mathbf{Cs}$. We now describe both ideas in more detail.

1^{st} Approach: Rejection Sampling. In this solution, we let the prover $\mathcal{P}$ add additional random elements $x_1, x_2, \dots$ to the witness, which are supposed to be small. The verifier $\mathcal{V}$ will then, as part of his challenge in the circuit sampling, ask $\mathcal{P}$ to open a subset of $x_1, x_2, \dots$ to show that most of the remaining ones are indeed of small size. $\mathcal{P}$ will then open sums of each $\mathbf{u}[i]$ with some x_j, subject to the constraint that this does not leak information about $\mathbf{s}$. $\mathcal{V}$ later tests that each such $\mathbf{u}[i] + x_j$ is of bounded norm.

As part of rejection sampling a prover aborts whenever the argument would leak information. But our goal is that the argument is complete with overwhelming probability. To achieve this, we use an idea which is inspired by the "imperfect proof" of [BDLN16]. There, the authors gave a protocol that showed how to prove knowledge of $\ell - \kappa$ out of ℓ SIS instances using cut-and-choose and rejection sampling. Their approach aborts only with negligible probability and turns out to be compatible with our application. The circuit sampler, on a high level, works as follows:

1. $\mathcal{P}$ will sample $x_1, \dots, x_{16\kappa}$ uniformly at random from $[-\pi \cdot m \cdot \beta, \pi \cdot m \cdot \beta] \subset \mathbb{F}$ and commit them as part of $\widehat{w}$.
2. $\mathcal{V}$ with probability 1/2 puts each x_i into a set E. It samples a random matrix $\mathbf{C} \in \mathbb{B}^{5\kappa \times m}$ and sends $E, \mathbf{C}$ as challenges to $\mathcal{P}$.
3. $\mathcal{P}$ now sets up a circuit C as follows:
   (a) C will output $\{x_e\}_{e \in \overline{E}}$. $\mathcal{V}$ checks that $x_e \in [-\pi \cdot m \cdot \beta, \pi \cdot m \cdot \beta]$.
   (b) Compute $\mathbf{u} = \mathbf{Cs}$ in the circuit. $\mathcal{P}$ will go through $\mathbf{u}[1], \dots, \mathbf{u}[5 \cdot \kappa]$, take the first unused $e \in E$ and test if $\mathbf{u}[i] + x_e \in [-(\pi-1) \cdot m \cdot \beta, (\pi-1) \cdot m \cdot \beta]$. If so, then it makes C output $v_i = \mathbf{u}[i] + x_e$, otherwise it removes e from E and repeats this procedure with the next-largest $e' \in E$. $\mathcal{V}$ checks that $v_i \in [-(\pi - 1) \cdot m \cdot \beta, (\pi - 1) \cdot m \cdot \beta]$.

We present the full sampler in the full version, together with a proof of the following Theorem.

Theorem 5. *The aforementioned approach yields a statistically correct, α-sound and perfectly simulatable circuit sampler for the relation $R_{\mathrm{SIS}}^{m,n,q,4\pi m\cdot\beta}$ where $\alpha = \max\{1/|\mathbb{F}|, 2^{-\kappa}\}$.*

A drawback of this approach is the rather big slack of $4\pi \cdot m$. This slack is caused by two reasons. First, there is an inherent increase of m due to the use of Lemma 4. In addition, using Rejection Sampling means that we lose another factor $\pi = 100$. One could decrease the constant by using a discrete Gaussian distribution for the x_i as in [Lyu12], but we opted for presenting the above idea due to its simplicity. On the positive side, there are no non-linear gates in the sampled circuit and $\mathcal{P}$ will only have to add $16 \cdot \kappa$ more values to the witness, independently of β. The sampled circuit will output $\ell + 5\kappa + 1$ elements of $\mathbb{F}$, which in expectancy is around $13\kappa + 1$ (since each of the 16κ random samples is opened with probability 1/2).

Summing up, the communication of the argument (excluding $\tau_{\mathcal{P}}$) when using $\Pi_{\mathsf{sac}}^{\mathrm{samp}}$ is $\mathtt{base} + \log_2 q \cdot M(m + 29\kappa + 1)$ bits.

