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

Carsten Baum* Ariel Nof*

Department of Computer Science, Bar-Ilan University
carsten.baum@biu.ac.il, ariel.nof@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 uses MPC with preprocessing such as recently proposed by Katz, Kolesnikov and Wang (ACM CCS 2018). Our argument system uses only symmetric-key primitives and utilizes a version of the so-called SPDZ-protocol which has efficiency benefits for arithmetic circuits compared to other approaches.

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 and show how our solution compares in terms of argument size to existing work. We moreover 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 seconds*. To the best of our knowledge, our construction outperforms all known approaches for the SIS problem with post-quantum security either in terms of computation or communication complexity.

1 Introduction

Zero-Knowledge Arguments of Knowledge (ZKAoK) are interactive protocols that allow a computationally bounded prover to convince a verifier that she knows a witness for a certain statement, without revealing any further information about the witness. Since their introduction in the 80ies [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,BSBHR18,WTS+18,BSCR+18] 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 $\boldsymbol{t}$ while the prover wants to prove knowledge of a secret $\boldsymbol{s}$ such that $\boldsymbol{t} = \mathbf{A}\boldsymbol{s} \bmod q$ and $||\boldsymbol{s}||_\infty \leq \beta$. The SIS problem and related problems are crucial building blocks for post-quantum lattice-based cryptography and constructing efficient ZKAoK with a small communication complexity 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],

* Supported 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.

who were the first to examine arguments of knowledge for SIS using generic proof systems. They observed that for certain argument schemes, the above SIS function has both a very low multiplication depth and low total number of multiplications, if the secret $\boldsymbol{s}$ is a binary vector. However, many general ZKAoK systems only provide asymptotic efficiency, meaning that they require the circuit to be very big before their strengths play out [AHIV17,BBC$^+$18]. Moreover, many of these approaches achieve sub-linear communication complexity at the cost of high prover's running time [PHGR16]. Other approaches are insecure to post-quantum attacks [WTS$^+$18,MBKM19,BCC$^+$16,PHGR16] or rely on knowledge assumptions that are poorly understood. Finally, none of existing general solutions takes advantage of the unique structure of the SIS problem.

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 what this paradigm is.

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 it can be computed ahead of time. To the best of our knowledge, the only MPC-in-the-head scheme to use preprocessing was introduced recently in [KKW18].

1.2 Our Contributions

In this work, we construct a new practically efficient ZKAoK for arithmetic circuits, together with a multitude of extensions to make it applicable to 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. The presentation of the two constructions is in Section 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 $\boldsymbol{s}$ contains bounded values, so the first part comes essentially for free. We initially tweak the approach of [BD10] and only allow $\boldsymbol{s}$ to consist of bits, which allows for a very fast argument of size using one square gate per element of $\boldsymbol{s}$. Then, we show how to handle more general distributions of $\boldsymbol{s}$ and introduce some specific optimizations to reduce communication and computation. This is described in Section 5.

Experimental Results. We implemented our zero-knowledge protocol for the Binary SIS problem (i.e., where the secret $\boldsymbol{s}$ 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 seconds 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 Section 6.

Sampling circuits on the fly. A major source of optimizations to our application is the fact that our "MPC-in-the-head" protocol essentially 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 Section 4.

1.3 Related Work

The past few years saw a tremendous progress in the efficiency of zero-knowledge arguments of knowledge for arithmetic circuits. ZK-SNARKs (Succinct Non-Interactive Arguments of Knowledge) such as Pinocchio [PHGR16] and libsnark [BSCTV14] offer a very low argument size and verification time, at the expense of keys in the size of Megabytes and a large prover runtime due to the use of pairings. This also means that they are inherently not post-quantum secure. From weaker assumptions, Bootle et al. [BCC^{+}16] recently constructed a highly efficient argument system, a variant of which can also be adapted to the post-quantum setting [BBC^{+}18]. Albeit asymptotically more efficient, our construction outperforms [BBC^{+}18] for small circuits. In addition to [BCC^{+}16], Wahby et al. [WTS^{+}18] present a practically efficient ZK-Snark from the Discrete Log assumption. MPC-in-the-head [IKOS07] was first shown to be practical by the work of Giacomelli et al. [GMO16]. Later works such as the Ligero argument system [AHIV17] improved upon that work and provided an argument with proof size that is sub-linear in the size of the circuit C. Recently, Ben-Sasson et al. presented a new Zero-Knowledge Interactive Oracle Proof called Stark [BSBHR18] which improves asymptotically upon IKOS-style proof systems, as well as a practically efficient protocol called Aurora [BSCR^{+}18] with proof size that is only poly-logarithmic in C. These constructions are asymptotically better than ours in terms of proof size. However, the computational work of the prover is orders of magnitudes higher than in our construction due to the extensive use of polynomial interpolation. In addition, they do not have the property that linear gates have no effect on the complexity, which makes our protocol fit to applications such as SIS.

Zero-Knowledge Arguments for SIS. While the approach of [BD10] can more be seen as a theoretical result, there are three main lines of work for SIS arguments of knowledge. The identification scheme of Stern [Ste96] was modified into a argument of knowledge by Kagawa et al. [KTX08] for their commitment scheme, which was then generalized to be a proper argument of knowledge by Ling et al. [LNSW13]. This argument system has constant soundness error and its efficiency quickly degrades with the bound β on the witness. The argument is extremely versatile and allows constructions such as e.g. ring signatures [LLNW17].

A second line of work more directly follows standard Schnorr-style proofs and uses rejection sampling to be zero-knowledge [Lyu09,Lyu12]. Like Stern-style arguments it has constant soundness error, but it supports also non-binary secrets with negligible communication overhead. On the other hand, the soundness guarantee only holds with some "slack", meaning that while an honest prover starts with a secret $||\boldsymbol{s}||_\infty \leq \beta$, the protocol only guarantees that $||\boldsymbol{s}||_\infty \leq \tau \cdot \beta$ where τ depends, among other things, on the dimensions of $\mathbf{A}$.

A third was recently introduced by del Pino et al. [dPLS19]. They use a variant of the Bulletproof scheme to adapt range proofs for the structured lattice setting. As they argue in the paper, the computational efficiency of their scheme hinges on using Ring-SIS or Ring-LWE primitives, while the soundness also relies on the Discrete Log assumption.

We provide a detailed comparison of these works with our construction in Section 7, including comparison of proof size, whenever possible, for the SIS application.

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. The elements are being represented by the interval $[-(q-1)/2, (q-1)/2]$. We use λ as the computational and κ as the statistical security parameters, and generally assume that $q \approx poly(\lambda, \kappa)$. We use bold lower-case letters such as $\boldsymbol{s}$ to denote a vector and bold upper-case letters like $\mathbf{A}$ for matrices. We let $\boldsymbol{s}[i]$ denote the ith component of the vector $\boldsymbol{s}$. Furthermore, we use $[n]$ as an abbreviation for the set $\{1, \ldots, n\}$.

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 a field $\mathbb{F}$, and each wire w will hold a value from this field or $\perp$ initially. C has $n_{\mathtt{in}}$ input wires as well as $n_{\mathtt{out}}$ 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 will have $n_{\mathtt{gates}} = n_C - n_{\mathtt{in}}$ gates in total and we define the set $\mathcal{G} = \{n_{\mathtt{in}} + 1, \ldots, n_C\}$.

We define functions $L : \mathcal{G} \to \mathcal{W} \setminus \mathcal{O}$, $R : \mathcal{G} \to (\mathcal{W} \setminus \mathcal{O}) \cup \{\perp\}$ 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 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 $\mathtt{eval}(C, x)$ with $x \in \mathbb{F}^{n_{\mathtt{in}}}$ then runs the circuit as follows

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

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 $C(x) = y$ if $\mathtt{eval}(C, x)$ returns the value $y \in \mathbb{F}^{n_{\mathtt{out}}}$.

We denote by n_{mul} and n_{sq} the number of multiplication and square gates in the circuit respectively.

2.2 Zero-Knowledge Proof 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 iTM is a probabilistic polynomial-time TM. Let $L_R \subseteq \{0,1\}^*$ be an NP language and R be its related NP-relation for circuits over $\mathbb{F}$. Thus $(x = (C, y), w) \in R$ iff $(C, y) \in L_R$ and $\mathtt{eval}(C, w) = y$. We write $R_x = \{w \mid (x, w) \in R\}$ for the set of witnesses for a fixed x.

Honest Verifier Proof of Knowledge (HVPoK). Assume $(\mathcal{P}, \mathcal{V})$ is a pair of PPT iTMs and let $\xi : \{0,1\}^* \to [0,1]$ be a function. We say that $(\mathcal{P}, \mathcal{V})$ is a *proof of knowledge* with *knowledge error* ξ if the following properties hold:

Completeness: If $\mathcal{P}$ and $\mathcal{V}$ follow the protocol on input x and private input w to $\mathcal{P}$ where $(x, w) \in R$, 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 $x \in L$ and oracle access to $\hat{\mathcal{P}}$, the algorithm $\mathcal{E}$ outputs a string w such that $(x, w) \in L_R$ in expected number of steps bounded by $O(\frac{1}{\delta(x) - \xi(x)})$.

We say that $(\mathcal{P}, \mathcal{V})$ is an *honest verifier zero-knowledge proof of knowledge*, if in addition to the two above properties, the following property is also satisfied.

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

This definition is sufficient, since public-coin protocols like the protocols we consider in this work, which satisfies the zero-knowledge with respect to the honest verifier, can be easily transformed to protocols which are zero-knowledge in general by having the verifier commits to his challenges at the beginning of the execution. Moreover, it is possible also to obtain a non-interactive zero-knowledge proof of knowledge (NIZKPoK) via the Fiat-Shamir transformation [FS86]. If the soundness of a proof only holds against a computationally bounded prover, then one generally talks about *computationally sound proofs* which are known or *Arguments of Knowledge* (AoK).

2.3 Commitments and Collision-Resistant Hash Functions (CRHF)

We use *Commitments* and *Collision-Resistant* Hash Functions (CRHF) as buildings blocks in our constructions and thus introduce them now shortly. 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 to different messages m and m'.

A Collision-Resistant Hash Functions (CRHF) is a function H for which it is "hard" to find two inputs x and x' such that $H(x) = H(x')$. Usually, hash functions can shrink a long message into a short digest one. This means that for almost all messages a collision *must* exist. The requirement is therefore that such a collision will be very hard to find for any PPT adversary.

2.4 The Short Integer Solution Problem

We will now formalize the SIS problem, which was already informally introduced in the introduction. Let q be a prime such that $\mathbb{F}_q$ is the base field of the argument system. At the same time, q will also be the modulus of the SIS instance which is defined over $\mathbb{Z}_q$. Though $\mathbb{F}_q \simeq \mathbb{Z}_q$ in our setting, we keep the distinction that we use $\mathbb{F}_q$ when we talk about circuits, while we use $\mathbb{Z}_q$ when we talk about SIS. To define the Short Integer Solution problem, let $n, m \in \mathbb{N}$ such that $n \ll m$. We naturally embed $\mathbb{Z}_q$ into $\mathbb{Z}$ by considering and identifying each $\mod q$-number with an element from the interval $[-\frac{q-1}{2}, \frac{q-1}{2}] \subset \mathbb{Z}$. This allows us to consider the ℓ_p-norm of vectors $\boldsymbol{s} \in \mathbb{Z}_q^m$ and we let $||\boldsymbol{s}||_\infty$ be the ∞-norm of the embedding of $\boldsymbol{s}$ into the module $\mathbb{Z}^m$. We define $S_\beta^m \subset \mathbb{Z}_q^m$ to be the subset of m-element vectors with ℓ_∞-norm $\leq \beta$.

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

We can collect all possible sets of $(\mathbf{A}, \boldsymbol{s}, \boldsymbol{t})$ that fulfill above definition in an NP-relation

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

In practice, one often encounters proofs that do not show exactly that $\boldsymbol{s} \in S_\beta^m$ 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 τ which is mostly referred to as *slack*. We have that $R_{\mathtt{SIS}}^{m,n,q,\beta} \subseteq R_{\mathtt{SIS}}^{m,n,q,\tau\cdot\beta}$ if $\tau \geq 1$, so any honest prover will still make the verifier accept if it proves for $R_{\mathtt{SIS}}^{m,n,q,\tau\cdot\beta}$ instead.

For simplicity, we will also consider a special type of SIS where the vector $\boldsymbol{s}$ of the witness has to be binary.

Definition 2 (Binary-SIS). *Let m, n, q be defined as above. Given $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and $\boldsymbol{t} \in \mathbb{Z}_q^n$, the (inhomogeneous) Binary SIS-problem is to find $\boldsymbol{s} \in \{0,1\}^m$ such that $\boldsymbol{t} = \mathbf{A}\boldsymbol{s} \mod q$.*

This Binary-SIS problem is not uncommon and the versions of the schemes of [BD10,LNSW13] are actually defined for it. We specifically define its relation as

$$R_{\mathtt{B-SIS}}^{m,n,q} = \{(x, w) = ((\mathbf{A}, \boldsymbol{t}), \boldsymbol{s}) \mid \boldsymbol{s} \in \{0,1\}^m \wedge \mathbf{A} \in \mathbb{F}_q^{n\times m} \wedge \boldsymbol{t} = \mathbf{A}\boldsymbol{s}\}$$

3 Honest Verifier Arguments of Knowledge for Arithmetic Circuits

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

3.1 The MPC protocol

Our MPC protocol is a simplified version of the SPDZ 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 the a sharing of x. We use a simple additive secret sharing, i.e., a sharing of x consists of random $x_1, \ldots, x_N \in \mathbb{Z}_q$ such that $x = x_1 + \cdots + x_N \mod q$, where P_i holds x_i. We define the following operations on the secret sharing scheme.

- $\mathsf{open}([\![x]\!])$: In this procedure, the parties reveal the secret x by having each party send its share. Upon receiving x_j from each P_j, party P_i computes $x = \sum_{j=1}^{N} x_j \mod q$.
- $[\![x]\!] + [\![y]\!]$: 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.
- $[\![x]\!] + \sigma$: Given a sharing $[\![x]\!]$ and a public constant σ, party P_1 define $x_1 + \sigma$ as its share of the result, whereas the other parties' shares remain the same.
- $\sigma \cdot [\![x]\!]$: Given a sharing $[\![x]\!]$ and a public constant σ, each party P_i define $\sigma \cdot x_i$ as its share of the product.

Multiplication operation. We say that the triple $([\![a]\!], [\![b]\!], [\![c]\!])$ is a random multiplication triple if a and b are random and $c = a \cdot b$. To multiply two shared values $[\![x]\!]$ and $[\![y]\!]$ using a preprocessed random triple $([\![a]\!], [\![b]\!], [\![c]\!])$, the parties work as follows:

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

The above is the well-known Beaver Circuit Randomization [Bea91] which holds since

$$\begin{aligned}
[\![z]\!] &= [\![c]\!] - \alpha \cdot [\![b]\!] - \beta \cdot [\![a]\!] + \alpha \cdot \beta \\
&= [\![ab]\!] - (x-a) \cdot [\![b]\!] - (y-b) \cdot [\![a]\!] + (x-a) \cdot (y-b) \\
&= [\![ab - xb - ab - ya - ba + xy + ay + xb + ab]\!] \\
&= [\![xy]\!]
\end{aligned}$$

Square operation. We say that the pair $([\![b]\!], [\![d]\!])$ is a random square if b is random and $d = b^2$. To compute the square $[\![x^2]\!]$ given $[\![x]\!]$ using a preprocessed $([\![b]\!], [\![d]\!])$, the parties work as follows:

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

Note that the above holds since

$$[\![z]\!] = \alpha \cdot ([\![x]\!] + [\![b]\!]) + [\![d]\!] = (x-b) \cdot ([\![x]\!] + [\![b]\!]) + [\![b^2]\!] = [\![x^2 - b^2 + b^2]\!] = [\![x^2]\!].$$

The protocol. The above building blocks can easily be combined to securely run $\texttt{eval}(\cdot)$ on a circuit C: after the inputs are secret-shared using $[\![\cdot]\!]$, the parties apply G as defined in Section 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 a 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 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 distributed computation and then send the output shares back to the clients.

Formally, let $\mathcal{F}_{\texttt{tr}}$ and $\mathcal{F}_{\texttt{sq}}$ be ideal functionalities that provide the parties with random multiplication triples and squares. We define $\mathsf{view}_{I,\pi}^{\mathcal{F}_{\texttt{tr}},\mathcal{F}_{\texttt{sq}}}(C)$ to be the view of a subset of parties I during the execution of a protocol π on an arithmetic circuit C, in the $(\mathcal{F}_{\texttt{tr}}, \mathcal{F}_{\texttt{sq}})$-hybrid model and in the client-server model described above. This consists of the input shares, the correlated randomness they receive from the ideal functionalities and the messages they obtain from the other parties while computing multiplication and square gates. We prove the security of the protocol in Theorem 1.

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 $|I| \leq N-1$, there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ such that $\{\mathcal{S}(I,C)\} \equiv \{\mathsf{view}_{I,\pi}^{\mathcal{F}_{\texttt{tr}},\mathcal{F}_{\texttt{sq}}}(C)\}$*

Proof. Intuitively, this follows from the fact that the corrupted parties see only shares of the values on the wires of the circuit that could open to any value or random public values. Formally, the simulator $\mathcal{S}$ begins by choosing t random shares for each input wire and adding them to the view of the corrupted parties in I. Then, it goes over the circuit in topological order; for addition gates and multiplication-by-a-constant, it does the local operation on the corrupted parties' shares as defined by the protocol. For multiplication gates, $\mathcal{S}$ chooses t shares of a,b and c and adds them to the corrupted parties' view. Then, it chooses random α and β, compute the honest parties' shares of them accordingly (knowing the corrupted parties' shares) and adds them to the view. Then, it computes the corrupted parties' shares of z as defined by the protocol. Similarly, for square gates, it chooses t random shares for b and d and adds them to the view of the corrupted parties. Then, it chooses a random α, computes the honest parties' shares accordingly, adds them to the view and computes the corrupted parties' shares of the output.

The only difference between the simulated view and the real execution is the way α and β are chosen. However, in both cases the these values are uniformly distributed. Thus, the view generated by the simulator is identically distributed to the view in the real execution as required. $\square$

3.2 HVZKAoK Protocol using Cut–and–Choose

We are now ready to present our first HVZKAoK protocol $\Pi_{\mathtt{c\&c}}$. It is based on the MPC protocol presented in the previous section and relying on the cut–and–choose technique to generate correct random multiplication triples and squares. The formal description appears in Fig. 1.a and Fig. 1.b.