2^{nd} Approach: The Power of Random Bits. The previous solution has the disadvantage of having a comparably high slack of $4\pi m$. On the other hand, it does not use any non-linear gates. We will now show how to decrease the slack to be essentially m by reintroducing one square gate and adding computational work. To reduce the slack, we will again rely on Lemma 4. But instead of performing rejection sampling on the output, we perform a range proof for each element of the matrix product $\mathbf{u} = \mathbf{Cs}$. The problem that arises is that $\mathbf{C}$ is only chosen at runtime, while the committed witness must be independent of the actual values in $\mathbf{C}$. At the same time, we must construct the argument in such a way that the circuit C will not reveal any information about the product except for bounds on each value.

We resolve this problem as follows: if the witness has $||\mathbf{s}||_\infty \leq \beta$, then since $\mathbf{C} \in \mathbb{B}^{\kappa\times m}$ it must hold that $||\mathbf{Cs}||_\infty \leq m \cdot \beta$. Thus, letting r be the smallest integer such that $m \cdot \beta < 2^r$, it suffices for the prover to show that $\mathbf{u}[i] \in [-2^r, 2^r - 1]$ (which can be done using bit decomposition as in the generalization of Sect. 5.1). To show the inclusion $\mathcal{P}$ can add random bits $x_0^i, \ldots, x_r^i$ to the witness. Then, once the challenge is received from $\mathcal{V}$ and $\mathbf{u}$ is known to $\mathcal{P}$, it can compute the bit decomposition $\mathbf{u}[i] + 2^r = \sum_{j=0}^{r} 2^j h_j^i$ for each $i \in [\kappa]$ and tell $\mathcal{V}$ for each $j \in \{0, \ldots, r\}$ if it should use x_j^i or $1 - x_j^i$ to represent h_j^i. As all x_i^j are chosen randomly, this yields a simulatable circuit. The only issue that remains is for $\mathcal{P}$ to prove that each x_i^j is indeed a bit. For this task, we use the method presented in Sect. 5.2, which uses polynomial evaluation and requires a single non-linear gate. We describe the full circuit sampler in the full version, together with a proof of the following Theorem.

Theorem 6. *Assume that $(q-1)/2 > 4m\beta$. The aforementioned approach yields a perfectly correct, α-sound and perfectly simulatable circuit sampler for the relation $R_{\mathrm{SIS}}^{m,n,q,2m\cdot\beta+4}$ where $\alpha = \max\{\frac{2(r+1)\kappa-1}{|\mathbb{F}|}, 2^{-\kappa}\}$ and r is the smallest integer such that $m \cdot \beta \leq 2^r - 1$.*

Table 1. Parameters used in the experiments for $\Pi_{\mathsf{c\&c}}$ and argument size per parameter set as a function of $\rho = m \cdot \log_2 |\mathbb{F}|$.

N	Cut-and-Choose					
	$\xi \leq 2^{-40}$			$\xi \leq 2^{-80}$		
	M	τ	Comm. of $\mathcal{P}$ (in KB)	M	τ	Comm. of $\mathcal{P}$ (in KB)
2	75	34	$31 + 0.123 \cdot \rho$	145	63	$61.1 + 0.246 \cdot \rho$
4	55	32	$22.4 + 0.069 \cdot \rho$	105	57	$44.8 + 0.144 \cdot \rho$
8	55	38	$20.7 + 0.051 \cdot \rho$	95	57	$42 + 0.114 \cdot \rho$
16	45	26	$23.4 + 0.057 \cdot \rho$	95	63	$41.5 + 0.096 \cdot \rho$
32	45	28	$23.8 + 0.051 \cdot \rho$	85	47	$50.4 + 0.114 \cdot \rho$
64	45	28	$26 + 0.051 \cdot \rho$	85	49	$53 + 0.108 \cdot \rho$

The circuit we obtain has $m + \kappa(r+1)$ inputs, one square gate and $\kappa + 2$ outputs. Then the total communication of this argument when using $\Pi_{\mathsf{sac}}^{\mathrm{samp}}$ is $\mathtt{base} + \log_2 q \cdot M(m + \kappa(r+2) + 5)$ bits.

6 Evaluation and Experimental Results

We ran extensive experiments to measure the performance of our two protocols for the Binary-SIS problem. As setup we used Amazon C5.9xlarge instances using two servers with Intel Platinum 8000 series processors (Skylake-SP) which have clock speed up to 3.4 GHZ, 36 virtual cores per server (utilized based on the experiment setup) and 72 Gb RAM. The network bandwidth between the nodes is 10 Gpbs. For our implementation we used only the baseline construction for the Binary-SIS problem presented in Sect. 5.1. Nevertheless, this includes the three general optimizations described in Sect. 3.4. Hash functions as well as commitments were implemented using SHA-256. Generation of pseudo-randomness from a seed was done using AES in counter-mode where the seed is the AES key. Thus, $|\mathsf{hash}| = |\mathsf{com}| = 256$ bits and $|\mathsf{sd}| = 128$ bits.

We used five sets of parameters for our experiments: (i) $\log_2 |\mathbb{F}| = 15$, $n = 256$ and $m = 1024$; (ii) $\log_2 |\mathbb{F}| = 15$, $n = 256$ and $m = 4096$; (iii) $\log_2 |\mathbb{F}| = 31$, $n = 512$ and $m = 2048$; (iv) $\log_2 |\mathbb{F}| = 59$, $n = 1024$ and $m = 4096$; and (v) $\log_2 |\mathbb{F}| = 61$, $n = 1024$ and $m = 4096$.

The first parameter set reflects SIS-based constructions that do not need any additional functionality. For example, they can be used to instantiate [KTX08] with a binary secret. The second parameter set is then used to study the impact of using a much larger message in the commitment scheme, which also shows how the matrix size impacts the runtimes. The third set would be a typical example for SIS-based constructions such as somewhat homomorphic commitments and allows to prove that a committed message is small. An example for an application would be the commitment scheme of [BDL+18]. The last two sets are used for applications such as somewhat homomorphic encryption schemes like [BGV14].

Table 2. Parameters used in the experiments for Π_{sac} and argument size per parameter set as a function of $\rho = m \cdot \log_2 |\mathbb{F}|$.

N	Sacrificing			
	$\xi \leq 2^{-40}$		$\xi \leq 2^{-80}$	
	M	Comm. of $\mathcal{P}$ (in KB)	M	Comm. of $\mathcal{P}$ (in KB)
2	40	$26.2 + 0.16 \cdot \rho$	80	$51.8 + 0.32 \cdot \rho$
4	20	$16 + 0.08 \cdot \rho$	40	$31.3 + 0.16 \cdot \rho$
8	14	$13.2 + 0.056 \cdot \rho$	27	$24.8 + 0.108 \cdot \rho$
16	10	$10.9 + 0.04 \cdot \rho$	20	$21.2 + 0.08 \cdot \rho$
32	8	$9.9 + 0.032 \cdot \rho$	16	$19.1 + 0.064 \cdot \rho$
64	7	$9.6 + 0.028 \cdot \rho$	14	$18.6 + 0.056 \cdot \rho$

Table 3. Best running times in MSec for different sets of SIS parameters, $\kappa = 40$.

$\log_2 \lvert\mathbb{F}\rvert$	n	m	Cut-and-Choose				Sacrificing		
			N	M	τ	Time	N	M	Time
15	256	1024	2	75	34	73.2	4	20	59.4
15	256	4096	2	75	34	295.8	4	20	252.6
31	512	2048	2	75	34	252.3	4	20	217.5
59	1024	4096	2	75	34	1010.4	2	40	1075.1
61	1024	4096	2	75	34	1204.6	2	40	1228.8

We ran experiments for 40 and 80 bits of statistical security κ. For the parameter N, i.e. the number of parties in the underlying MPC protocol, we used the values $2, 4, 8, 16, 32$ and 64. Then, given the desired level of security and N we searched for the parameters for each protocol that minimized the overall cost.

In $\Pi_{\mathsf{c\&c}}$, there are two parameters to define: M (number of pre-processing executions) and τ (number of pre-processing executions to open). To obtain these, we wrote a script that finds the minimal M and τ such that $\xi(M, N, \tau) \leq 2^{-40}$ or 2^{-80}. In Π_{sac}, we observe that for our choices of $|\mathbb{F}|$ and N, it holds that $\frac{3N+|\mathbb{F}|-3}{N \cdot |\mathbb{F}|} \approx \frac{1}{N}$ and so it suffices to choose M such that $\xi(M, N) \approx \frac{1}{N^M} \leq 2^{-40}$ or 2^{-80}.