The idea behind the protocol is that the prover $\mathcal{P}$ proves its knowledge of w such that $C(\boldsymbol{w}) = \boldsymbol{y}$ by simulating a secure N-party computation of the circuit over an additive sharing of 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 his 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, $\mathcal{V}$ chooses randomly τ of them, which are revealed and opened to him. 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 $\mathcal{V}$. The verifier chooses a random subset of $N-1$ parties in each execution, whose views are opened and tested for consistency. If theses two tests passed successfully and the output of the circuit is $\boldsymbol{y}$, then $\mathcal{V}$ outputs acc. Observe that $\mathcal{V}$ cannot learn any information about the witness $\boldsymbol{w}$ 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 he 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 send just 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 (in Section 3.4 we will see that it is possible to reduce the data sent here from N seeds to $\log N$ seeds). 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, $c_{e,k} = a_{e,k} \cdot b_{e,k}$ is also fixed. Thus, when generating the sharing of $c_{e,k}$, it is necessary to "fix" the initial sharing derived from the random seeds. Therefore, 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 to the generation of 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 fix the initial sharing by adding the offset (denoted this time by $\phi_{e,k}$) to obtain a correct sharing of the given input. Thus, the prover $\mathcal{P}$ 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 $C(\boldsymbol{w}) \neq \boldsymbol{y}$. 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 verifier. In order to make the output of the protocol be $\boldsymbol{y}$, $\mathcal{P}$ 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 proved. We therefore provide now a proof that the 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 a secure commitment scheme. Then, the protocol $\Pi_{\mathsf{c\&c}}$ is an HVZKAoK with knowledge error (soundness) $\xi_{\mathsf{c\&c}}(M,N,\tau)$.*

Proof. We prove the that each of the three properties defined in Section 2.2 are satisfied by our protocol.

Completeness. This follows trivially from the correctness of the MPC protocol.

Honest Verifier Zero Knowledge. This property follows from the security of the MPC protocol as defined in Theorem 1 and by the hiding property of the commitment scheme. Specifically, let $\mathcal{S}_\pi$ be the simulator that exists for the MPC protocol described in the proof of Theorem 3.1. We construct a honest-verifier zero-knowledge simulator $\mathcal{S}$ for our protocol, which works as follows:

1. $\mathcal{S}$ chooses random $E \subset [M]$ such that $|E| = \tau$. Then, for each $e \in \bar{E} = [M] \setminus E$, it chooses a random $i_e \in [N]$.
2. For each $e \in E$, $\mathcal{S}$ prepares the pre-processing data as an honest prover would do in Round 1, with one exception: it computes Π_e as a commitment to a 0-string.
3. For each $e \in \bar{E}$, $\mathcal{S}$ chooses $\mathsf{sd}_{e,i}$ for each $i \in I_e = [N] \setminus \{i_e\}$. Then, for the pre-processing in Round 1, it generates state_e by choosing random $\Delta_{e,k}$ for each multiplication/square gate. In addition, it chooses random $\phi_{e,k}$ for each input wire k and compute Π_e accordingly. Proceeding

The "Cut–and–Choose" Based HVZK Argument $\Pi_{\mathsf{c\&c}}$ (Part 1)

Let H be a collision-resistant hash function and com be a commitment scheme

Inputs: Both the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ hold $\boldsymbol{y} \in \mathbb{F}^{n_{\mathtt{out}}}$, a description of an arithmetic circuit C over a finite field $\mathbb{F}$ and parameters M, N, τ; the prover $\mathcal{P}$ also hold a witness $\boldsymbol{w} \in \mathbb{F}^{n_{\mathtt{in}}}$ such that $C(\boldsymbol{w}) = \boldsymbol{y}$.

Round 1:

1. For each $e \in [M]$:
   (a) $\mathcal{P}$ initializes an empty string state_e.
   (b) $\mathcal{P}$ chooses a master seed sd_e and use it to generate $\mathsf{sd}_{e,1}, \dots, \mathsf{sd}_{e,N}$.
   (c) For each multiplication gate $g_k \in \mathcal{G}$:
      i. $\mathcal{P}$ defines three random sharings $[\![a_{e,k}]\!], [\![b_{e,k}]\!]$ and $[\![c_{e,k}]\!]$ by using $\mathsf{sd}_{e,i}$ for each $i \in [N]$ to generate $a_{e,k,i}, b_{e,k,i}$ and $c_{e,k,i}$.
      ii. $\mathcal{P}$ computes $a_{e,k} = \sum_{i=1}^{N} a_{e,k,i}$ and $b_{e,k} = \sum_{i=1}^{N} b_{e,k,i}$ and $c_{e,k} = a_{e,k} \cdot b_{e,k}$.
      Then, it sets $\Delta_{e,k} = c_{e,k} - \sum_{i=1}^{N} c_{e,k,i}$ and $\mathsf{state}_e \leftarrow \mathsf{state}_e || \Delta_{e,k}$.
      iii. $\mathcal{P}$ sets the random triple for this gate to be $([\![a_{e,k}]\!], [\![b_{e,k}]\!], [\![c_{e,k}]\!] + \Delta_{e,k})$
   (d) For each square gate $g_k \in \mathcal{G}$:
      i. $\mathcal{P}$ defines two random sharings $[\![b_{e,k}]\!]$ and $[\![d_{e,k}]\!]$ by using $\mathsf{sd}_{e,i}$ for each $i \in [N]$ to generate $b_{e,k,i}$ and $d_{e,k,i}$.
      ii. $\mathcal{P}$ computes $b_{e,k} = \sum_{i=1}^{N} b_{e,k,i}$.
      Then, it computes $\Delta_{e,k} = (b_{e,k})^2 - \sum_{i=1}^{N} d_{e,k,i}$ and $\mathsf{state}_e \leftarrow \mathsf{state}_e || \Delta_{e,k}$.
      iii. $\mathcal{P}$ sets the random square to this gate to be $([\![b_{e,k}]\!], [\![d_{e,k}]\!] + \Delta_{e,k})$.
   (e) $\mathcal{P}$ chooses a linear random sharing of the inputs:
      i. For each $i \in [N]$, $\mathcal{P}$ uses $\mathsf{sd}_{e,i}$ to generate $w_{e,1,i}, \dots, w_{e,n_{\mathtt{in}},i}$.
      ii. For each $k \in \mathcal{I}$, $\mathcal{P}$ sets $\phi_{e,k} = w_k - \sum_{i=1}^{N} w_{e,k,i}$.
   (f) $\mathcal{P}$ chooses a random string $g_e \in \{0,1\}^\lambda$ and then it computes $\Omega_e = \mathtt{com}(\phi_{e,1} || \cdots || \phi_{e,n_{\mathtt{in}}}, g_e)$.
   (g) For each $i \in [N]$, $\mathcal{P}$ uses $\mathsf{sd}_{e,i}$ to generate $r_{e,i} \in \{0,1\}^\lambda$ and then it computes $\Gamma_{e,i} = \mathtt{com}(\mathsf{sd}_{e,i}, r_{e,i})$.
   (h) $\mathcal{P}$ uses sd_e to generate a random string $s_e \in \{0,1\}^\lambda$ and computes $\Gamma_e = \mathtt{com}(\mathsf{state}_e, s_e)$.
   (i) Finally, $\mathcal{P}$ computes $h_e = H(\Gamma_e || \Gamma_{e,1} || \cdots || \Gamma_{e,N})$.
2. $\mathcal{P}$ computes $h_\Gamma = H(h_1 || \cdots || h_M)$, $h_\Omega = H(\Omega_1 || \cdots || \Omega_M)$ and sends them to $\mathcal{V}$.

Round 2: $\mathcal{V}$ chooses a random challenge $E \subset [M]$ such that $|E| = \tau$ and sends it to $\mathcal{P}$.

Round 3:

1. Let $\bar{E} = [M] \setminus E$. First, $\mathcal{P}$ chooses $\mathsf{sd}_{\bar{E}}$.
2. For each $e \in \bar{E}$:
   (a) $\mathcal{P}$ initializes an empty string view_e. Then, it goes over the circuit in topological order and simulates each gate's computation using the MPC protocol described in Section 3.1, while consuming the random triples and squares it prepared in Round 1.
      i. For each multiplication gate g_k, $\mathcal{P}$ sets: $\mathsf{view}_e \leftarrow \mathsf{view}_e || \alpha_{e,k,1} || \cdots || \alpha_{e,k,N} || \beta_{e,k,1} || \cdots || \beta_{e,k,N}$.
      ii. For each square gate g_k, $\mathcal{P}$ sets: $\mathsf{view}_e \leftarrow \mathsf{view}_e || \alpha_{e,k,1} || \cdots || \alpha_{e,k,N}$.
   (b) Let $o_{e,1,i}, \dots, o_{e,n_{\mathtt{out}},i}$ be the shares on the output wires held by party P_i at the end of the computation. Then, for each output wire $k \in \mathcal{O}$, $\mathcal{P}$ sets: $\mathsf{view}_e \leftarrow \mathsf{view}_e || o_{e,k,1} || \cdots || o_{e,k,N}$.
   (c) $\mathcal{P}$ uses $\mathsf{sd}_{\bar{E}}$ to generate $g_e \in \{0,1\}^\lambda$ and computes $\Pi_e = \mathtt{com}(\mathsf{view}_e, g_e)$.
3. $\mathcal{P}$ computes $h_\pi = H(\Pi_{e_1} || \cdots || \Pi_{e_{|\bar{E}|}})$.
4. $\mathcal{P}$ sends to $\mathcal{V}$: $\{\mathsf{sd}_e\}_{e \in E}$, $\{\Omega_e\}_{e \in E}$ and h_π.

Fig. 1.a: HVZK argument using "Cut–and–Choose"

to Round 3, $\mathcal{S}$ generates view_e by following the instructions of $\mathcal{S}_\pi$ with $I = I_e$ and using $\mathsf{sd}_{e,i}$ to generate the required randomness. For the commitments $\Gamma_{e,\bar{i}_e}$, $\mathcal{S}$ uses the 0-string as the committed message.

4. For each $e \in \bar{E}$, for each $k \in \mathcal{O}$, $\mathcal{S}$ set $o_{e,k,i_e} = y_k - \sum_{i \in I_e} o_{e,i}$.
5. $\mathcal{S}$ computes all the hash values as an honest prover would do.
6. $\mathcal{S}$ outputs a transcript of the protocol.

The "Cut–and–Choose" Based HVZK Argument $\Pi_{\mathsf{c\&c}}$ (Part 2)

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

Round 5: For each $e \in \bar{E}$:
Let $I_e = [N] \setminus \{\bar{i}_e\}$. Then, $\mathcal{P}$ sends the following to $\mathcal{V}$: (1) $\mathsf{sd}_{\bar{E}}$, $\{\mathsf{sd}_{e,i}\}_{i \in I_e}$ (2) $\Gamma_{e,\bar{i}_e}$ (3) $\alpha_{e,k,\bar{i}_e}$, $\beta_{e,k,\bar{i}_e}$ and $\Delta_{e,k}$ for each multiplication or square gate g_k (4) g_e and $\{\phi_{e,k}\}_{k=1}^{n_{\mathsf{in}}}$ and (5) $o_{e,1,\bar{i}_e}, \ldots, o_{e,n_{\mathsf{out}},\bar{i}_e}$.

Output: $\mathcal{V}$ output acc iff all the following checks succeeds:

1. For each $e \in E$, $\mathcal{V}$ uses sd_e to compute h_e as a honest prover would do.
   For each $e \in \bar{E}$, $\mathcal{V}$ uses $\{\mathsf{sd}_{e,i}\}_{i \in I_e}$ to compute $\Gamma_{e,i}$ as a 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 $h_\Gamma = H(h_1 || \cdots || h_M)$.
2. For each $e \in \bar{E}$, $\mathcal{V}$ computes Ω_e using $\{\phi_{e,k}\}_{k=1}^{n_{\mathsf{in}}}$ and g_e. Then, using $\{\Omega_e\}_{e \in E}$ received from $\mathcal{P}$, the verifier $\mathcal{V}$ checks that $h_\Omega = H(\Omega_1 || \cdots || \Omega_M)$.
3. For each $e \in \bar{E}$, $\mathcal{V}$ computes view_e as an honest prover would do by going over the circuit in topological order and using $\{\mathsf{sd}_{e,i}\}_{i \in I_e}$, the shares $\alpha_{e,k,\bar{i}_e}, \beta_{e,k,\bar{i}_e}$ and $\Delta_{e,k}$ received from $\mathcal{P}$ for each multiplication and square gate, and $\{o_{e,k,\bar{i}_e}\}_{k=1}^{n_{\mathsf{out}}}$. Then, it computes Π_e as a honest prover would do.
   Finally, $\mathcal{V}$ checks that $h_\pi = H(\Pi_{e_1} || \cdots || \Pi_{e_{|\bar{E}|}})$.
4. For each $e \in \bar{E}$, for each $k \in \mathcal{O}$, $\mathcal{V}$ checks that $\sum_{i=1}^{N} o_{e,k,i} = y_k$

Fig. 1.b: HVZK Argument using "Cut–and–Choose", continued

From the indistinguishably of the transcript generated by $\mathcal{S}_\pi$ and the hiding property of the commitment scheme, it follows by an hybrid argument that the view generated by $\mathcal{S}$ is computationally indistinguishable from the view in a real execution.

Knowledge Soundness. We proceed to prove the soundness property of the protocol. For simplicity we assume that the commitment scheme is perfectly binding and that there are no collisions for the hash function.

We first argue that if the cheating probability $\delta(x)$ is higher than $\xi_{\mathsf{c\&c}}(M, N, \tau)$, then there exists at least one MPC instance (out of M) where the prover has committed to a valid witness $\boldsymbol{w}$. Recall that we consider deterministic provers and so the first prover's message where he commits to the input $\boldsymbol{w}$ is fixed. Now, let $\mathbf{G}$ be a 0/1-matrix where each column corresponds to a possible first challenge of $\mathcal{V}$ (i.e., τ pre-processings to be opened) and each row corresponds to a possible second challenge chosen by $\mathcal{V}$ (i.e., $M-\tau$ parties indices for which the view should *not* to be opened). Thus, $\delta(x)$ is the fraction of '1' entries in $\mathbf{G}$. Let $\xi_{\mathsf{c\&c}}(M, N, \tau) = \frac{\binom{M-c*}{\tau}}{\binom{M}{\tau}} \cdot N^{M-\tau-c*}$ (i.e., $c*$ is the value for which the expression for ξ written above is maximized). We can write $\xi_{\mathsf{c\&c}}$ also as $\frac{\binom{M-c*}{\tau} \cdot N^{c*}}{\binom{M}{\tau} \cdot N^{M-\tau}}$. Observe that the number of entries in $\mathbf{G}$ is $\binom{M}{\tau} \cdot N^{M-\tau}$. From our assumption that $\delta(x) > \frac{\binom{M-c*}{\tau} \cdot N^{c*}}{\binom{M}{\tau} \cdot N^{M-\tau}}$ it thus follows that the number of '1' entries in $\mathbf{G}$ is higher than $\binom{M-c*}{\tau} \cdot N^{c*}$. Next, assume that in the interaction with the prover $\mathcal{P}^*$, it corrupts c of the pre-processings. Clearly, if any of these are opened, then the transcript won't be accepted by $\mathcal{V}$. Thus, there can be '1' entries only in $\binom{M-c}{\tau}$ columns in $\mathbf{G}$. For each of these columns, there exists N^c possible challenges for the MPC instances where the pre-processing is incorrect. Since there are more than $\binom{M-c*}{\tau} \cdot N^{c*} \geq \binom{M-c}{\tau} \cdot N^c$ entries with '1' in $\mathbf{G}$, then there must exist two accepting transcripts with the same first challenge E and with different second challenge $\{i_e\}_{e \in \bar{E}}$ and $\{i'_e\}_{e \in \bar{E}}$, where $i_e \neq i'_e$ for an MPC instance e with *correct* pre-processing. This means that all the views of the parties in e are also correct. Thus, the witness used in this instance must be a valid witness.

Next, given that a valid witness $\boldsymbol{w}$ is used in execution e, observe that it is possible to extract this witness using two accepting transcripts $(E, \{i_e\}_{e\in\bar{E}}), (E', \{i'_e\}_{e\in\bar{E}'})$ such that the challenge for e is different. Specifically, it is required that one of the following will hold: $e \in E \cup E' \setminus E \cap E'$ or $i'_e \neq i_e$. This suffices since in the first case it is possible to extract the seeds of all parties from one transcript (where the pre-processing of e is opened) and the adjustment sent by the $\mathcal{P}$ from the second transcript (where e's pre-processing is not opened), there by obtaining the input shares of all parties. In the second case, where $i'_e \neq i_e$, one of the transcripts reveals $N-1$ input shares whereas the other reveals the remaining share, thereby again allowing us to compute the witness by adding all shares.

Let $\delta(x) = \xi_{\mathsf{c\&c}}(M, N, \tau) + \epsilon$ for some $\epsilon > 0$. We now describe an extractor $\mathcal{E}$ to obtain such two transcripts:

1. Probe the matrix $\mathbf{G}$ until the first '1' entry was found. Denote by $\boldsymbol{c} = (c_1, \ldots, c_M)$ be the challenge for this entry, where for each $e \in [M]$, c_e is the challenge for the ith execution.
2. For each execution e run an extractor $\mathcal{E}_e$, who probe $\mathbf{G}$ at random until an entry '1' is found for which the challenge $\boldsymbol{c}'$ is such that $c'_e \neq c_e$.
3. For each $\boldsymbol{c}'$ outputted by $\mathcal{E}_e$, extract the witness $\boldsymbol{w}$ used in execution e using $\boldsymbol{c}$ and $\boldsymbol{c}'$ (as explained in the text above), and check that $C(\boldsymbol{w}) = \boldsymbol{y}$. If yes, output $\boldsymbol{w}$ and halt.

First, observe that the expected running time of the first step is $\frac{1}{\delta} < \frac{1}{\epsilon}$. For the second step, we prove the following Lemma:

Lemma 1. *Let $J = \{j_{e_1}, j_{e_2}, \ldots\} \subseteq [M]$ be the set of indices which correspond to executions with valid witnesses and let $|J|$ denote its size. Then there exists an $e \in J$ such that $\Pr[\mathsf{acc} | c'_e \neq c_e] \geq \epsilon/M$, where acc is the event where the verifier accepts.*

Proof. We denote by Jeq the event $\forall e \in J : \ c'_e = c_e$ and by $\overline{\mathsf{Jeq}}$ its negation $\exists e \in J : \ c'_e \neq c_e$. Assume in contradiction that for *all* $e \in J$ it holds that $\Pr[\mathsf{acc}|c'_e \neq c_e] < \epsilon/M$.

It follows that

$$\begin{aligned}
\delta(x) &= \Pr[\mathsf{acc} \wedge \mathsf{Jeq}] + \Pr\left[\mathsf{acc} \wedge \overline{\mathsf{Jeq}}\right] = \Pr[\mathsf{acc} \wedge \mathsf{Jeq}] + \Pr\left[\mathsf{acc} \mid \overline{\mathsf{Jeq}}\right] \cdot \Pr\left[\overline{\mathsf{Jeq}}\right] \\
&\leq \Pr[\mathsf{acc} \wedge \mathsf{Jeq}] + \Pr\left[\mathsf{acc} \mid \overline{\mathsf{Jeq}}\right] \\
&= \Pr[\mathsf{acc} \wedge \mathsf{Jeq}] + \Pr\left[\mathsf{acc} \,\middle|\, c'_{e_{j_1}} \neq c_{e_{j_1}} \vee c'_{e_{j_2}} \neq c_{e_{j_2}} \vee \cdots \right] \\
&\leq \Pr[\mathsf{acc} \wedge \mathsf{Jeq}] + \sum_{e_j \in J} \Pr\left[\mathsf{acc} \,\middle|\, c'_{e_j} \neq c_{e_j}\right] \\
&< \Pr[\mathsf{acc} \wedge \mathsf{Jeq}] + \sum_{e_j \in J} \epsilon/M \\
&\leq \Pr[\mathsf{acc} \wedge \mathsf{Jeq}] + M \cdot \epsilon/M \\
&= \Pr[\mathsf{acc} \wedge \mathsf{Jeq}] + \epsilon
\end{aligned} \tag{1}$$

where the second inequality is obtained by using Union Bound, the third inequality follows from our assumption on $\Pr[\mathsf{acc}|c'_e \neq c_e]$ for all $e \in J$ and the last inequality holds since $|J| \leq M$.

We now proceed to bound $\Pr[\mathsf{acc} \wedge \mathsf{Jeq}]$, which is the probability of acceptance with all challenges for the executions with the valid witness remaining unchanged. In the following, we say that $c_e = 0$ if the challenge for e is to open the pre-processing and $c_e \neq 0$ otherwise. Now, without loss of generality, assume that for $e \in \{e_{j_1}, \ldots, e_{j_k}\}$, it holds that $c_e = 0$, whereas for $e \in \{e_{j_{k+1}}, \ldots, e_{j_{|J|}}\}$ it holds that $c_e \neq 0$ (i.e., exactly k of the executions with valid witness were chosen to be opened). It follows that

$$\Pr\left[c_{e_{j_1}} = \cdots = c_{e_{j_k}} = 0 \wedge c_{e_{j_{k+1}}} \neq 0 \wedge \cdots \wedge c_{e_{j_{|J|}}} \neq 0\right] = \frac{\binom{M-|J|}{\tau-k}}{\binom{M}{\tau}}$$

as well as

$$\Pr\left[\mathsf{Jeq} \;\middle|\; c_{e_{j_1}} = \cdots = c_{e_{j_k}} = 0 \wedge c_{e_{j_{k+1}}} \neq 0 \wedge \cdots \wedge c_{e_{j_{|J|}}} \neq 0\right] = \frac{\binom{M-|J|}{\tau-k}}{\binom{M}{\tau}} \cdot \frac{1}{N^{|J|-k}} \tag{2}$$

(recall that for executions $e_{j_{k+1}}, \ldots, e_{j_{|J|}}$ a second challenge is chosen with probability $1/N$).

Next, observe that once the challenges of the executions with correct witness are fixed and remain unchanged, we can compute the probability of obtaining a second acc using our formula of $\xi_{\mathsf{c\&c}}$. That is, conditioned on the event that k MPC preprocessings are opened and $|J| - k$ are not opened, it holds that

$$\begin{aligned}\Pr&\left[\mathsf{acc} \;\middle|\; \left(\mathsf{Jeq} \mid c_{e_{j_1}} = \cdots = c_{e_{j_k}} = 0 \wedge c_{e_{j_{k+1}}} \neq 0 \wedge \cdots \wedge c_{e_{j_{|J|}}} \neq 0\right)\right] \\ &\leq \frac{\binom{M-|J|-\bar{c}}{\tau-k}}{\binom{M-|J|}{\tau-k}} \cdot \frac{1}{N^{M-|J|-\tau+k-\bar{c}}}\end{aligned} \tag{3}$$

(note that here $\bar{c}$ is the number of corrupted pre-processings only within the executions with bad witness. This means that $\bar{c} \leq c$). This holds since if the probability was higher, then a valid witness must have been used in some of the executions. However, in (3), the distribution is only over executions with an *invalid* witness.

Combining (2) and (3) together, we conclude that

$$\begin{aligned}\Pr&\left[\mathsf{acc} \wedge \mathsf{Jeq} \;\middle|\; c_{e_{j_1}} = \cdots = c_{e_{j_k}} = 0 \wedge c_{e_{j_{k+1}}} \neq 0 \wedge \cdots \wedge c_{e_{j_{|J|}}} \neq 0\right] \\ &\leq \frac{\binom{M-|J|-\bar{c}}{\tau-k}}{\binom{M-|J|}{\tau-k}} \cdot \frac{1}{N^{M-|J|-\tau+k-\bar{c}}} \cdot \frac{\binom{M-|J|}{\tau-k}}{\binom{M}{\tau}} \cdot \frac{1}{N^{|J|-k}} \\ &= \frac{\binom{M-|J|-\bar{c}}{\tau-k}}{\binom{M}{\tau}} \cdot \frac{1}{N^{M-\tau-\bar{c}}} < \frac{\binom{M-c*}{\tau}}{\binom{M}{\tau}} \cdot \frac{1}{N^{M-\tau-c*}}\end{aligned}$$

where the last inequality holds since $\bar{c} \leq c$ (c is the number of overall corrupted pre-processings, where $\bar{c}$ is the derived by looking only on bad processings of the executions), $k + (c - \bar{c}) \leq |J|$ ($c - \bar{c}$ is the number of corrupted pre-processings for executions with good witness while k is the number of such executions that are opened and so their pre-procssing must be correct) and so $k + c \leq |J| + \bar{c}$ which means that

$$\binom{M - |J| - \bar{c}}{\tau - k} N^{\bar{c}} \leq \binom{M - k - c}{\tau - k} N^{c} \leq \binom{M - c}{\tau} N^{c} \leq \binom{M - c*}{\tau} N^{c*}.$$

Here, the second inequality follows because $\binom{x-1}{y-1} < \binom{x}{y}$ and the last step is due to the definition of $c*$.

Finally, we show that $\Pr[\mathsf{acc} \wedge \mathsf{Jeq}] \leq \frac{\binom{M-c*}{\tau}}{\binom{M}{\tau}} \cdot \frac{1}{N^{M-\tau-c*}}$. This easily holds since

$$\begin{aligned}\Pr[\mathsf{acc} \wedge \mathsf{Jeq}] &= \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (4)\\ &= \sum_{Q \subseteq J} \Bigg(\Pr[\mathsf{acc} \wedge \mathsf{Jeq} \mid \forall e \in Q : c_e = 0 \wedge \forall e \in J \setminus Q : c_e \neq 0] \cdot \\ &\quad \Pr[\forall e \in Q : c_e = 0 \wedge \forall e \in J \setminus Q : c_e \neq 0] \Bigg) \\ &\leq \frac{\binom{M-c*}{\tau}}{\binom{M}{\tau}} \cdot \frac{1}{N^{M-\tau-c*}} \sum_{Q \subseteq J} \Pr[\forall e \in Q : c_e = 0 \wedge \forall e \in J \setminus Q : c_e \neq 0] \\ &= \frac{\binom{M-c*}{\tau}}{\binom{M}{\tau}} \cdot \frac{1}{N^{M-\tau-c*}}.\end{aligned}$$

Here the last equality holds since $1 = \sum_{Q \subseteq J} \Pr[\forall e \in Q : c_e = 0 \wedge \forall e \in J \setminus Q : c_e \neq 0]$ which follows from the fact that the events whose probability we compute are all disjunct, but together they cover all possible challenges occurring for the elements in J.

Going back to (1), we have that $\delta < \epsilon + \frac{\binom{M-c*}{\tau}}{\binom{M}{\tau}} \cdot \frac{1}{N^{M-c*-\tau}}$ in contradiction to the assumption that $\delta = \xi(M, N, \tau) + \epsilon$. Thus, there must exists an execution e with a valid witness, for which $\Pr[\mathsf{acc} \mid c'_e \neq c_e] \geq \epsilon/M$. □

Recall that our extractor tries to extract the witness from all executions until it succeeds to extract a correct witness from some execution. From Lemma 1, there exists an execution e with a valid witness for which the probability of probing an accepting transcript that allows us to extract is higher than $\frac{\epsilon}{M}$. Thus, the expected number of steps until the witness is extracted is bounded by $\frac{M}{\epsilon}$. Note that M depends only on the statistical security parameter but its size in independent of the common input x held by the prover and the verifier (which is the circuit in our ZKPOK system). Thus, we conclude that if the success cheating probability is higher than $\xi_{\mathsf{c\&c}}(M, N, \tau)$, then a valid witness can be extracted in $O(\frac{|x|}{\epsilon})$ expected number of steps (recall that the extractor checks the validity of witnesses by running $C(\boldsymbol{w})$. Thus, the running time also depends on the common input x which is allowed by the definition). This concludes the proof. □

3.3 HVZKAoK Protocol Using Imperfect Preprocessing and Sacrificing

In this section, we present our second HVZKAoK protocol Π_{sac}. In this protocol, instead of ensuring that the preprocessings are correct, 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 compute $[\![\alpha]\!] = \epsilon [\![x]\!] + [\![a]\!]$ and $[\![\beta]\!] = [\![y]\!] + [\![b]\!]$.
3. The parties run $\mathsf{open}([\![\alpha]\!])$ and $\mathsf{open}([\![\beta]\!])$ to obtain α and β.
4. The parties locally compute $[\![v]\!] = \epsilon [\![z]\!] - [\![c]\!] + \alpha \cdot [\![b]\!] + \beta \cdot [\![a]\!] - \alpha \cdot \beta$.
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

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

In contrast, if one (or both) of the triples are incorrect, then the parties will accept with probability of at most $1/|\mathbb{F}|$ as shown in Lemma 2.

Lemma 2. *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 $\frac{1}{|\mathbb{F}|}$.*

Proof. Let $\Delta_z = z - x \cdot y$ and $\Delta_c = c - a \cdot b$. If the parties output acc then it means that $v = 0$, i.e.,

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

Next, consider the following cases:

– *Case 1: $\Delta_z = 0,\ \Delta_c \neq 0$.* In this case, $v = -\Delta_c \neq 0$. Thus, the parties will not output acc in contradiction to the assumption.
– *Case 2: $\Delta_z \neq 0,\ \Delta_c \neq 0$.* In this case, $v = 0$ iff $\epsilon = \Delta_c \cdot (\Delta_z)^{-1}$. Since ϵ is uniformly distributed over $\mathbb{F}$, this happens with probability $1/|\mathbb{F}|$.
– *Case 3: $\Delta_z \neq 0,\ \Delta_c = 0$.* In this case, $v = 0$ iff $\epsilon = 0$ which, yet again, happens with probability $1/|\mathbb{F}|$.

Going over all cases, we conclude that the lemma follows. □

Verification of a square pair using another. Similarly, one can use a random square ($[\![b]\!], [\![d]\!]$) to verify the correctness of a given square ($[\![x]\!], [\![z]\!]$) by working 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 proved in Lemma 3.

Lemma 3. *If ($[\![x]\!], [\![z]\!]$) or ($[\![b]\!], [\![d]\!]$) is an incorrect square, then the parties output* acc *in the sub-protocol above with probability $\frac{2}{|\mathbb{F}|}$.*

Proof. Let $\Delta_d = d - b^2$ and $\Delta_z = z - x^2$ and assume that the parties output acc. This means that

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

We consider the following three cases:

- *Case 1: $\Delta_z = 0,\ \Delta_d \neq 0$.* In this case, $v = 0$ iff $\epsilon^2 = 0 \mod |\mathbb{F}|$, which holds iff $\epsilon = 0$. Since ϵ is chosen randomly from $\mathbb{F}$, this holds with probability $\frac{1}{|\mathbb{F}|} < \frac{2}{|\mathbb{F}|}$.
- *Case 2: $\Delta_z \neq 0,\ \Delta_d \neq 0$.* In this case, $v = 0$ iff $\epsilon^2 = \Delta_z \cdot (\Delta_d)^{-1}$. Now, if $\epsilon = 0$, then $v = \Delta_z$ and the parties will reject. Otherwise, since there are $\frac{|\mathbb{F}|-1}{2}$ squares in $\mathbb{F}$, then the parties will accept with probability $\frac{2}{|\mathbb{F}|-1}$. Thus, overall the parties accept with probability $\frac{1}{|\mathbb{F}|} \cdot 0 + (1 - \frac{1}{|\mathbb{F}|}) \cdot \frac{2}{|\mathbb{F}|-1} = \frac{2}{|\mathbb{F}|}$.
- *Case 3: $\Delta_z \neq 0,\ \Delta_d = 0$.* In this case, $v = \Delta_z \neq 0$ and thus the parties will not output acc.

Going over all possible cases, we conclude that the lemma follows. □

The protocol. Our second PoK protocol is formally described in Fig. 2.a and Fig. 2.b. 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, the verifier $\mathcal{V}$ 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 $\mathcal{V}$, the prover $\mathcal{P}$ 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 his 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 the previous protocol, for values that are fixed, i.e., inputs, multiplications and squares, the prover 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 and the output of the circuit is $\boldsymbol{y}$.

Cheating probability (soundness). We compute the probability that $\mathcal{V}$ outputs acc when $C(\boldsymbol{w}) \neq \boldsymbol{y}$. Observe that each of the M executions is independent from the other executions. When considering a single execution, $\mathcal{P}$ can cheat in either computing the view of one of the parties or cheat in defining 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

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

Formal proof. We are now ready to prove formally that the protocol Π_{sac} is an honest verifier zero-knowledge argument of knowledge.

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

Proof. We prove that each of the three properties defined in Section 2.2 are satisfied by our protocol.

Completeness. This follows trivially from the correctness of the MPC protocol.

Honest Verifier Zero Knowledge. We construct a simulator $\mathcal{S}$ for our protocol which works as follows:

1. For $e = 1, \ldots, M$, $\mathcal{S}$ chooses random challenges $\epsilon_{e,k}$ for each multiplication and square gate g_k in C and random $\bar{i}_e \in [N]$.
2. $\mathcal{S}$ simulates the first step of the protocol: for each $e \in [M]$ and $i \in [N] \setminus \bar{i}_e$, it defines the shares of the random multiplication triples and squares and the shares on each wire of the circuit as an honest prover would do. For generating state_e, $\mathcal{S}$ chooses ransom $\Delta_{e,k}$ for each random triple/square gate and random $\varphi_{e,k}$ for each multiplication/square gate and random $\phi_{e,k}$ for each input wire k. Then, it computes $\mathsf{state}_{e,i}$ and $\Gamma_{e,i}$ as an honest verifier. For $\Gamma_{e,\bar{i}_r}$, it uses the 0-string as the committed message. Finally, $\mathcal{S}$ computes h_Γ as an honest prover.
3. $\mathcal{S}$ simulates Round 3:
   (a) First, it initializes an empty string view_e for all $e \in [M]$.

The "Sacrificing" Based HVZK Argument Π_{sac} (Part 1)

Let H be a collision-resistant hash function and $\mathtt{com}$ be a commitment scheme

Inputs: Both the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ hold $\boldsymbol{y} \in \mathbb{F}^{n_{\mathtt{out}}}$, a description of an arithmetic circuit C over a finite field $\mathbb{F}$ and parameters M, N; the prover $\mathcal{P}$ also hold a witness $\boldsymbol{w} \in \mathbb{F}^{n_{\mathtt{in}}}$ such that $C(\boldsymbol{w}) = \boldsymbol{y}$.

Round 1:

1. For each $e \in [M]$:
   (a) $\mathcal{P}$ initializes an empty string state_e.
   (b) For each $i \in [N]$, $\mathcal{P}$ initializes an empty string $\mathsf{state}_{e,i}$.
   (c) $\mathcal{P}$ chooses a master seed sd_e and seeds $\mathsf{sd}_{e,1}, \ldots, \mathsf{sd}_{e,N}$.
   Then, for each $i \in [N]$, it sets $\mathsf{state}_{e,i} \leftarrow \mathsf{sd}_{e,i}$.
   (d) $\mathcal{P}$ prepares the pre-processing data:
   – For each multiplication gate $g_k \in \mathcal{G}$:
   i. For each $i \in [N]$, $\mathcal{P}$ uses $\mathsf{sd}_{e,i}$ to generate $a_{e,k,i}, b_{e,k,i}, c_{e,k,i}$. This 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. $\mathcal{P}$ sets $\Delta_{e,k} = a_{e,k} \cdot b_{e,k} - c_{e,k}$ and $\mathsf{state}_e \leftarrow \mathsf{state}_e || \Delta_{e,k}$.
   iii. $\mathcal{P}$ defines the random triple for this gate 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 $i \in [N]$, $\mathcal{P}$ uses $\mathsf{sd}_{e,i}$ to generate $b_{e,k,i}$ and $d_{e,k,i}$. This 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. $\mathcal{P}$ sets $\Delta_{e,k} = (b_{e,k})^2 - d_{e,k}$ and $\mathsf{state}_e \leftarrow \mathsf{state}_e || \Delta_{e,k}$.
   iii. $\mathcal{P}$ defines the random square for this gate to be $([\![b_{e,k}]\!], [\![d_{e,k}]\!] + \Delta_{e,k})$.
   (e) $\mathcal{P}$ chooses a linear random sharing of the inputs:
   i. For each $i \in [N]$, $\mathcal{P}$ uses $\mathsf{sd}_{e,i}$ to generate $w_{e,1,i}, \ldots, w_{e,n_{\mathtt{in}},i}$.
   This shares define the random sharings $[\![w_{e,1}]\!], \ldots, [\![w_{e,n_{\mathtt{in}}}]\!]$, where $w_{e,k} = \sum_{i=1}^{N} w_{e,k,i}$.
   ii. For each input wire $k \in \mathcal{I}$, $\mathcal{P}$ sets $\phi_{e,k} = w_k - \sum_{i=1}^{N-1} w_{e,k,i}$ and $\mathsf{state}_e \leftarrow \mathsf{state}_e || \phi_{e,k}$.
   The sharing on this wire is defined to be $[\![w_{e,k}]\!] + \phi_{e,k}$.
   (f) $\mathcal{P}$ simulates the computation of the circuit C going gate-by-gate in topological order. In particular:
   – For each addition or multiplication-by-a-constant gates, $\mathcal{P}$ computes the parties' output shares via the local operation described in Section 3.1.
   – For each multiplication gate $g_k \in \mathcal{G}$ with sharing $[\![x_k]\!]$ and $[\![y_k]\!]$ on its input wires:
   i. For each $i \in [N]$, $\mathcal{P}$ uses $\mathsf{sd}_{e,i}$ 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. $\mathcal{P}$ sets: $\varphi_{e,k} = x_k \cdot y_k - \sum_{i=1}^{N} z_{e,k,i}$ and $\mathsf{state}_e \leftarrow \mathsf{state}_e || \varphi_{e,k}$.
   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 $i \in [N]$, $\mathcal{P}$ uses $\mathsf{sd}_{e,i}$ 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. $\mathcal{P}$ sets: $\varphi_{e,k} = (x_k)^2 - \sum_{i=1}^{N} z_{e,k,i}$ and $\mathsf{state}_e \leftarrow \mathsf{state}_e || \varphi_{e,k}$.
   The sharing on the output wire is defined to be $[\![z_{e,k}]\!] + \varphi_{e,k}$.
   (g) $\mathcal{P}$ uses sd_e to generate $r_e \in \{0,1\}^\lambda$ and computes $\Gamma_e = \mathtt{com}(\mathsf{state}_e, r_e)$.
   (h) For each $i \in [N]$, $\mathcal{P}$ uses $\mathsf{sd}_{e,i}$ to generate $r_{e,i} \in \{0,1\}^\lambda$ and then it computes $\Gamma_{e,i} = \mathtt{com}(\mathsf{state}_{e,i}, r_{e,i})$.
   (i) Finally, $\mathcal{P}$ computes $h_e = H(\Gamma_e || \Gamma_{e,1} || \cdots || \Gamma_{e,N})$.
2. $\mathcal{P}$ computes $h_\Gamma = H(h_1 || \cdots || h_M)$ and sends it to $\mathcal{V}$.