We summarize the parameters used in our experiments in Tables 1, 2. In addition, for each set of parameters we give the size of the argument in Kbits as a formula of the SIS problem parameters $\rho = m \cdot \log_2 |\mathbb{F}|$. Observe that as the number of parties N grows, the number of MPC instances in Π_{sac} becomes much smaller than the number required in $\Pi_{\mathsf{c\&c}}$, which is translated to smaller proof size. This implies that our new 'sacrificing'-based approach *outperforms* the 'cut–and–choose'-based method for arithmetic circuits over large fields.

Running Times. In Table 3 we present the running times (in Msec.) of the two protocols for 40 bits of security respectively. The results for 80-bit of security are presented in the full version. For each set of parameters for the SIS problem we report only the best running times achieved together with the MPC protocol parameters which lead to the result. As the number of non-linear gates in this circuit is small, it is not surprising that both schemes achieve similar results. Observe that small numbers of parties in the MPC protocol lead to faster running times, in contrast to proof size which is getting smaller when the number of parties is increased.

It is worth noting that a major source of improvement we discovered was to postpone the modular reduction in the matrix multiplication to the end. That is, when the prover/verifier multiply a row in the matrix $\mathbf{A}$ with a vector of shares of $\mathbf{s}$ (which is eventually what the computed circuit does), it is highly beneficial to do the reduction modulo q only at the end of the matrix multiplication. This simple optimization alone yields an improvement of approximately 33%.

Using Multi-threads. The above results were obtained using a single thread. As computation time is the bottleneck, we examined what happens when working with multiple threads which seems to be a straightforward optimization. This experiment was run for the "toughest" instance of the SIS problem, with $\log|\mathbb{F}| = 61$, $n = 1024$ and $m = 4096$ and with the MPC protocol parameters who yielded the best running time in Table 3. The full results appear in the full version. As we discovered, using two threads already cut the running time by half and using *20 threads speeds-up the runtime by more than 80%.* As a consequence, we obtain a ZKAoK that runs in *less than 0.5 s* even for the of SIS instance with the largest parameters. This is orders of magnitude faster than any previous implementation for arithmetic circuits of the same size.

Faster Matrix Products and Structured Lattices. In this work we solely focus on unstructured matrices $\mathbf{A}$ for SIS. By micro-benchmarking the results, we observe that as the size of the matrix $\mathbf{A}$ grows, the time spent on computing the matrix multiplication becomes dominant. In particular, for the large instances, matrix multiplication takes >85% of the overall local computation time. As we use only textbook matrix multiplication, this leaves plenty of room for improvement. Furthermore, on the verifier side it is possible to batch the matrix multiplications together as only verification is needed. Another direction would be to use structured matrices i.e. structured lattices, which opens the door for FFT-like algorithms.

7 Related Work

The landscape for (lattice-based) ZK arguments has drastically changed during the past years. We will now describe how our protocol compares with other state-of-the-art arguments of knowledge in terms of communication, computation time, accuracy of the proof and the cryptographic assumptions. As most of existing work focuses only on minimizing the proof size, we can only estimate in many cases what will be the running time compared to ours. For this section, we used $N = 16$ parties in underlying MPC protocol for our scheme and set M accordingly to achieve the desired soundness. We stress that it is possible to further increase the number of parties in the underlying MPC and reduce the proof size even more, but at the cost of increasing also the running time.

Protocols for exact SIS. We subsume all protocols that prove the exact solution here. These are either based on Stern-type arguments [LNSW13], direct applications of MPC-in-the-head/IOP [AHIV17, BCR+19] or special-purpose protocols [BLS19, Beu19, YAZ+19]. Though STARKs [BBHR19] fall into the second category, we do not consider those as related work as they are rather tailored to computations with looping components. While [LNSW13] is a specific technique tailored to problems such as SIS, [AHIV17, BCR+19] require an arithmetic circuit (similar to us) for the verification of the statement. The comparison in term of proof size to these works is presented in Table 4.

Table 4. Proof sizes for Binary-SIS and 5-bit secrets, small constant slack, $\kappa = 40$.