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

Fig. 2.a: HVZK Argument using "Sacrificing"

(b) For each $e \in [M]$, it chooses random $\alpha_{e,k,i}$ and $\beta_{e,k,i}$ for each multiplication and square gate g_k and $i \in [N]$ and adds them to view_e (note that here it chooses shares for all parties including party $\bar{i}_e$).

The "Sacrificing" Based HVZK Argument Π_{sac} (Part 2)

Round 3:
1. $\mathcal{P}$ chooses a random seed sd_E.
2. $\mathcal{P}$ uses sd_ι to generate random coefficients $\epsilon_{e,k}$ as the verifier would do.
3. For each $e \in [M]$:
   (a) $\mathcal{P}$ initializes an empty string view_e.
   (b) For each multiplication gate g_k (in topological order), $\mathcal{P}$ simulates the verification procedure described in the text using $\epsilon_{e,k}$. In addition, it sets: $\mathsf{view}_e \leftarrow \mathsf{view}_e||\alpha_{e,k,1}||\cdots||\alpha_{e,k,N}||\beta_{e,k,1}||\cdots||\beta_{e,k,N}$.
   (c) For each square gate g_k (in topological order), $\mathcal{P}$ simulates the verification procedure described in the text using $\epsilon_{e,k}$. In addition, it sets: $\mathsf{view}_e \leftarrow \mathsf{view}_e||\alpha_{e,k,1}||\cdots||\alpha_{e,k,N}$.
   (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 $i \in [N]$, $\mathcal{P}$ sets: $\mathsf{view}_e \leftarrow \mathsf{view}_e||v_{e,k,1}||\cdots||v_{e,k,N}$.
   (e) Let $o_{e,1,i}, \ldots, o_{e,n_{\mathtt{out}},i}$ be the shares on the output wires of C held by party P_i. Then, for output wire $k \in \mathcal{O}$, $\mathcal{P}$ sets: $\mathsf{view}_e \leftarrow \mathsf{view}_e||o_{e,k,1}||\cdots||o_{e,k,N}$.
4. $\mathcal{P}$ uses sd_E to generate $g_e \in \{0,1\}^\lambda$ and computes $\Pi_e = \mathtt{com}(\mathsf{view}_e, g_e)$.
5. $\mathcal{P}$ computes $h_\pi = H(\Pi_1||\cdots||\Pi_M)$ and sends it to $\mathcal{V}$.

Round 4: For each $e \in [M]$: $\mathcal{V}$ chooses a random $\bar{i}_e \in [N]$ and sends it 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 $\mathcal{V}$: 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 $(\Delta_{e,k}, \varphi_{e,k}, \alpha_{e,k,\bar{i}_e}, \beta_{e,k,\bar{i}_e}, v_{e,k,\bar{i}_e})$ 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: $\mathcal{V}$ output acc iff all the following checks succeeds:
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 $h_\Gamma = H(h_1||\cdots||h_M)$.
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 $h_\pi = H(\Pi_1||\cdots||\Pi_M)$.
3. For each $e \in [M]$ and multiplication/square gate g_k, $\mathcal{V}$ checks that $\sum_{i=1}^N v_{e,k,i} = 0$
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. 2.b: HVZK Argument using "Sacrificing", continued

(c) For each $e \in [M]$, multiplication/square gate g_k and $i \in [N] \setminus \{\bar{i}_e\}$, $\mathcal{S}$ computes $v_{e,k,i}$ as an honest prover. Then, it sets $v_{e,k,\bar{i}_e}$ such that $\sum_{i=1}^n v_{e,k,i} = 0$ and adds $v_{e,k,i}$ for all $i \in [N]$ to view_e.
(d) For each $e \in [M]$, $i \in [N] \setminus \{\bar{i}_e\}$ and $k \in [n_{\mathtt{out}}]$, $\mathcal{S}$ computes $o_{e,k,i}$ as an honest prover. Then, it sets $o_{e,k,\bar{i}_e}$ such that $\sum_{i=1}^n o_{e,k,i} = y_k$ and adds $o_{e,i}$ for all $i \in [N]$ to view_e.
(e) $\mathcal{S}$ computes $\{\Pi_e\}_{e \in [M]}$ and h_π as an honest prover would do.

4. $\mathcal{S}$ outputs the transcript of the protocol.

The only difference between the simulation and a real execution is the way the commitments to the shares of party $\bar{i}_e$ are computed and the way $\Delta_{e,k}, \phi_{e,k}, \varphi_{e,k}, \alpha_{e,k,i}$ and $\beta_{e,k,i}$ are chosen (in the simulation they are chosen uniformly whereas in the real execution they computed deterministically as the slack between random sharings and actual values that are on the wires). However, from the hiding property of the commitment the former does not change and since $\Delta_{e,k}, \phi_{e,k}, \varphi_{e,k}, \alpha_{e,k,i}$ and $\beta_{e,k,i}$ are all uniformly distributed over $\mathbb{F}$ in both executions, the latter does not change as well (since all of them are defined by a random sharing which is kept secret from the verifier). Therefore, we conclude that the transcript generated by $\mathcal{S}$ is indistinguishable from a real execution.

Knowledge Soundness. As in the proof of Theorem 2, we assume for simplicity that the commitment scheme is perfectly binding and that there are no collisions for the hash function. In addition, we consider the case where there are only multiplication gates in the circuit. The proof for the case when there are also square gates is similar with the appropriate changes. Recall that we denote by n_{mul} the number of multiplication gates in the circuit. Clearly, a cheating prover might not need to cheat in every multiplication gate in order to make the output correct, as this depends on the structure of the circuit. Thus, throughout the proof, we assume that the prover cheats in k multiplications and that this suffices to totally manipulate the output.

As the challenges of the different MPC instances are independent, we proceed by analyzing a single instance first (i.e. let $M = 1$, $\xi_{\mathsf{sac}}(1, N) = \xi_{\mathsf{sac}}(N)$) and then argue how this generalized to arbitrary choices of M. We begin by showing that if the cheating probability $\delta(x)$ is higher than $\xi_{\mathsf{sac}}(N)$, then the prover must have committed to the correct witness $\boldsymbol{w}$.

Lemma 4. *Let $M = 1$. If $\delta(x) > \xi_{\mathsf{sac}}(N)$, then the prover $\mathcal{P}^*$ must have committed to the correct witness $\boldsymbol{w}$ in Round 1.*

Proof. To see this, let $\mathbf{G}$ be a 0/1 matrix, where each column corresponds to a possible first challenge of the verifier, each row corresponds to a possible second challenge and the bit in each cell indicates whether the verifier outputs acc or not. It follows that there are $|\mathbb{F}|^{n_{mul}}$ columns and N rows in $\mathbf{G}$. Thus, if $\delta(x) > \frac{N+|\mathbb{F}|-1}{N\cdot|\mathbb{F}|}$, then the number of '1' entries in $\mathbf{G}$ is larger than $(N + \mathbb{F} - 1) \cdot |\mathbb{F}|^{n_{mul}-1}$. Now, assume for the sake of contradiction that the prover has cheated in $k > 0$ multiplication triples. By Lemma 2 there is one challenge for each of these gates for which the verifier won't detect cheating. With loss of generality we say that the challenges $\epsilon_1, \ldots, \epsilon_k$ are for the corrupted gates and challenges $\epsilon_{k+1}, \ldots, \epsilon_{n_{mul}}$ are for the remaining gates, and denote the k challenges, for which the verification procedure ends successfully, by $\bar{\epsilon}_1, \ldots, \bar{\epsilon}_k$. Thus, there are $|\mathbb{F}|^{n_{mul}-k}$ columns that can be filled with '1' entries (in each such column the challenges $\epsilon_1, \ldots, \epsilon_k$ are fixed to $\bar{\epsilon}_1, \ldots, \bar{\epsilon}_k$ and so it is required to choose a challenge only for the remaining $n_{mul} - k$ multiplication gates). For the remaining columns, we claim that there must exist at least one column with more than a single '1' entry. This holds since otherwise the number of '1' entries in $\mathbf{G}$ is bounded by $|\mathbb{F}|^{n_{mul}-k}\cdot N + (|\mathbb{F}|^{n_{mul}} - |\mathbb{F}|^{n_{mul}-k})\cdot 1 = (N + |\mathbb{F}|^k - 1)\cdot|\mathbb{F}|^{n_{mul}-k}$ (for $|\mathbb{F}|^{n_{mul}-k}$ columns all N entries are filled with '1' and for the other columns there is a single '1' entry). However, this is in contradiction to our assumption that there are more than $(N + |\mathbb{F}| - 1) \cdot |\mathbb{F}|^{n_{mul}-1}$ entries with '1' in $\mathbf{G}$, since

$$(N + |\mathbb{F}| - 1) \cdot |\mathbb{F}|^{n_{mul}-1} \geq (N + |\mathbb{F}|^k - 1) \cdot |\mathbb{F}|^{n_{mul}-k} \tag{5}$$

(to see that this inequality holds, observe that it is equivalent to $(N + |\mathbb{F}| - 1) \cdot |\mathbb{F}|^{k-1} \geq N + |\mathbb{F}|^k - 1$ which is equivalent to $N\cdot|\mathbb{F}|^{k-1} + |\mathbb{F}|^k - |\mathbb{F}|^{k-1} \geq N + |\mathbb{F}|^k - 1$ which can be written as $(N-1)\cdot|\mathbb{F}|^{k-1} \geq N - 1$ which holds for any $k > 0$). We conclude that there must be a column with challenges $\epsilon'_1, \ldots, \epsilon'_{n_{mul}}$ such that $\exists i \in [k] : \ \epsilon'_i \neq \bar{\epsilon}_i$, that has at least two '1' entries. That is, for the first challenge which corresponds to that column, the prover can answer successfully at least two different second challenges. Let c_1 and c_2 the challenges corresponding to such two accepting transcripts, i.e., both have the same first challenge $\epsilon'_1, \ldots, \epsilon'_{n_{mul}}$ and different second challenge i_1 and i_2. Note that it is possible to compute a witness from c_1 and c_2, since two different second challenges reveals the inputs of all the parties. Let $\boldsymbol{w}^*$ be the witness computed from c_1 and c_2. We argue that this is a valid witness, i.e., that $C(\boldsymbol{w}) = \boldsymbol{y}$. This holds since when the verifier accepts, $C(\boldsymbol{w}) \neq \boldsymbol{y}$ only if one of two events occur: (a) one of the parties' views is inconsistent but it is not chosen to be opened by $\mathcal{V}$ or (b) there are multiplication gates that were not correctly computed but $\mathcal{V}$ chooses the one single challenge that makes the verification procedure end successfully for each of these gates. However, the first event does not happen, since c_1 and c_2 differs in the second challenge, and thus all parties' views are covered and verified in the two executions. The second event does not happen as well for c_1 and c_2 since for both of them the first challenge has the property that $\exists i \in [k] : \ \epsilon'_i \neq \bar{\epsilon}_i$ (recall that $\bar{\epsilon}_i$ is the only challenge for the corrupted gate i that makes the verification procedure succeed). This contradicts our assumption on the number of multiplication triples which are incorrect. We conclude that $\boldsymbol{w}$ must be a valid witness and thus the claim follows. $\square$

In the protocol however we will have $M > 1$ MPC instances. We therefore define the set

$$S = \left\{ \tau = \begin{pmatrix} \epsilon_1^{(1)}, \ldots, \epsilon_{n_{mul}}^{(M)}, \\ i_1, \ldots, i_M \end{pmatrix} \in \mathbb{F}^{n_{mul} \cdot M} \times [N]^M \;\middle|\; \begin{matrix} \tau \text{ is challenge vector} \\ \text{of accepting transcript} \end{matrix} \right\}$$

where $\tau|_j = (\epsilon_1^{(j)}, \ldots, \epsilon_{n_{mul}}^{(j)}, i_j)$ denotes the challenges of MPC instance j of the challenge vector τ and $\tau|_j^P = i_j$ denotes the choice of party which is not opened in the j-th MPC protocol. Furthermore, define $S_1, \ldots, S_M$ where

$$S_j = \left\{ \left(\epsilon_1^{(j)}, \ldots, \epsilon_{n_{mul}}^{(j)}, i_j\right) \in \mathbb{F}^{n_{mul}} \times [N] \;\middle|\; \exists \tau \in S \wedge \tau|_j = \left(\epsilon_1^{(j)}, \ldots, \epsilon_{n_{mul}}^{(j)}, i_j\right) \right\}.$$

We start out with the fact that $\delta(x) > \xi_{\mathsf{sac}}(M, N)$ and so $|S| = \delta(x) \cdot (|\mathbb{F}|^{n_{mul}} \cdot N)^M > \xi_{\mathsf{sac}}(M, N) \cdot (|\mathbb{F}|^{n_{mul}} \cdot N)^M$, which follows from the definition of S. Now, assume that $\forall j \in [M] : \ |S_j| \leq \xi(N) \cdot (|\mathbb{F}|^{n_{mul}} \cdot N)$. Since $S = \{\tau \mid \forall j \in [M] : \ \tau|_j \in S_j\}$ it must hold that $|S| \leq |S_1| \cdots |S_M|$ because every τ must be a combination of different $\tau|_j$. But that implies

$$|S| \leq \xi_{\mathsf{sac}}(N)^M \cdot (|\mathbb{F}|^{n_{mul}} \cdot N)^M = \xi_{\mathsf{sac}}(M, N) \cdot (|\mathbb{F}|^{n_{mul}} \cdot N)^M$$

in contradiction to our assumption on the upper-bound of each $|S_j|$. Therefore, there must exist a $j \in [M]$ such that $|S_j| > \xi_{\mathsf{sac}}(N) \cdot (|\mathbb{F}|^{n_{mul}} \cdot N)$. From Lemma 4 it thus follows that this MPC instance must have the correct witness committed:

Corollary 1. *If $\delta(x) > \xi_{\mathsf{sac}}(M, N)$, then the prover $\mathcal{P}^*$ must have committed to the correct witness $\boldsymbol{w}$ in Round 1 in at least one of the MPC instances.*

Observe that once a correct witness is committed to in the first round in one of the MPC instances, then two accepting transcripts where the second challenge is different for such an instance are sufficient to extract it. We therefore can define an extractor $\mathcal{E}$ (similar to the one in the proof of the previous protocol) which works as follows:

1. Choose random vectors $\tau \in \mathbb{F}^{n_{mul} \cdot M} \times [N]^M$ until $\tau \in S$ was found.
2. For each $j \in [M]$ run the following in parallel:
   (a) Choose random vectors $\tau' \in \mathbb{F}^{n_{mul} \cdot M} \times [N]^M$ such that $\tau'|_j^P \neq \tau|_j^P$ until $\tau' \in S$ was found.
   (b) Extract $\boldsymbol{w}_j$ from MPC instance j using $\tau|_j, \tau'|_j$. If $C(\boldsymbol{w}) = \boldsymbol{y}$ then output $\boldsymbol{w}$ and stop, otherwise continue.

We now prove that the expected running time of our extractor satisfies the security definition.

Lemma 5. *Let $\delta(x) = \xi_{\mathsf{sac}}(M,N)+\varepsilon$. Then, the expected running time of extractor $\mathcal{E}$ is $O(M|x|/\varepsilon)$.*

Proof. The proof is very similar to the one of the previous protocol. We will find the first $\tau \in S$ in time $O(|x|/\varepsilon)$ by definition. Thus, let us argue that the overall expected runtime until we find a correct witness $\boldsymbol{w}$ is $M|x|/\varepsilon$ as in the previous proof.

By Corollary 1 there must be at least one MPC instance with a correct witness committed. Let $k \geq 1$ be the overall number of these correct committed witnesses, and without loss of generality they are in MPC instances $1,\ldots,k$. As argued above, we can extract if we find τ' such that $\tau'|_j^P \neq \tau|_j^P$ for $j \in [k]$. For the sake of contradiction, assume that $\Pr\left[\mathsf{acc} \mid \tau'|_j^P \neq \tau|_j^P\right] < \varepsilon/M$ for all $j \in [k]$. We can rewrite the success probability as

$$\begin{aligned}
\delta(x) = \Pr[\mathsf{acc}] =& \Pr\left[\mathsf{acc} \wedge \tau'|_1^P = \tau|_1^P \wedge \cdots \wedge \tau'|_k^P = \tau|_k^P\right] + \\
& \Pr\left[\mathsf{acc} \wedge \left(\tau'|_1^P \neq \tau|_1^P \vee \cdots \vee \tau'|_k^P \neq \tau|_k^P\right)\right] \\
\leq& \Pr\left[\mathsf{acc} \wedge \tau'|_1^P = \tau|_1^P \wedge \cdots \wedge \tau'|_k^P = \tau|_k^P\right] + \sum_{j=1}^{k} \Pr\left[\mathsf{acc} \wedge \tau'|_j^P \neq \tau|_j^P\right] \\
<& \Pr\left[\mathsf{acc} \wedge \tau'|_1^P = \tau|_1^P \wedge \cdots \wedge \tau'|_k^P = \tau|_k^P\right] + M \cdot (\varepsilon/M) \qquad (6) \\
=& \Pr\left[\mathsf{acc} \mid \tau'|_1^P = \tau|_1^P \wedge \cdots \wedge \tau'|_k^P = \tau|_k^P\right] \cdot \Pr\left[\tau'|_1^P = \tau|_1^P \wedge \cdots \wedge \tau'|_k^P = \tau|_k^P\right] + \varepsilon \\
\leq& \left(\frac{(N+|\mathbb{F}|-1)}{|\mathbb{F}|\cdot N}\right)^{M-k} \cdot \frac{1}{N^k} + \varepsilon \qquad (7) \\
\leq& \frac{(N+|\mathbb{F}|-1)^M}{|\mathbb{F}|^M \cdot N^M} + \varepsilon \qquad (8) \\
=& \xi_{\mathsf{sac}}(N,M) + \varepsilon = \delta(x).
\end{aligned}$$

Here Eq. (6) uses the assumed upper-bound on all $\Pr[\mathsf{acc} \mid \tau'|_j^P \neq \tau|_j^P]$ while Eq. (7) follows from the assumption that the $M-k$ instances have an incorrect witness and so by Lemma 4, the acceptance probability in each of them is bounded by $\xi_{\mathsf{sac}}(N)$ and since the probability that the second challenge in the other k instances remains the same is $\frac{1}{N^k}$. Finally, Eq. (8) follows since $\frac{(N+|\mathbb{F}|-1)^{M-k}}{(|\mathbb{F}|\cdot N)^{M-k}\cdot N^k} = \frac{(N+|\mathbb{F}|-1)^{M-k}}{|\mathbb{F}|^{M-k}\cdot N^M}$ and so $\frac{(N+|\mathbb{F}|-1)^{M-k}}{|\mathbb{F}|^{M-k}\cdot N^M} < \frac{(N+|\mathbb{F}|-1)^{M}}{|\mathbb{F}|^{M}\cdot N^M}$ since $|\mathbb{F}|^k < (N+|\mathbb{F}|-1)^k$, which holds because $N > 1$ and $k > 0$.

The resulting contradiction implies that there exists an instance j with valid witness such that $\Pr\left[\mathsf{acc} \mid \tau'|_j^P \neq \tau|_j^P\right] \geq \varepsilon/M$ which means that $\mathcal{E}$ will find the correct witness in expected runtime that is bounded by $O(M|x|/\varepsilon)$ as required (recall that $|x|$ in our protocol is the size of the circuit C). □

We have proved that if the cheating probability is higher then $\xi_{\mathsf{sac}}(M,N)$ by ε, then the prover must have committed to the correct witness and this can be extracted from him with expected running time that is bounded by $O(M|x|/\varepsilon)$. Note that M is independent from the common input and depends only on the statistical security parameter, and thus given a security parameter M can be viewed as a constant. This concludes the proof. □

3.4 Additional Optimizations

Reducing the number of seeds sent by the prover. In both of our protocols, 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. Specifically, let sd_e be the root of a binary tree of height $\log N$ where the seed in each internal node is used to generate the seeds of it's two descendants. The value of the ith leaf is labeled as $\mathsf{sd}_{e,i}$ and used to generate the randomness of party P_i.

Now, when the verifier chooses a random $\bar{i}_e \in [N]$ as its challenge, instead of sending him $\mathsf{sd}_{e,i}$ for each $i \in [N] \setminus \{\bar{i}_e\}$, it suffices to send just $\log N$ seeds. Specifically, $\mathcal{P}$ can iterate over the tree beginning with the root, and for each node j, where $\mathsf{sd}_{e,\bar{i}_e}$ is not in the induced sub-tree rooted at j, send the seed of j to the verifier. Once a seed has been chosen to be sent, $\mathcal{P}$ does not proceed to traverse in the sub-tree rooted in j but will descend into the other direction. Overall, in the first protocol, the communication induced by sending the seeds in $M-\tau$ emulations is then reduced from $(M-\tau)\cdot(N-1)$ seeds to $(M-\tau)\log N$, whereas in the second protocol it is reduced from $M\cdot(N-1)$ seeds to $M\log N$, which can be significant when the number of parties is large.

Batch verification in $\boldsymbol{\Pi}_{\mathsf{sac}}$. In Π_{sac} each multiplication triple is being verified separately. In order to save communication it is possible to batch-verify all triples by taking a linear combination of all $[\![v]\!]$s and open only the result. Specifically, given a batch of triples $([\![x_1]\!],[\![y_1]\!],[\![z_1]\!]),\ldots,([\![x_m]\!],[\![y_m]\!],[\![z_m]\!])$ to verify using a batch of random triples $([\![a_1]\!],[\![b_1]\!],[\![c_1]\!]),\ldots,([\![a_m]\!],[\![b_m]\!],[\![c_m]\!])$ (the same applies to squares), the parties first compute $[\![v_k]\!] = \epsilon_k \cdot [\![z_k]\!] - [\![c_k]\!] + \alpha_k \cdot [\![b_k]\!] + \beta.[\![a_k]\!] - \alpha_k \cdot \beta_k$ for each $k \in [m]$ (as in Lemma 2). Then, they jointly generate public random coefficients $\gamma_1,\ldots,\gamma_m \in \mathbb{F}$ and locally compute $[\![v]\!] = \sum_{i=1}^{m} \gamma_k \cdot [\![v_k]\!]$. Finally, the parties open $[\![v]\!]$ and check equality to 0. If $v_k = 0$ for all $k \in [m]$ then obviously $v = 0$ as well. In contrast, if there exists $k \in [m]$ such that $v_k \neq 0$, then $v = 0$ with probability $1/|\mathbb{F}|$. This is summed up in the following two propositions.

Proposition 1. *Let $([\![x_1]\!],[\![y_1]\!],[\![z_1]\!]),\ldots,([\![x_m]\!],[\![y_m]\!],[\![z_m]\!])$ and $([\![a_1]\!],[\![b_1]\!],[\![c_1]\!]),\ldots,([\![a_m]\!],[\![b_m]\!],[\![c_m]\!])$ be two lists of m triples. If there exists $\hat{k} \in [m]$ such that $([\![x_{\hat{k}}]\!],[\![y_{\hat{k}}]\!],[\![z_{\hat{k}}]\!])$ or $([\![a_{\hat{k}}]\!],[\![b_{\hat{k}}]\!],[\![c_{\hat{k}}]\!])$ is incorrect, then the parties output* acc *in the batch verification protocol with probability at most $\frac{2}{|\mathbb{F}|}$.*

Proof. If the parties output acc, this means that $v = \sum_{k=1}^{m} \gamma_k \cdot v_k = 0$. From Lemma 2 it follows that $v_{\hat{k}} = 0$ with probability $\frac{1}{|\mathbb{F}|}$ and so $v_{\hat{k}} \neq 0$ with probability $1 - \frac{1}{|\mathbb{F}|}$. In the latter case, $v = 0$ iff $\gamma_{\hat{k}} = (-\sum_{\substack{k=1\\k\neq\hat{k}}}^{m} \gamma_k \cdot v_k) \cdot (v_{\hat{k}})^{-1}$ which happens with probability $1/|\mathbb{F}|$ since $\gamma_{\hat{k}}$ is chosen uniformly from $\mathbb{F}$. Thus, we have that the overall probability that the parties output acc is bounded by $\frac{1}{|\mathbb{F}|} + (1 - \frac{1}{|\mathbb{F}|})\frac{1}{|\mathbb{F}|} < \frac{2}{|\mathbb{F}|}$ as required. □

Proposition 2. *Let $([\![x_1]\!],[\![z_1]\!]),\ldots,([\![x_m]\!],[\![z_m]\!])$ and $([\![b_1]\!],[\![d_1]\!]),\ldots,([\![b_m]\!],[\![d_m]\!])$ be two lists of m squares. If there exists $\hat{k} \in [m]$ such that $([\![x_{\hat{k}}]\!],[\![z_{\hat{k}}]\!])$ or $([\![b_{\hat{k}}]\!],[\![d_{\hat{k}}]\!])$ is incorrect, then the parties output* acc *in the batch verification protocol with probability of at most $\frac{3}{|\mathbb{F}|}$.*

Proof. From Lemma 3 it follows that $v_{\hat{k}} = 0$ with probability $\frac{2}{|\mathbb{F}|}$. Thus, the statement follows from exactly by the same argument as in the proof of Proposition 1. □

Plugging in the batch verification procedure, Π_{sac} is changed so that the random coefficients for the linear combination are chosen by the verifier and handed to the prover as and additional challenge. Specifically, in Round 2, the verifier picks a random coefficient $\gamma_{e,k}$ for each instance e and each multiplication/square gate, and hands it to the prover in addition to the random elements $\epsilon_{e,k}$ that are used inside the verification procedure. Then, in Round 3 the prover simulates the verification procedure for each multiplication/square gate and then simulates each party i locally taking the linear combination $v_{e,i} = \sum_k \gamma_{e,k} \cdot v_{e,k,i}$, where $v_{e,k,i}$ is its share of $v_{e,k}$. Finally, $\mathcal{P}$ sets $\mathsf{view}_e \leftarrow \mathsf{view}_e || v_{e,1} || \cdots || v_{e,N}$. This significantly reduces the communication of the protocol, since once the verifier chooses the one party $\bar{i}_e$ whose view is not opened in execution e, the prover does not need to send the share of party $\bar{i}_e$ for each $v_{k,e}$, but rather only its share of v_e, since only one single value is opened and checked for the entire circuit. On the other hand, observe that using this optimization also affects the soundness of the protocol, as the probability of not being caught in the verification procedure is now increased to $\frac{2}{|\mathbb{F}|}$ for cheating in multiplication gates and $\frac{3}{|\mathbb{F}|}$ for square gates (observe that once again manipulating the output of square gates would be more beneficial for the prover). Thus, if there are square gates in the circuit, the updated cheating probability is bounded by

$$\xi_{\mathsf{sac}}(M,N) = \left(\frac{3}{|\mathbb{F}|} + \left(1 - \frac{3}{|\mathbb{F}|}\right) \cdot \frac{1}{N}\right) = \left(\frac{3N + |\mathbb{F}| - 3}{|\mathbb{F}| \cdot N}\right)$$

whereas if there are multiplication gates in the circuit (and no square gates), the updated cheating probability is bounded by

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

Batching the output correctness check. Similarly to this previous optimization, 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 before sending it to $\mathcal{V}$. Recall that in both $\Pi_{\mathsf{c\&c}}, \Pi_{\mathsf{sac}}$, $\mathcal{V}$ checks for each output wire $k \in [n_{\mathsf{out}}]$ in each execution e, that the shares $\{o_{e,k,i}\}_{i=1}^N$ add up to the output y_k that should be on that wire. This is the same as checking that $\sum_{i=1}^N o_{e,k,i} - y_k = 0$. Thus, we can take a random linear combination $\sum_{k \in [n_{\mathsf{out}}]} \gamma_{e,k} \cdot (\sum_{i=1}^N o_{e,k,i} - y_k)$ with values $\gamma_{e,k}$ chosen by $\mathcal{V}$ and check that the result equals 0. This reduces communication, because now the prover is required to send the verifier only one single output share of party $\bar{i}_e$ instead of a share per each output wire (recall that for the other parties their entire view is revealed and thus the share on the output wires can be computed by the verifier).

In order to plug this idea into $\Pi_{\mathsf{c\&c}}$ while maintaining security, we need to add another round where, after the view during the circuit computation is committed, the verifier chooses randomly the random coefficients $\gamma_{e,k}$ for each execution e and output wire k and hands them to the prover who then computes for each party i the random linear combination of its shares. Specifically, for each party i with shares $\{o_{e,k,i}\}_{k \in [n_{\mathsf{out}}]}$, it computes $o_{e,i} = \sum_{k \in [n_{\mathsf{out}}]} \gamma_{e,k} \cdot o_{e,k,i}$, and then commits to $o_{e,1} || \cdots || o_{e,N}$ and sends the commitment to the verifier. Only then, the verifier chooses the party $\bar{i}_e$, whose view is kept secret and hands it to the prover. Then, in the final round, the prover sends all the data specified in the protocol's description with the exception being that instead of sending $\{o_{e,k,\bar{i}_e}\}_{k \in [n_{\mathsf{out}}]}$ for each execution e, it suffices to send $o_{e,\bar{i}_e}$ only. The verifier then computes $o_{e,i}$ for each $i \in [N] \setminus \{\bar{i}_e\}$, and using $o_{e,\bar{i}_e}$ checks that $\sum_{i=1}^N o_{e,i} - \sum_{k \in [n_{\mathsf{out}}]} \gamma_{e,k} \cdot y_k = 0$ for each execution e.

For Π_{sac} no additional rounds are required. After the prover $\mathcal{P}$ has committed in the first round to the shares of the parties on all the wires of the circuit, $\mathcal{V}$ can send the random coefficients along with the challenges it sends for the verification procedure. The prover $\mathcal{P}$ then computes the linear combination of the output shares and commits to the obtained shares as explained above.

As in the previous optimization, we need to update the soundness of the two protocols, since taking a random linear combination can result with having 0, even though the sharing on the output wires were not correct (this happens with probability $\frac{1}{|\mathbb{F}|}$ as in the previous optimization). In $\Pi_{\mathsf{c\&c}}$ we denote by c_1 the number of pre-processings corrupted by the prover and by c_2 the number of emulations where the prover cheats in computing the view of one of the parties. In the remaining executions, the prover do not cheat (but only uses an invalid witness), hoping that the output verification will succeed due to the random linear combination. Then, we have that the cheating probability is

$$\xi_{\mathsf{c\&c}}(M,N,\tau) = \max_{\substack{0\le c_1\le M-\tau \\ 0\le c_2\le M-\tau-c_1}} \left\{ \frac{\binom{M-c_1}{\tau}}{\binom{M}{\tau}\cdot N^{c_2}\cdot |\mathbb{F}|^{M-\tau-c_1-c_2}} \right\}.$$

We remark that in all our instantiations, it always hold that $|\mathbb{F}| > N$ and thus the best strategy for a cheating prover is to set $c_2 = M - \tau - c_1$, which means that the cheating probability remains the same as before.

For Π_{sac}, we argue that the cheating probability ξ_{sac} remains the same. This holds since the current optimization is independent of the verification of multiplications/squares process, meaning that both generate different outputs which the verifier checks in the last round. In particular, when the prover cheats in one of the multiplications/squares, it can get away with it only by receiving the "correct" challenge for the verification process or by changing one view, exactly as before. Thus, for each instance, the prover can either manipulate the output of multiplication/square gates or act honestly with an invalid witness and hope that taking the random linear combination of the incorrect outputs will equal to the random linear combination of the publicly known outputs. Therefore, the cheating probability for each instance is bounded by $\max\left\{\frac{2N+|\mathbb{F}|-2}{|\mathbb{F}|\cdot N}, \frac{1}{|\mathbb{F}|}\right\}$ (in the case where the circuit consists of multiplication gates and no square gates; the analysis is similar for the opposite case). However, observe that $\frac{2N+|\mathbb{F}|-2}{|\mathbb{F}|\cdot N} > \frac{1}{|\mathbb{F}|}$ (since $N + |\mathbb{F}| - 2 > 0$), and thus the soundness remains the same as before.

To sum-up the discussion, this optimization does not increase the soundness error of our protocols. However, the number of rounds is increased in the first protocol.

3.5 Communication and Computation Cost Analysis

In this section, we estimate the cost of our two protocols. The analysis includes all three optimizations described in the previous section. We denote by $|\mathsf{hash}|$, $|\mathsf{sd}|$ and $|\mathsf{com}|$ the length of the hash values, seeds and commitments.

Computation 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 over those of the additions over $\mathbb{F}$ 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.

Communication cost for $\Pi_{\mathsf{c\&c}}$. The communication cost of messages sent from $\mathcal{P}$ to $\mathcal{V}$ in each round is: (1) Round 1: $2|\mathsf{hash}|$; (2) Round 3: $\tau \cdot |\mathsf{sd}| + (M - \tau) \cdot |\mathtt{com}| + |\mathsf{hash}|$; (3) Round 5: $|\mathsf{sd}| + (M - \tau) \cdot (\log N \cdot |\mathsf{sd}| + |\mathtt{com}| + 3\log_2(|\mathbb{F}|) \cdot n_{mul} + 2\log_2(|\mathbb{F}|) \cdot n_{sq} + \log_2(|\mathbb{F}|) \cdot n_{\mathtt{in}} + \log_2(|\mathbb{F}|))$.

Summing the above, we obtain that the overall communication cost incurred by messages sent by $\mathcal{P}$ is

$$|\mathsf{hash}| \cdot 3 + |\mathsf{sd}| \cdot (\tau + 1 + \log N(M - \tau)) + |\mathtt{com}| \cdot 2(M - \tau) + \log_2(|\mathbb{F}|) \cdot (M - \tau)(3n_{mul} + 2n_{sq} + n_{\mathtt{in}} + 1).$$

Communication cost for Π_{sac}. The communication cost of messages sent from $\mathcal{P}$ to $\mathcal{V}$ in each round is: (1) Round 1: $|\mathsf{hash}|$; (2) Round 3: $|\mathsf{hash}|$; (3) Round 5: $|\mathsf{sd}| + M \cdot (|\mathsf{sd}| + \log N \cdot |\mathsf{sd}| + |\mathtt{com}| + 4\log_2(|\mathbb{F}|) \cdot n_{mul} + 3\log_2(|\mathbb{F}|) \cdot n_{sq} + \log_2(|\mathbb{F}|) + \log_2(|\mathbb{F}|) \cdot n_{\mathtt{in}} + \log_2(|\mathbb{F}|)$.

We obtain that the overall cost of communication sent from $\mathcal{P}$'s side is

$$|\mathsf{hash}| \cdot 2 + |\mathsf{sd}| \cdot (2 + M \log N) + |\mathtt{com}| \cdot M + \log_2(|\mathbb{F}|) \cdot M(4n_{mul} + 3n_{sq} + n_{\mathtt{in}} + 2)$$

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 $O(\log(\lambda) \cdot \kappa \cdot (|C|/\log(N)))$.

One might be tempted to draw conclusions from the above expressions regarding which of the protocols is more communication-efficient. However, recall that the soundness error of the protocols is not the same. This means that different values will be chosen for the parameters M and N in each of the protocols, thereby affecting the proof size in different ways. In fact, as we will see in Section 6, the soundness error of Π_{sac} allows having smaller parameters for M, N (e.g., for the same N in both protocols, a smaller M can be taken for Π_{sac}), thus reducing the overall communication cost.

4 Sampling Circuits on the Fly

At the end of the previous section we introduced an optimization where the verifier $\mathcal{V}$ checks the circuit's 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 the random coefficients which will be used to compute the linear combination *after* the prover $\mathcal{P}$ commits to the view of the parties during the circuit's computation. 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 Section 5. Thus, we take a pause in this section to formally establish this idea, so that security of all optimizations of this kind can be derived easily without the need to re-prove security each time.