	$\lvert\mathbb{F}\rvert \approx 2^{32}$ Binary	$\lvert\mathbb{F}\rvert \approx 2^{32}$ $\beta = 15$	$\lvert\mathbb{F}\rvert \approx 2^{61}$ Binary	$\lvert\mathbb{F}\rvert \approx 2^{61}$ $\beta = 15$
Stern [Ste96]	971 KB	7285 KB	3703 KB	27775 KB
Ligero [AHIV17]	45 KB	55 KB	55 KB	80 KB
Ours, baseline	357 KB	2138 KB	1359 KB	8148 KB
Ours, amortized	179 KB	1069 KB	680 KB	4075 KB

We did not include proof sizes for the Aurora protocol [BCR+19], as the authors there did not provide a general expression for the proof size, but rather experimental results for the binary field $\mathbb{F}_{2^{192}}$. Nevertheless, we expect them to be comparable to the sizes reported for [AHIV17]. We note that the prover running time according to their experiments is ≈200 s, and so is expected to be at least one order of magnitude bigger than in our protocols. The same applies to Ligero [AHIV17], which requires extensive FFT computations for large polynomials, which cause the prover's running time to be much higher than ours. We thus conclude that these approaches, which achieve sun-linear communication, outperform our approach in the non-interactive setting. However, in the interactive setting- for example, when used as a building block in a larger interactive protocol (that use e.g. lattice-based commitments) with strong runtime requirements then our computationally efficient prover is advantageous. Concurrently to this work, the works of [BLS19, Beu19, YAZ+19] have improved upon the state of the art of ZKAoK for lattice-based primitives. While it can be expected that their solutions have the same or better communication complexity than our approach for exact SIS, it is still unclear what is their computational cost, as none of these works provides an implementation. Furthermore, in comparison to their work our protocols can be used to prove arbitrary statements.

Protocols for SIS with Slack. Here, we compare with the argument system from the signature scheme of [Lyu12] (see Table 5).

We compare the proof size of [Lyu12] with our baseline protocol and with the two solutions described in Sect. 5.3. We see that in particular the 2nd protocol of Sect. 5.3 improves upon [Lyu12] for all three considered cases. This is particularly true in the cases where the gap between β and $|\mathbb{F}|$ is small, as our proof size increases as $|\mathbb{F}|$ grows whereas the size of [Lyu12] depends on the bound β but not on $|\mathbb{F}|$ when optimized correctly. At the same time, increasing β seems not to substantially change the communication complexity of either of our two proofs, whereas it has a direct impact on [Lyu12].

Table 5. Proof sizes for non-constant slack with $\log_2(|\mathbb{F}|) = 32$ and $\kappa = 40$.

Protocol	Slack	Binary SIS	SIS with $\beta = 15$
Sigma-protocol [Lyu12]	288 m	184 KB	223 KB
Ours, Approach 1 ($\kappa = 8$)	400 m	100 KB	100 KB
Ours, Approach 2 ($\kappa = 8$)	<3 m	96 KB	97 KB
Ours, Exact	1	179 KB	1069 KB

Other Approaches. Recently, del Pino et al. [dPLS19] showed how to obtain a ZK argument for our problem setting. While they have a drastically smaller proof size (in the order of 1.5 KB), their construction relies on the DLog assumption and is therefore not post-quantum secure. Moreover, their computational efficiency relies on using structured lattices, which we do not need. The same applies to Hyrax [WTS+18], Sonic [MBKM19] or Libra [XZZ+19], who rely on the DLog-assumption. Older ZK-SNARKs such as [PHGR16, BSCTV14] would offer low argument size and verification time but in addition to large keys and a high prover runtime also rely on very strong assumptions. Similarly, the work of [BCC+16] is also in the DLog setting. Its lattice-based variant [BBC+18] is so far not implemented, may have large hidden constants and itself uses ZKAoKs for SIS as building blocks.

Thanks and Acknowledgements. The authors want to thank Roey Sefi and Assi Barak for their help with the implementation as well as Carmit Hazay, Yehuda Lindell and Avishay Yanai as well as the anonymous reviewers for their helpful comments.

The work of both authors was mainly done at Bar Ilan University. Both authors acknowledge support by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office. The work of Carsten was additionally funded by the European Research Council (ERC) under the European Unions' Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO).