Although in the above optimization of linear combination the verifier chooses solely the circuit that will be evaluated, we consider a more general definition where both the prover and the verifier sample the circuit together from a set of possible circuits. The sampling process must begin only after the prover have committed and fixed the witness 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 to the circuit which will later be used. 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. This makes it possible to still apply the Fiat-Shamir transform in order to make the protocol non-interactive. The prover $\mathcal{P}$, 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 not allow him to break the soundness of the protocol or the zero-knowledge property.

The section is organized as follows. We first provide a formal definition for the notion of circuit sampling. Then, we show how to incorporate it into our argument system and argue that the obtained system remains a zero-knowledge argument of knowledge. Finally, we show how the output linear combination optimization described above is indeed an instantiation of the general notion.

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). For $(x, w) \in R$ we say that* $\mathsf{Sample} = (\mathsf{ExtWitness}, \mathsf{Response}, \mathsf{SampCircuit})$ *is an R-circuit sampler if*

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

We next define a security game which follows the way we embed these algorithms into our argument system. Consider the following game, which we denote by $\mathsf{Game}_{R,\mathcal{P}}((x, w), S_{\mathcal{P}}, S_{\mathcal{V}}, \lambda)$, executed with a prover $\mathcal{P}$:

1. $\mathcal{P}$ outputs $\overline{\boldsymbol{w}}$.
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. Use the R-circuit sampling algorithm to compute $(C, Y) \leftarrow \mathsf{SampCircuit}(x, \tau_{\mathcal{P}}, \tau_{\mathcal{V}})$.
5. Output 1 iff $C(\overline{\boldsymbol{w}}) \in Y$.

To understand the game, observe that Step 1 emulates the commitment to the witness, made by $\mathcal{P}$ in the first step of our argument systems, 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 the $\mathsf{SampCircuit}$ algorithm is gathered, (C, Y) are being determined, and $\mathcal{P}$ wins if computing the circuit C on $\overline{\boldsymbol{w}}$ yields a valid output. Note that in the above definition there is no validation ensuring that the message $\tau_{\mathcal{P}}$ that is sent in the game is valid. This can be done by $\mathsf{SampCircuit}$ outputting $Y = \emptyset$ for an *invalid* choice of $\tau_{\mathcal{P}}$.

We have three requirements from the circuit sampler. First, an obvious requirement is that if the prover 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. We say that* Sample *is correct, if when $\mathcal{P}$ on input (x, w) computes $\overline{\boldsymbol{w}} \leftarrow \mathsf{ExtWitness}(x, w)$ and $\tau_{\mathcal{P}} \leftarrow \mathsf{Response}(x, w, \overline{\boldsymbol{w}}, \tau_{\mathcal{V}})$, it holds that $\mathsf{Game}_{R,\mathcal{P}}((x, w), S_{\mathcal{P}}, S_{\mathcal{V}}, \lambda) = 1$ with probability negligibly close to 1.*

The second property required from the circuit sampler is soundness. Similarly to the standard definition of this notion, we require that if the prover wins in the above game with probability larger than α, then it will be possible to extract the correct witness.

Definition 5 (α-sound R-circuit Sampler). *Let* Sample *be an R-circuit sampler. We say that* Sample *is α-sound if given* $\Pr[\mathsf{Game}_{R,\mathcal{P}}((x,w),S_{\mathcal{P}},S_{\mathcal{V}},\lambda)=1]>\alpha$ *(where the distribution is over $\tau_{\mathcal{V}}\in S_{\mathcal{V}}$), there exists a deterministic PPT extractor $\mathcal{E}(\overline{\boldsymbol{w}})$ which outputs w' such that $(x,w')\in R$ with probability* 1.

While the definition looks similar to that of knowledge soundness used for proofs of knowledge, there are crucial differences: the extraction is from $\overline{\boldsymbol{w}}$, and it is done in polynomial time and with probability 1. This is because extraction of a candidate witness w' from $\overline{\boldsymbol{w}}$ is an "easy" task (as we will see in all our circuit sampling uses) and so the only question is whether w' is a valid witness 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 $\overline{\boldsymbol{w}}$

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 $\overline{\boldsymbol{w}}$ to an outsider. However, we additionally need simulatability of $C(\overline{\boldsymbol{w}})$: the sampled circuit may enforce the relation R in different ways than a static circuit would do (thus the set Y) and this could potentially leak information. We will see an occurrence of this phenomenon in one of the optimizations which we present later, where we use rejection sampling inside the circuit.

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

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

where $\mathcal{P}$ assume to act honestly as in Definition 4.

4.2 Circuit Sampling and the Zero-Knowledge Argument.

We now include the above approach into the protocol Π_{sac} due to the simplicity of the analysis and leave an adaptation of the first protocol as future work.

The modified protocol $\Pi_{\mathsf{sac}}^{\mathrm{samp}}$ works as follows, where we only highlight the additional steps:

Round 1: For each $e\in[M]$ (i.e. each MPC instance) $\mathcal{P}$ computes $\overline{\boldsymbol{w}}_e\leftarrow\mathsf{ExtWitness}(x,w)$. Then, it chooses the randomness used for the execution e (i.e., the seeds used to derive all randomness). Finally, $\mathcal{P}$ commits to the extended witness and the randomness and send 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: For each $e\in[M]$ $\mathcal{P}$ locally computes $\tau_{\mathcal{P},e}\leftarrow\mathsf{Response}(x,w,\overline{\boldsymbol{w}}_e,\tau_{\mathcal{V},e})$ as well as $(C_e,Y_e)\leftarrow\mathsf{SampCircuit}(x,\tau_{\mathcal{P},e},\tau_{\mathcal{V},e})$. It uses C_e in for MPC protocol instance e and sends the remaining first round messages together with $\tau_{\mathcal{P},e}$ to $\mathcal{V}$.

Round 4: $\mathcal{V}$ runs round 2 as in the regular protocol.

Round 5: $\mathcal{P}$ runs round 3 as in the regular protocol.

Round 6: $\mathcal{V}$ runs round 4 as in the regular protocol.

Round 7: $\mathcal{P}$ runs round 5 as in the regular protocol.

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

We next prove that protocol $\Pi_{\mathsf{sac}}^{\mathrm{samp}}$ is an honest verifier zero-knowledge argument of knowledge.

Lemma 6. *Let R be an NP-relation and* Sample *be a correct, α-sound and simulatable R-circuit sampler. Then $\Pi_{\mathsf{sac}}^{samp}$ is an honest verifier zero-knowledge argument of knowledge that is statistically complete for the relation R with knowledge error (soundness) $(\alpha + (1-\alpha)\cdot \xi_{\mathsf{sac}}(N))^M$.*

Recall that $\xi_{\mathsf{sac}}(N) = \frac{|\mathbb{F}|+N-1}{|\mathbb{F}|\cdot N}$ and so the soundness error of modified protocol, according to the lemma, is $\left(\alpha + (1-\alpha)\cdot \frac{|\mathbb{F}|+N-1}{|\mathbb{F}|\cdot N}\right)^M$.

To prove the lemma, the main change here compared to the proof from Section 3 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. Completeness and the zero-knowledge property, on the other hand, follow directly from the previous security argument.

Proof. We prove the three properties required by the definition in Section 2.

Completeness. $\Pi_{\mathsf{sac}}^{\mathrm{samp}}$ is complete, as Sample is a correct R-circuit sampler. We allowed Sample some slack to occasionally abort, but this will only happen with negligibly small probability.

Honest Verifier Zero-Knowledge. We can construct a simulator $\mathcal{S}'$ for the new protocol as follows: first, $\mathcal{S}'$ samples all messages of $\mathcal{V}$ as in the protocol, together with all the $\tau_{\mathcal{V},e}$ which it chooses honestly. Next, $\mathcal{S}'$ samples $(\tau_{\mathcal{P},e}, C_e(\overline{\boldsymbol{w}}_e))$ using $\mathcal{S}$ from Definition 6 and additionally generates $C_e \leftarrow \mathsf{SampCircuit}(x, \tau_{\mathcal{V},e}, \tau_{\mathcal{P},e})$ for each $e \in [M]$. Then, run the simulator of the original argument of knowledge-scheme, where we hard-wire the outputs $C_e(\overline{\boldsymbol{w}})$ from $\mathcal{S}$ into the individual MPC instances.

The Zero-Knowledge property now follows by a hybrid argument: we let $\mathcal{H}_0$ be the distribution of transcripts of the protocol. In the first hybrid $\mathcal{H}_1$, we use the simulator of the overall argument of knowledge of Section 3 but compute $(\tau_{\mathcal{P},e}, C_e(\boldsymbol{w}_e)) \leftarrow \mathsf{Response}(x, w, \boldsymbol{w}_e, \tau_{\mathcal{V},e})$ as in the protocol. By the proofs from the preceding section $\mathcal{H}_0 \approx_s \mathcal{H}_1$. Next, we define a sequence of hybrids $\mathcal{H}_2^i$ for $i = 0, \ldots, M$. In hybrid $\mathcal{H}_2^i$ we do the same as in $\mathcal{H}_1$, but replace $(\tau_{\mathcal{P},e}, C_e(\boldsymbol{w}_e))$ in the first i instances by the output of $\mathcal{S}$ from Definition 6.

By definition, we then have $\mathcal{H}_1 = \mathcal{H}_2^0$. For each $i \in [M]$ we have $\mathcal{H}_2^{i-1} \approx_s \mathcal{H}_2^i$ by the definition of $\mathcal{S}$ as we only change one value. Finally, observe that $\mathcal{S}'$ outputs the same distribution as $\mathcal{H}_2^M$ and thus the zero-knowledge property follows.

Knowledge Soundness. Since our proof here is only an adaptation of the proof of Theorem 3, we only highlight the changes that must be inserted to it.

The proof of Theorem 3 consists of three steps: (i) Show that for $M = 1$ any $\delta(x) > \xi_{\mathsf{sac}}(N)$ implies that a correct witness must have been committed. (ii) Show that this generalizes to $M > 1$ executions, so that $\delta(x) > \xi_{\mathsf{sac}}(M, N)$ means that at least one of the M instances has a correct witness committed. (iii) Show that by sending different challenges, it is possible to extract the witness from at least one of the MPC instances with a correct witness.

The first step, where $M = 1$ follows from the proof of Lemma 4 and the soundness of the sampler. Specifically, assume that the success probability $\delta(x)$ of the prover is higher than $\alpha + (1-\alpha)\cdot\xi_{\mathsf{sac}}(N)$ and assume that it didn't commit to the correct witness $\boldsymbol{w}$ (and so the extended witness $\overline{\boldsymbol{w}}$ is also incorrect). Let K be the set of circuits that might be returned by $\mathsf{SampCircuit}$ for a fixed x and $\overline{\boldsymbol{w}}$. Then we have

$$\begin{aligned}\Pr[\mathsf{acc}] &= \Pr[\mathsf{acc} \mid C(\overline{\boldsymbol{w}}) \in Y]\cdot\Pr[C(\overline{\boldsymbol{w}}) \in Y] + \Pr[\mathsf{acc} \mid C(\overline{\boldsymbol{w}}) \notin Y]\cdot\Pr[C(\overline{\boldsymbol{w}}) \notin Y] \\ &= \alpha + (1-\alpha)\cdot\xi_{\mathsf{sac}}(N)\end{aligned}$$

where the distribution is over K and over the challenges of the verifier. This contradicts the assumption that $\delta(x) = \Pr[\mathsf{acc}] > \alpha + (1-\alpha)\cdot\xi_{\mathsf{sac}}(N)$ and thus the prover must have committed to the correct witness.

Next, the above can be easily generalized to any $M > 1$. That is, if we have $M > 1$ executions, then $\delta(x) > (\alpha + (1-\alpha)\cdot\xi_{\mathsf{sac}}(N))^M$ implies that there is at least one execution where the prover have committed to a correct witness. This follows from the same argument as in the proof of Theorem 3 which led to Corollary 1, with the only difference being that here the space of possible changes is larger and includes $|K|$.

Going into the third step of the proof, we can define the extractor $\mathcal{E}$ in the same way as in the proof of Theorem 3. Informally speaking, $\mathcal{E}$ first probe challenges at random till a first accepting transcript is found. Then, $\mathcal{E}$ runs M processes in parallel where process j runs until a second accepting transcript has been found where the second challenge *in execution* j is different from the challenge in the first accepting transcript. Holding two accepting transcripts where the second challenge is different is sufficient for extracting the committed witness (as $\mathcal{E}$ has now the input shares of all parties), and so if the correct witness was found by one of the processes, then $\mathcal{E}$ halts (recall that once a correct extended witness $\overline{\boldsymbol{w}}$ is found then by the soundness of the sampler, $\boldsymbol{w}$ is computed in polynomial time).

Assume that $\delta(x) = \Pr[\mathsf{acc}] = (\alpha + (1-\alpha)\cdot\xi_{\mathsf{sac}}(N))^M + \varepsilon$ for some $\varepsilon > 0$. Then, the first step requires $1/\delta(x) < 1/\varepsilon$ expected number of steps. For the second step, we show that there exists an execution $j \in [M]$ with the correct witness for which each attempt in the second step of $\mathcal{E}$ succeeds with probability $> \varepsilon/M$, implying that the correct witness will be found in the second step within expected M/ε number of steps.

Assume without loss of generality that the first k executions are the executions with the correct witness and assume in contradiction that the probability of finding the desired second accepting transcript in each of them is $\leq \varepsilon/M$. Then, using exactly the same argument as in Lemma 5 (see Eq. 6 and 7) we have

$$\delta(x) = \Pr\left[\mathsf{acc}\right] \leq (\alpha + (1-\alpha)\cdot\xi_{\mathsf{sac}}(N))^{M-k}\cdot\frac{1}{N^k} + \varepsilon.$$

To complete the proof, we therefore need to show that

$$(\alpha + (1-\alpha)\cdot\xi_{\mathsf{sac}}(N))^{M-k}\cdot\frac{1}{N^k} < (\alpha + (1-\alpha)\cdot\xi_{\mathsf{sac}}(N))^{M}.$$

This is equivalent to showing that

$$1 < (\alpha + (1-\alpha)\cdot\xi_{\mathsf{sac}}(N))^{k}\cdot N^k$$

which holds if

$$1 < (\alpha + (1-\alpha)\cdot\xi_{\mathsf{sac}}(N))\cdot N = \alpha\cdot N + (1-\alpha)\cdot\xi_{\mathsf{sac}}(N)\cdot N.$$

Recall that $\xi_{\mathsf{sac}}(N) = \frac{N+|\mathbb{F}|-1}{N\cdot|\mathbb{F}|}$, and so we need to show that

$$1 < \alpha\cdot N + (1-\alpha)\cdot\frac{N+|\mathbb{F}|-1}{|\mathbb{F}|}.$$

Observe that $1 < \frac{N+|\mathbb{F}|-1}{|\mathbb{F}|} < N$ (where the latter follows since $|\mathbb{F}| > 1$ and $N > 1$). Thus, it follows that

$$1 < \frac{N+|\mathbb{F}|-1}{|\mathbb{F}|} = \alpha\cdot\frac{N+|\mathbb{F}|-1}{|\mathbb{F}|} + (1-\alpha)\cdot\frac{N+|\mathbb{F}|-1}{|\mathbb{F}|} < \alpha\cdot N + (1-\alpha)\cdot\frac{N+|\mathbb{F}|-1}{|\mathbb{F}|}$$

which is exactly what we wanted to show.

We conclude that $\delta(x) = \Pr[\mathsf{acc}] < (\alpha + (1-\alpha)\cdot\xi_{\mathsf{sac}}(N))^M + \varepsilon$ in contradiction to our assumption. This means that our extractor can find the correct witness within expected number of $O(M|x|/\epsilon)$ steps, exactly as in the proof of Theorem 3. This concludes the proof. □

4.3 Batched Output Correctness Check as a Circuit Sampler

We now revisit the batching of the output verification from Section 3.4 and consider it in the context of circuit sampling. Recall that in this optimization, the verifier chooses random coefficients that are used to compute the linear combination of the outputs, so that only one value is eventually opened and checked instead of checking the correctness for each output wire of the original circuit.

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

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

SampCircuit: On input $\tau_{\mathcal{V}} = (\boldsymbol{\gamma}) \in \mathbb{F}^{n_{\mathtt{out}}}$ output the circuit C' which performs the following:
1. Compute $\boldsymbol{y}' = C(\overline{\boldsymbol{w}})$ where $\boldsymbol{y}' \in \mathbb{F}^{n}_{\mathtt{out}}$.
2. Compute $y_1 = \sum_{i=1}^{n_{\mathtt{out}}} \boldsymbol{\gamma}[i]\cdot(\boldsymbol{y}'[i] - \boldsymbol{y}[i])$.
3. Output y_1.

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

Response: Output 1.

Fig. 3: Batching the Output Check as a Circuit Sampler

We first define the three algorithms of the circuit sampler for this optimization: ExtWitness receives $((C, \boldsymbol{y}), \boldsymbol{w})$ as an input and returns the extended witness $\overline{\boldsymbol{w}}$, which in this case is just $\boldsymbol{w}$. Response receives as an input the tuple $((C, \boldsymbol{y}), \boldsymbol{w}, \overline{\boldsymbol{w}}, \tau_{\mathcal{V}})$, but note that in this optimization, the verifier's challenge $\tau_{\mathcal{V}}$ fully defines the circuit and thus the output of Response is just 1. Finally, SampCircuit receives $((C, \boldsymbol{y}), \tau_{\mathcal{V}}, \tau_{\mathcal{P}})$ 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 $\boldsymbol{y}'[k]$ by the expected public value $\boldsymbol{y}[k]$; (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 - 0. We summarize the construction in Fig. 3.

The three algorithms defined above satisfies 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 $\frac{1}{|\mathbb{F}|}$ due to the fact that the random coefficients are uniformly chosen (see Lemma 2). Finally, Simulation of the sampler is also trivial as here both $\tau_{\mathcal{P}}$ and Y are deterministic and known in advance.

A remark about the soundness of the method. The reader might have observed that there is a disagreement between Section 3.4 and Section 4.2 regarding the soundness of the argument system when using the batched output correctness check. Specifically, in Section 3.4 we argued that the soundness of Π_{sac} remains $\xi_{\mathsf{sac}}(N, M)$ as when the optimization is not used (more precisely, the soundness is $\left(\max\left\{\xi_{\mathsf{sac}}(N), \frac{1}{|\mathbb{F}|}\right\}\right)^M$ which equals to $\xi_{\mathsf{sac}}(N, M) = (\xi_{\mathsf{sac}}(N))^M$ since $\xi_{\mathsf{sac}}(N) > \frac{1}{|\mathbb{F}|}$), whereas Lemma 6 from Section 4.2 implies that the soundness is $\left(\frac{1}{|\mathbb{F}|} + (1 - \frac{1}{|\mathbb{F}|})\xi_{\mathsf{sac}}(N)\right)^M$.

To understand the difference, note that the way the optimization is incorporated into the protocols in each of the sections is not the same. In this section, we incorporate the optimization through the general framework of circuit sampling. This means that the MPC protocol is being simulated by the prover only after the circuit was sampled and so when computing multiplication/square gates, the cheating prover knows whether $C(\boldsymbol{w}) = \boldsymbol{y}$. Thus, it knows whether cheating in the computation of these gates is required or not. Therefore, in each of the M executions, with probability $\frac{1}{|\mathbb{F}|}$ cheating is successful in the circuit sampling step, and with probability $1 - \frac{1}{|\mathbb{F}|}$ the prover will have to manipulate the MPC protocol simulation which succeeds with probability $\xi_{\mathsf{sac}}(N)$. However, looking into the optimization (and not just plugging it into the general framework of circuit sampling), observe that it only adds layers on top of the initial circuit and that it does not add any multiplication nor square gate to the circuit. Thus, we can ask the prover to simulate the computation of all multiplication and square gate already in the first step before the additional layers were sampled. In this case, the prover needs to decide in each of the M executions whether to manipulate the MPC computation or not in the first step. Thus, a cheating prover who wishes to maximize its success probability will choose to cheat in the MPC simulation only if the probability to succeed is higher than the probability that a random linear combination of incorrect outputs will yield 0. This is exactly the way the optimized protocol is described in Section 3.4 and hence the "different" soundness error. We remark that in the other circuit sampling optimizations we will see in the next section, the circuit that is sampled contains also square gates, and thus the prover can delay its decision whether to cheat in the MPC simulation to after the circuit is known, which means that the obtained soundness is indeed as indicated by Lemma 6.

5 Proving Knowledge of SIS Instances

The protocols from Section 3 are *asymptotically* less communication-efficient than previous argument systems such as [AHIV17,BBC+18]. However, they have advantages when the circuit size is not too big and 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. The approach can be simply generalized to secrets from a larger interval, but only at the expense of 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 argument system we present, we will mention what is the resulted size of the proof, based on the analysis of $\Pi_{\mathsf{sac}}^{\mathrm{samp}}$, which is the same as that of Π_{sac} from Section 3.5.

5.1 The Baseline Proof for SIS

We start by presenting a argument for the Binary SIS problem presented in Section 2.4. The reason behind that is because general range proofs are hard using a circuit over $\mathbb{F}_q$ 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.

Let us first recap the definition of the relation that we aim to prove, which was given in Section 2.4 as

$$R^{m,n,q}_{\mathtt{B-SIS}} = \{(x, w) = ((\mathbf{A}, \boldsymbol{t}), \boldsymbol{s}) \mid \boldsymbol{s} \in \{0,1\}^m \wedge \mathbf{A} \in \mathbb{F}_q^{n\times m} \wedge \boldsymbol{t} = \mathbf{A}\boldsymbol{s}\}$$

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

The obtained circuit is described in Fig. 4. For ease of notation we let $a_{i,j} \in \mathbb{F}_q$ 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 Section 3, with $\mathcal{V}$ testing that the circuit's output $\hat{\boldsymbol{y}}$ equals $(\boldsymbol{t}[1], \ldots, \boldsymbol{t}[n], 0, \cdots, 0)$. 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.

Witness: $\boldsymbol{w} = (\boldsymbol{s}[1], \ldots, \boldsymbol{s}[m]) \in \mathbb{F}_q^m$

Computation:
1. $\forall i \in [m]$ compute $r_i \leftarrow \boldsymbol{s}[i]^2$
2. $\forall j \in [n]$ compute $y_j \leftarrow \sum_{i\in[m]} a_{j,i}\boldsymbol{s}[i]$
3. $\forall i \in [m]$ compute $y_{i+n} \leftarrow r_i - \boldsymbol{s}[i]$

Output: $\hat{\boldsymbol{y}} \leftarrow (y_1, \ldots, y_{m+n})$

Fig. 4: An Arithmetic circuit representation of $R^{m,n,q}_{\mathtt{B-SIS}}$; The circuit contains m square gates, has m inputs and $m+n$ outputs.

Using the cost analysis from Section 3.5 (for the scarifying-based protocol), We conclude that the total number of bits communicated by $\mathcal{P}$ is

$$|\mathsf{hash}| \cdot 2 + |\mathsf{sd}| \cdot (2 + M \log N) + |\mathtt{com}| \cdot M + \log q \cdot M(4m + 2)$$

Generalizing the Distribution of the input $\boldsymbol{s}$. It is immediate to extend the construction from Fig. 4 to other input distributions. We first generalize it to secrets $\boldsymbol{s}$ such that $||\boldsymbol{s}||_\infty \leq 1$. Again, we want to mainly use squaring instead of multiplication gates. Since $\boldsymbol{s}[i] \in \{-1, 0, 1\} \iff \boldsymbol{s}[i]^3 - \boldsymbol{s}[i] = 0$ we could implement this test using one squaring and one multiplication gate. To further optimize this, observe that the polynomial $X^4 - X^2$ has the same roots as $X^3 - X$ (albeit with different multiplicity) but can be computed using two squaring gates instead (since $X^4 - X^2 = (X^2)^2 - X^2$). Thus, in this setting, $\mathcal{P}$'s total communication is

$$|\mathsf{hash}| \cdot 2 + |\mathsf{sd}| \cdot (2 + M \log N) + |\mathtt{com}| \cdot M + \log q \cdot M(7m + 2)$$

More generally, if we want to prove that $0 \leq \boldsymbol{s}[i] < 2^r$ for some fixed $r \in \mathbb{N}$, the most direct way is to again prove that $\boldsymbol{s}[i]$ is the root of the polynomial $f(X) = \Pi_{j=0}^{2^r-1}(X - j)$. This will increase the argument size by a factor of 2^r compared with the Binary SIS argument, but for the aforementioned interval one can reduce this to $O(r)$: a number $\boldsymbol{s}[i]$ is in the interval $[0, 2^r - 1]$ if and only if there exists $s_{i,1}, \ldots, s_{i,r}$ such that $\boldsymbol{s}[i] = \sum_{j=0}^{r-1} s_{i,j+1} 2^j$ and $s_{i,j} \in \{0, 1\}$. This means that the interval check can be done by providing the bit decomposition $s_{i,1}, \ldots, s_{i,r}$ of each $\boldsymbol{s}[i]$, testing if these $s_{i,j}$ are indeed bits (using one square gate per test) and then reconstructing $\boldsymbol{s}[i]$ on the fly. The circuit construction is summarized in Fig. 5.

Witness: $\boldsymbol{w} = (s_{1,1}, \ldots, s_{1,r}, \ldots, s_{m,1}, \ldots, s_{m,r}) \in \mathbb{F}_q^{m \cdot r}$.

Computation:
1. $\forall i \in [m], j \in [r]$ compute $r_{i,j} \leftarrow s_{i,j}^2$
2. $\forall i \in [m]$ compute $\boldsymbol{s}[i] \leftarrow \sum_{j=0}^{r-1} s_{i,j+1} 2^j$
3. $\forall j \in [n]$ compute $y_j \leftarrow \sum_{i \in [m]} a_{j,i} \boldsymbol{s}[i]$
4. $\forall i \in [m], j \in [r]$ compute $y_{i,j} \leftarrow r_{i,j} - s_{i,j}$

Output: $\hat{\boldsymbol{y}} \leftarrow (y_1, \ldots, y_n, y_{1,1}, \cdots, y_{m,r})$

Fig. 5: An Arithmetic circuit representation of $R_{\mathtt{SIS}}^{m,n,q}$ where each $\boldsymbol{s}[i]$ is in the interval $[0, 2^r)$; The circuit contains $m \cdot r$ square gates, has $m \cdot r$ inputs and $n + m \cdot r$ outputs.

The witness w is valid if $\hat{\boldsymbol{y}}$ equals $(\boldsymbol{t}[1], \ldots, \boldsymbol{t}[n], 0, \cdots, 0)$. While the witness is now expanded by a factor r, we in total only have to compute $m \cdot r$ square gates. All other operations are linear and do not influence the argument size, which in total is

$$|\mathsf{hash}| \cdot 2 + |\mathsf{sd}| \cdot (2 + M \log N) + |\mathtt{com}| \cdot M + \log q \cdot M(4m \cdot r + 2)$$

Proving Knowledge of general SIS instances. Finally, we aim at constructing an argument of knowledge for SIS as defined in Definition 1, for which we need $\boldsymbol{s}[i] \in [-\beta, \beta]$ or alternatively, that $((\mathbf{A}, \boldsymbol{y}), \boldsymbol{s}) \in R_{\mathtt{SIS}}^{m,n,q,\beta}$. Instead of proving this exact bound, we show that $((\mathbf{A}, \boldsymbol{y}), \boldsymbol{s}) \in R_{\mathtt{SIS}}^{m,n,q,2\beta}$ using the circuit representation from Fig. 5. Therefore, consider the following algorithm:

1. $\mathcal{P}, \mathcal{V}$ set $\hat{\boldsymbol{t}} = \boldsymbol{t} + \mathbf{A}\boldsymbol{\beta}$ and $\mathcal{P}$ additionally sets $\hat{\boldsymbol{s}} = \boldsymbol{s} + \boldsymbol{\beta}$.
2. Choose the smallest $r \in \mathbb{N}$ such that $\beta \leq 2^r - 1$.
3. $\mathcal{P}, \mathcal{V}$ run one of our two protocols on the circuit from Fig. 5 using the interval $[0, 2^{r+1} - 1]$ and with the common output being $(\hat{\boldsymbol{t}}[1], \ldots, \hat{\boldsymbol{t}}[n], 0, \ldots, 0)$.

Then, the above algorithm is a zero-knowledge argument of knowledge for the SIS relation with slack 2 as long as $2\beta < \frac{q-1}{2}$. To see this, we only have to look at correctness and soundness as the zero-knowledge property follows trivially. For correctness, the chosen r will always lead to $\hat{\boldsymbol{s}}$ being in the right interval since $2\beta \leq 2(2^r - 1) < 2^{r+1} - 1$. And due to the bound on β and q, adding and subtracting β will not cause a wrap-around mod$q$. For soundness, it is immediate that every extracted $\hat{\boldsymbol{s}}'$ with $\hat{\boldsymbol{t}} = \mathbf{A}\hat{\boldsymbol{s}}'$ can be turned into a witness $\boldsymbol{s}'$ for $\boldsymbol{t}$ by setting $\boldsymbol{s}' = \hat{\boldsymbol{s}}' - \boldsymbol{\beta}$. In this case, it holds for each coefficient $\boldsymbol{s}'[i]$ that

$$\begin{aligned}\boldsymbol{s}'[i] &\in [-\beta, 2^{r+1} - 1 - \beta] \\ &\in [-\beta, 2^{r+1} - 2^r] \\ &\in [-\beta, 2^r]\end{aligned}$$

Note that in the worst case, β is a power of 2 which means that we must set r such that $2^r - 1 = 2\beta - 1$. But then $\boldsymbol{s}'[i] \in [-\beta, 2\beta]$ and the claimed slack follows.

Thus, 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. $\mathcal{P}$ sends

$$|\mathsf{hash}| \cdot 2 + |\mathsf{sd}| \cdot (2 + M \log N) + |\mathtt{com}| \cdot M + \log q \cdot M(4m \cdot (\lfloor \log_2(\beta) \rfloor + 2) + 2)$$

5.2 Reducing Verification Time

Our two protocols from Section 3 are almost symmetric in the amount of computation that $\mathcal{P}, \mathcal{V}$ have to perform. Increasing the number of parties N in either protocol leads to a reduced number of instances M for which one has to simulate the inner MPC protocol. At the same time, this increases the number of transcripts of parties which $\mathcal{V}$ has to check, namely $M \cdot (N-1)$ in total. As we do not assume any restrictions on $\mathbf{A}$ of the SIS-instance (such as using Ring- or Module-SIS), the multiplication with $\mathbf{A}$ on the side of $\mathcal{V}$ turns out to be a bottleneck, since it has to multiply the matrix with $N-1$ input shares (corresponding to the $N-1$ parties for whom the view is tested). We now describe how to counteract this problem by batching the linear tests together across multiple parties and MPC instances.

Recall that in the protocol (as presented in Section 3) the verifier $\mathcal{V}$ recomputes the circuit from the inputs to the outputs in every MPC instance. Therefore, to verify the linear relation as in the protocol, $\mathcal{V}$ recomputes the output share $o_{e,j,i}$ for each execution $e \in [M]$, output wire $j \in [n_{\mathtt{out}}]$, and party $i \in I_e$ from the input shares $w_{e,1,i}, \ldots, w_{e,n_{\mathtt{in}},i}$ of each party by computing $o_{e,j,i} = \sum_{k \in [n_{\mathtt{in}}]} a_{j,k} w_{e,k,i}$. These output wire shares $o_{e,j,i}$ are then compared to the committed output shares in the commitment Π_e which $\mathcal{V}$ obtained from $\mathcal{P}$ as part of the argument.

One observation is that this computation of $o_{e,j,i}$ is linear in all the values that depend on the simulated party i as well as on the MPC instance. Thus instead of verifying each $o_{e,j,i}$ individually, we can check this with lower computational overhead by testing a linear combination across all parties and MPC instances instead. For this, we use the following standard argument:

Proposition 3. *Let $\mathbf{A} \in \mathbb{F}^{\phi \times \tau}$ and consider the following game with adversary $\mathcal{A}$:*

1. *$\mathcal{A}$ outputs $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_\rho \in \mathbb{F}^\tau$ as well as $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_\rho \in \mathbb{F}^\phi$.*
2. *Random $\lambda_1, \ldots, \lambda_\rho \in \mathbb{F}$ are chosen.*
3. *If $\exists j \in [\rho] :\ \boldsymbol{o}_j \neq \mathbf{A}\boldsymbol{w}_j$ and $\sum_{j \in [\rho]} \lambda_j \cdot \boldsymbol{o}_j = \mathbf{A} \left(\sum_{j \in [\rho]} \lambda_j \cdot \boldsymbol{w}_j \right)$ then the output of the game is 1 and and 0 otherwise.*

Then, the probability that the game's output is 1 *is at most* $1/|\mathbb{F}|$.

Proof. Assume that $\mathcal{A}$ wins (i.e., the output of the game is 1). Thus, there exists $\bar{j} \in [\rho]$ such that $\boldsymbol{o}_j = \mathbf{A}\boldsymbol{w}_j + \boldsymbol{\Delta}_j$, and $\boldsymbol{\Delta}_j$ is non-zero. It follows that

$$\sum_{j\in[\rho]} \lambda_j \cdot \boldsymbol{o}_j = \sum\nolimits_{j\in[\rho]} \lambda_j \cdot (\mathbf{A}\boldsymbol{w}_j + \boldsymbol{\Delta}_j) = \sum\nolimits_{j\in[\rho]} \lambda_j \cdot \mathbf{A}\boldsymbol{w}_j + \sum\nolimits_{j\in[\rho]} \lambda_j \cdot \boldsymbol{\Delta}_j$$
$$= \mathbf{A}(\sum\nolimits_{j\in[\rho]} \lambda_j \cdot \boldsymbol{w}_j) + \sum\nolimits_{j\in[\rho]\setminus\{\bar{j}\}} \lambda_j \cdot \boldsymbol{\Delta}_j + \lambda_{\bar{j}} \boldsymbol{\Delta}_{\bar{j}}.$$

Now, since $\mathcal{A}$ wins, this means that $\sum_{j\in[\rho]\setminus\{\bar{j}\}} \lambda_j \cdot \boldsymbol{\Delta}_j + \lambda_{\bar{j}} \boldsymbol{\Delta}_{\bar{j}} = 0$. As all the λ_j are chosen uniformly at random and $\boldsymbol{\Delta}_{\bar{j}} \neq \mathbf{0}$, this happens only with probability $1/|\mathbb{F}|$. □

Using the above Proposition, we can reduce verification time as follows. The verifier $\mathcal{V}$ will now first sample uniformly random values $\lambda_1, \ldots, \lambda_{M(N-1)}$ locally. Next, instead of computing the $M(N-1)$ matrix products with $\mathbf{A}$ ($N-1$ in each MPC instance e), it will compute a weighted sum of the $M(N-1)$ vectors first and then compute a single matrix product with $\mathbf{A}$. While this approach has soundness error $1/|\mathbb{F}|$, $\mathcal{V}$ can simply repeat it with different choices of λ to reduce the error probability.

The drawback of this approach is that it comes at the cost of increasing the proof size. In particular, according to the description of our protocols in Section 3, $\mathcal{P}$ only needs to send the output shares of one party (the shares $o_{e,j,\bar{i}_e}$ of party $P_{\bar{i}_e}$ whose view is not opened and so cannot be locally computed by the verifier), whereas the above optimization requires $\mathcal{P}$ to send the output shares of all parties to $\mathcal{V}$. Thus, this optimization to the computation time increases the argument size by $|\log q| \cdot M \cdot (N-1) \cdot n$ bits.

5.3 Reducing Communication by 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. 4, we defined a circuit for this problem that has m square gate. 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 as defined in Section 4, where only $\mathcal{V}$ has a challenge and so only $\mathcal{V}$ is actually sampling the circuit.

Assume that we want to check if m input sharings $\boldsymbol{s}[1], \ldots, \boldsymbol{s}[m]$ indeed are bits, and furthermore 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 $\forall i \in [m] : \ D(i) = \boldsymbol{s}[i]$. 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 $\forall i \in [m] : \ \boldsymbol{s}[i] \in \{0,1\}$ 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 $\boldsymbol{s}[i]$ 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 $\boldsymbol{s}[i]$). 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 Fig. 6 we summarize the above intuition. Given h points $T = \{(x_i, y_i)\}_{i\in[h]}$ we define the Lagrange coefficients $\ell_j^T(x) = \prod\limits_{\substack{1\le i\le h\\ i\ne j}} \frac{x-x_i}{x_j-x_i}$ for polynomial evaluation. In addition to Lagrange interpolation we also use the random linear combination of the outputs optimization described in the previous section.

Let $D(x)$ be a $(m-1)$–degree polynomial such that $\forall i \in [m] : D(i) = \boldsymbol{s}[i]$ and $B(x) = (D(x))^2$ be a $(2m-2)$–degree polynomial. Let $S_D = \{(i, D(i))\}_{i\in[m]}$ and $S_B = S_D \cup \{(i, B(i))\}_{i\in[2m-1]\setminus[m]}$ be the evaluation points of the polynomials D, B. We set $S_{\mathcal{V}} = \mathbb{F}^{n+1}$.