References

[AHIV17] Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: lightweight sublinear arguments without a trusted setup. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM (2017)

[BBC+18] Baum, C., Bootle, J., Cerulli, A., del Pino, R., Groth, J., Lyubashevsky, V.: Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 669–699. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_23

[BBHR19] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 701–732. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_23

[BCC+16] Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 327–357. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_12

[BCR+19] Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_4

[BD10] Bendlin, R., Damgård, I.: Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 201–218. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11799-2_13

[BDL+18] Baum, C., Damgård, I., Lyubashevsky, V., Oechsner, S., Peikert, C.: More efficient commitments from structured lattice assumptions. In: Catalano, D., De Prisco, R. (eds.) SCN 2018. LNCS, vol. 11035, pp. 368–385. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98113-0_20

[BDLN16] Baum, C., Damgård, I., Larsen, K.G., Nielsen, M.: How to prove knowledge of small secrets. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 478–498. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_17

[Bea91] Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_34

[Beu19] Beullens, W.: On sigma protocols with helper for MQ and PKP, fishy signature schemes and more. Cryptology ePrint Archive, Report 2019/490 (2019). https://eprint.iacr.org/2019/490

[BGV14] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. ACM Trans. Comput. Theory (TOCT) **6**(3), 1–36 (2014)

[BHR12] Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. ACM (2012)

[BJM19] Badertscher, C., Jost, D., Maurer, U.: Agree-and-prove: generalized proofs of knowledge and applications. Cryptology ePrint Archive, Report 2019/662 (2019). https://eprint.iacr.org/2019/662

[BL17] Baum, C., Lyubashevsky, V.: Simple amortized proofs of shortness for linear relations over polynomial rings (2017). https://eprint.iacr.org/2017/759

[BLS19] Bootle, J., Lyubashevsky, V., Seiler, G.: Algebraic techniques for short(er) exact lattice-based zero-knowledge proofs. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11692, pp. 176–202. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26948-7_7

[BSCTV14] Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von Neumann architecture. In: USENIX Security Symposium (2014)

[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

[DFMS19] Don, J., Fehr, S., Majenz, C., Schaffner, C.: Security of the fiat-shamir transformation in the quantum random-oracle model. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11693, pp. 356–383. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26951-7_13

[DN19] Dinur, I., Nadler, N.: Multi-target attacks on the picnic signature scheme and related protocols. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 699–727. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_24

[dPLS19] del Pino, R., Lyubashevsky, V., Seiler, G.: Short discrete log proofs for FHE and ring-LWE ciphertexts. In: Lin, D., Sako, K. (eds.) PKC 2019. LNCS, vol. 11442, pp. 344–373. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17253-4_12

[DPSZ12] Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_38

[FS86] 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

[GMO16] Giacomelli, I., Madsen, J., Orlandi, C.: Faster zero-knowledge for Boolean circuits. In: USENIX Security Symposium, Zkboo (2016)

[GMR89] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. **18**(1), 186–208 (1989)

[IKOS07] Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing. ACM (2007)

[KKW18] Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018 (2018)

[KTX08] Kawachi, A., Tanaka, K., Xagawa, K.: Concurrently secure identification schemes based on the worst-case hardness of lattice problems. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 372–389. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89255-7_23

[LN17] Lindell, Y., Nof, A.: A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017 (2017)

[LNSW13] Ling, S., Nguyen, K., Stehlé, D., Wang, H.: Improved zero-knowledge proofs of knowledge for the ISIS problem, and applications. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 107–124. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36362-7_8

[Lyu12] Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_43

[MBKM19] Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: zero-knowledge snarks from linear-size universal and updatable structured reference strings. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 2111–2128 (2019)

[PHGR16] Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: nearly practical verifiable computation. Commun. ACM **59**(2) (2016)

[Ste96] Stern, J.: A new paradigm for public key identification. IEEE Trans. Inf. Theory **42**(6) (1996)

[WTS+18] Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: Proceedings of the 2018 IEEE Symposium on Security and Privacy, SP 2018 (2018)

[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

[YAZ+19] Yang, R., Au, M.H., Zhang, Z., Xu, Q., Yu, Z., Whyte, W.: Efficient lattice-based zero-knowledge arguments with standard soundness: construction and applications. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11692, pp. 147–175. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26948-7_6