ExtWitness: On input

$$(x = (\boldsymbol{A}, \boldsymbol{t}[1], \dots, \boldsymbol{t}[n]), w = (\boldsymbol{s}[1], \dots, \boldsymbol{s}[m])) \in R^{m,n,q}_{\texttt{B-SIS}}$$

set $\overline{\boldsymbol{w}} = (\boldsymbol{s}[1], \dots, \boldsymbol{s}[m], b_1, \dots, b_{m-1}) \in \mathbb{F}_q^{2m-1}$ where $b_i = B(i+m) = D(i+m)^2$.

SampCircuit: On input $\tau_{\mathcal{V}} = (\boldsymbol{\gamma}, x) \in \mathbb{F}^n \times \mathbb{F}$ output the circuit C which performs the following:

1. $\forall j \in [n]$ compute $\overline{y}_j \leftarrow \sum_{i\in[m]} a_{j,i}\boldsymbol{s}[i]$
2. Set $y_1 \leftarrow \sum_{j\in[n]} \boldsymbol{\gamma}[j] \cdot (\overline{y}_j - \boldsymbol{t}[j])$
3. $z_1 \leftarrow \sum_{j=1}^{m} \ell_j^{S_D}(x) \cdot \boldsymbol{s}[j]$
4. $z_2 \leftarrow \sum_{j=1}^{m} \ell_j^{S_B}(x) \cdot \boldsymbol{s}[j] + \sum_{j=1}^{m-1} \ell_{j+m}^{S_B}(x) \cdot b_j$
5. $z_3 \leftarrow z_1^2$
6. $y_2 \leftarrow z_3 - z_2$
7. Output of C is (y_1, y_2)

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

Response: Output 1.

Fig. 6: Sampling of a circuit for $R^{m,n,q}_{\texttt{B-SIS}}$. This circuit contains 1 squaring gate, has $2m-1$ inputs and 2 outputs.

We now prove that the algorithm defined in Fig. 6 is a circuit sampler as defined in Section 4.1.

Theorem 4. *The algorithms in Fig. 6 yield a perfectly correct, $\frac{2m-2}{|\mathbb{F}|}$-sound and perfectly simulatable circuit sampler for the relation $R^{m,n,q}_{\texttt{B-SIS}}$.*

Proof. Correctness and the simulation property follow directly from the definition of the above algorithms. What remains to show is the $\frac{2m-2}{|\mathbb{F}|}$-soundness.

Fix a string $\overline{\boldsymbol{w}}$ and assume that soundness is higher than $\frac{2m-2}{|\mathbb{F}|}$. As $\Pr[y_1 = 0] > \frac{2m-2}{|\mathbb{F}|} > \frac{1}{|\mathbb{F}|}$ for all $m > 1$, this implies that $\overline{\boldsymbol{w}}[1], \dots, \overline{\boldsymbol{w}}[m]$ map to $\boldsymbol{t}$ under multiplication with $\mathbf{A}$ by the same argument as in Section 4.3. Consider the implicit polynomials $D(X), B(X)$ which were defined through $\overline{\boldsymbol{w}}$ and assume that $D(X)^2 \ne B(X)$. Then by the Schwartz-Zippel Lemma the polynomial $F(X) = D(X)^2 - B(X)$ can have at most $2m-2$ roots. But since $\Pr[y_2 = 0] > \frac{2m-2}{|\mathbb{F}|}$ its number of roots is bigger and so $F(X)$ must be the zero-polynomial. Due to the way $D(X), B(X)$ are constructed (i.e., $B(x) = D(x)$ for all $i \in [m]$), it thus follows that $\overline{\boldsymbol{w}}[1], \dots, \overline{\boldsymbol{w}}[m]$ are bits, which concludes the proof. $\square$

Applying this optimization and using $\Pi_{\mathsf{sac}}^{\mathrm{samp}}$, we obtain that the total communication is

$$|\mathsf{hash}| \cdot 2 + |\mathsf{sd}| \cdot (2 + M \log N) + |\mathtt{com}| \cdot M + \log q \cdot M(2m + 4)$$

which is approximately $\log q \cdot M(3m)$ bits smaller than the baseline approach.

5.4 Trading Argument Size for Slack

All of the arguments for SIS-instances that we have seen so far have in common that the gap between the norm of correct witnesses and the norm that the argument guarantees is small: if we start with $((\mathbf{A}, \boldsymbol{y}), \boldsymbol{s}) \in R_{\mathtt{SIS}}^{m,n,q,\beta}$ (i.e., $||\boldsymbol{s}||_\infty \leq \beta$) then the soundness guarantee is that a witness $\boldsymbol{s}'$ with $((\mathbf{A}, \boldsymbol{y}), \boldsymbol{s}') \in R_{\mathtt{SIS}}^{m,n,q,\tau\beta}$ could be extracted (i.e., $||\boldsymbol{s}||_\infty \leq \tau\beta$) where τ 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 $\boldsymbol{s}[i]$. The goal of this subsection is to give an *approximate* argument of size for the $\boldsymbol{s}[i]$ without having to resort to bit-decomposition for each $\boldsymbol{s}[i]$. 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 τ 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 $\boldsymbol{s}$ are small. For this we use a Lemma from [BL17] who showed that random linear combinations mod q of elements from $\boldsymbol{s}$ are with certain probability not much smaller than $||\boldsymbol{s}||_\infty$. Formally, they proved the following:

Lemma 7. *For all $\boldsymbol{s} \in \mathbb{F}_q^k$ it holds that*

$$\Pr_{\boldsymbol{c} \leftarrow \{0,1\}^k}\left[|\langle \boldsymbol{c}, \boldsymbol{s} \rangle| < \frac{1}{2} \cdot ||\boldsymbol{s}||_\infty\right] \leq \frac{1}{2} \quad \text{and} \quad \Pr_{\mathbf{C} \leftarrow \{0,1\}^{\ell \times k}}\left[||\mathbf{C} \cdot \boldsymbol{s}||_\infty < \frac{1}{2} \cdot ||\boldsymbol{s}||_\infty\right] \leq 2^{-\ell}.$$

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. We will now extend it to the case where we always see that lots such large elements in the product $\mathbf{C} \cdot \boldsymbol{s}$.

Lemma 8. *Let $\kappa, r \in \mathbb{N}^+, \boldsymbol{s} \in \mathbb{F}_q^k, \beta = ||\boldsymbol{s}||_\infty$ and define*

$$S_\kappa^\beta = \{\boldsymbol{h} \in \mathbb{F}_q^{r \cdot \kappa} \mid \exists T \subseteq [r \cdot \kappa] \wedge |T| > \kappa \wedge \forall i \in T: \ |\boldsymbol{h}[i]| \geq \frac{1}{2} \cdot \beta\}$$

Then

$$\Pr_{\mathbf{C} \leftarrow \{0,1\}^{(r \cdot \kappa) \times k}}\left[\mathbf{C} \cdot \boldsymbol{s} \notin S_\kappa^\beta\right] \leq \exp\left(-\kappa \frac{(r-2)^2}{2r}\right).$$

Proof. Set $\boldsymbol{r} = \mathbf{C} \cdot \boldsymbol{s}$ and define $X_i \in \{0, 1\}$ to be 0 iff $|\boldsymbol{r}[i]| \geq \beta/2$ and 1 otherwise. From Lemma 7 it follows that

$$\Pr_{\mathbf{C} \leftarrow \{0,1\}^{(r \cdot \kappa) \times k}}[X_i = 1] \leq 1/2$$

for all $i \in [r \cdot \kappa]$. Consider the mean $\overline{X} = \frac{1}{r \cdot \kappa} \sum_{i=1}^{r \cdot \kappa} X_i$. Then clearly $E[\overline{X}] = E[X_i] = \Pr[X_i = 1] \leq 1/2$ and thus $-E[\overline{X}] \geq -1/2$.

Using the one-sided Hoeffding inequality we obtain

$$\Pr_{\mathbf{C}\in\{0,1\}^{(r\cdot\kappa)\times k}}\left[\overline{X}-E[\overline{X}]\geq\frac{r-1}{r}-\frac{1}{2}\right]\leq\exp\left(-2(r\cdot\kappa)\cdot\left(\frac{r-2}{2r}\right)^2\right)$$

Since $-E[\overline{X}]\geq -1/2$, we have that

$$\Pr_{\mathbf{C}\in\{0,1\}^{(r\cdot\kappa)\times k}}\left[\overline{X}\geq\frac{r-1}{r}\right]\leq\exp\left(-\kappa\cdot\frac{(r-2)^2}{2r}\right)$$

Observe that for the event $\overline{X}\geq 1-1/r$ to happen, there must be $\leq\kappa$ variables for which $X_i=0$ (since in that case $\overline{X}=\frac{1}{r\cdot k}\sum_{i=1}^{r\cdot k}X_i\geq\frac{r\cdot k-k}{r\cdot k}=1-\frac{1}{r}$). Thus,

$$\Pr_{\mathbf{C}\in\{0,1\}^{(r\cdot\kappa)\times k}}\left[\overline{X}\geq\frac{r-1}{r}\right]=\Pr_{\mathbf{C}\leftarrow\{0,1\}^{(r\cdot\kappa)\times k}}\left[\mathbf{C}\cdot\boldsymbol{s}\notin S_\kappa^\beta\right]$$

and the claim follows. □

From the above Lemma we can easily derive the following:

Corollary 2. *For the same conditions as in Lemma 8, we have that if $r\geq 5$ then*

$$\Pr_{\mathbf{C}\leftarrow\{0,1\}^{(r\cdot\kappa)\times k}}\left[\mathbf{C}\cdot\boldsymbol{s}\notin S_\kappa^\beta\right]\leq 2^{-\kappa}$$

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 $\boldsymbol{s}$ with a random binary matrix to $\mathcal{V}$ without necessarily leaking information about $\boldsymbol{s}$.

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 $\boldsymbol{u}=\mathbf{C}\boldsymbol{s}$ and output $\boldsymbol{u}+$ "small" where "small" is a value of small norm. To achieve good soundness guarantees we let "small" only be polynomially bigger than $||\boldsymbol{u}||_\infty$ 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 $\boldsymbol{u}=\mathbf{C}\boldsymbol{s}$. We now describe both ideas in more detail and then formally express them in the context of circuit sampling, which allows to directly combine them with $\Pi_{\mathsf{sac}}^{\mathsf{samp}}$.

1st Approach: Rejection Sampling. In this solution, we let the prover $\mathcal{P}$ add additional random elements $x_1, x_2, \ldots$ 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 $x_1, x_2, \ldots$ to show that most of the remaining ones are indeed of small size. $\mathcal{P}$ will then open sums of each $\boldsymbol{u}[i]$ with some x_j, subject to the constraint that this does not leak information about $\boldsymbol{s}$. $\mathcal{V}$ later tests that each such $\boldsymbol{u}[i]+x_j$ 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. This, on a high level, works as follows:

1. $\mathcal{P}$ will sample $x_1, \ldots, x_{16\kappa}$ uniformly at random from $[-\pi \cdot m \cdot \beta, \pi \cdot m \cdot \beta] \subset \mathbb{F}$ and commit them as part of $\overline{\boldsymbol{w}}$.
2. $\mathcal{V}$ with probability 1/2 puts each x_i into a set E. Then, he samples a random bit matrix $\mathbf{C} \in \{0,1\}^{5\kappa \times m}$ and sends $E, \mathbf{C}$ as its challenges to the prover.
3. $\mathcal{P}$ now sets up a circuit C as follows:
   (a) C will output $\{x_e\}_{e \in \overline{E}}$. Then $\mathcal{V}$ can check that $x_e \in [-\pi \cdot m \cdot \beta, \pi \cdot m \cdot \beta]$.
   (b) Compute $\boldsymbol{u} = \mathbf{C}\boldsymbol{s}$ in the circuit. $\mathcal{P}$ will go through $\boldsymbol{u}[1], \ldots, \boldsymbol{u}[5 \cdot \kappa]$, take the first unused $e \in E$ and test if $\boldsymbol{u}[i] + x_e \in [-(\pi - 1) \cdot m \cdot \beta, (\pi - 1) \cdot m \cdot \beta]$. If so, then it makes C output $v_i = \boldsymbol{u}[i] + x_e$, otherwise it removes e from E and repeats this procedure with the next-largest $e' \in E$. $\mathcal{V}$ later checks that indeed $v_i \in [-(\pi - 1) \cdot m \cdot \beta, (\pi - 1) \cdot m \cdot \beta]$.

We present the full sampler in Fig. 7.

Set some auxiliary value $\pi = 100$. We will have $S_{\mathcal{V}} = \{(\boldsymbol{\gamma}, E, \mathbf{C}) \in \mathbb{F}_q^n \times \{0,1\}^{16\kappa} \times \{0,1\}^{5\kappa \times m}\}$.

ExtWitness: On input

$$(x = (\boldsymbol{t}[1], \ldots, \boldsymbol{t}[n]), w = (\boldsymbol{s}[1], \ldots, \boldsymbol{s}[m])) \in R_{\mathsf{SIS}}^{m,n,q,\beta}$$

sample $x_1, \ldots, x_{16\kappa}$ uniformly at random from $[-\pi \cdot m \cdot \beta, \pi \cdot m \cdot \beta] \subset \mathbb{F}$. Then set

$$\overline{\boldsymbol{w}} = (\boldsymbol{s}[1], \ldots, \boldsymbol{s}[m], x_1, \ldots, x_{16\kappa}) \in \mathbb{F}_q^{m+16\kappa}.$$

Response: On input $x, v, \overline{\boldsymbol{w}}, \tau_{\mathcal{V}}$ do the following:
1. Set $T \leftarrow \emptyset$ and let E be as in $\tau_{\mathcal{V}}$.
2. Let $\boldsymbol{u} \leftarrow \mathbf{C}\boldsymbol{s}$. For all $i \in [5\kappa]$:
   (a) Set $v_i \leftarrow \boldsymbol{u}[i] + x_e$ for the smallest $e \in E$.
   (b) Set $E \leftarrow E \setminus \{e\}$.
   (c) Set $T \leftarrow T \cup \{(i, e)\}$ if $v_i \in [-(\pi - 1) \cdot m \cdot \beta, (\pi - 1) \cdot m \cdot \beta]$, otherwise begin again for the next element in E.
3. Output $\tau_{\mathcal{P}} = T$.

SampCircuit: On input $x, \tau_{\mathcal{P}} = T, \tau_{\mathcal{V}} = (\boldsymbol{\gamma}, E, \mathbf{C})$ where $k = |\overline{E}|$ output the circuit C which performs the following:
1. Compute $\overline{\boldsymbol{y}} \leftarrow \mathbf{A}\boldsymbol{s}$.
2. Set $y \leftarrow \sum_{j \in [n]} \boldsymbol{\gamma}[j] \cdot (\overline{\boldsymbol{y}}[j] - \boldsymbol{t}[j])$.
3. Write $\overline{E} = \{e_1, \ldots, e_\ell\}$. Then for $i \in [\ell]$ set $v_i \leftarrow x_e$.
4. Compute $\boldsymbol{u} \leftarrow \mathbf{C}\boldsymbol{s}$.
5. For $i \in [5\kappa]$ set $v_{i+\ell} \leftarrow \boldsymbol{u}[i] + x_{t_i}$ where $(i, t_i) \in T$.
6. Output of C is $(y, v_1, \ldots, v_{\ell+5\kappa})$.

We implicitly set

$$Y \leftarrow \left\{ (0, v_1, \ldots, v_{k+5\kappa}) \in \mathbb{F}_q^{\ell+5\kappa+1} \,\middle|\, \begin{array}{l} \forall i \in [\ell] : \; v_i \in [-\pi \cdot m \cdot \beta, \pi \cdot m \cdot \beta] \wedge \\ \forall j \in [5\kappa] : \; v_{\ell+j} \in [-(\pi - 1) \cdot m \cdot \beta, (\pi - 1) \cdot m \cdot \beta] \end{array} \right\}.$$

Fig. 7: Sampling of a circuit for $R_{\mathsf{SIS}}^{m,n,q,4\pi m \cdot \beta}$.

Theorem 5. *The algorithms in Fig. 7 yield a statistically correct, α-sound and perfectly simulatable circuit sampler for the relation $R_{\mathsf{SIS}}^{m,n,q,4\pi m \cdot \beta}$ where $\alpha = \max\{1/|\mathbb{F}|, 2^{-\kappa}\}$.*

Proof. We prove that the sampler satisfies the definitions in Section 4.1.

Correctness. We show that if the prover follows the sampler instructions, then $C(\overline{\boldsymbol{w}}) \in Y$ except for a negligible probability in κ. Clearly, this event that $C(\overline{\boldsymbol{w}}) \notin Y$ occurs only when 16κ

samples of x_e are not sufficient. Thus, we need to show that this indeed happens only with negligible probability. To prove this, we first compute the probability that x_e is "thrown away" and not used in the sum with $\boldsymbol{u}[i]$. As we assume that the prover acts honestly, we know that $x_e \in [-\pi \cdot m \cdot \beta, \pi \cdot m\beta]$, i.e., x_e is sampled from a set of size $2\pi \cdot m \cdot \beta + 1$. In addition $-\beta \le \boldsymbol{s}[i] \le \beta$ and so $-m\beta \le \boldsymbol{u}[i] \le m\beta$. Denote by bad_x the event that x_e is not used. Thus, given $\boldsymbol{u}[i]$, we can write

$$\begin{aligned}\Pr[\mathsf{bad}_x] &= \Pr\left[\boldsymbol{u}[i] + x_e < -\pi \cdot m \cdot \beta + m \cdot \beta\right] + \Pr\left[\boldsymbol{u}[i] + x_e > \pi \cdot m \cdot \beta - m \cdot \beta\right] \\ &= \Pr\left[x_e < -\pi \cdot m \cdot \beta + m \cdot \beta - \boldsymbol{u}[i]\right] + \Pr\left[x_e > \pi \cdot m \cdot \beta - m \cdot \beta - \boldsymbol{u}[i]\right] \\ &= \frac{\pi \cdot m \cdot \beta + m \cdot \beta - \boldsymbol{u}[i] - (-\pi \cdot m \cdot \beta)}{2\pi \cdot m \cdot \beta + 1} + \frac{\pi \cdot m \cdot \beta - (\pi \cdot m \cdot \beta - m \cdot \beta - \boldsymbol{u}[i])}{2\pi \cdot m \cdot \beta + 1} \\ &= \frac{2m \cdot \beta}{2\pi \cdot m \cdot \beta + 1} < \frac{2m \cdot \beta}{2\pi \cdot m \cdot \beta} = \frac{1}{\pi}.\end{aligned}$$

Note that this probability is independent of the value $\boldsymbol{u}[i]$. Therefore, given that each x_e is being sent in the clear to $\mathcal{V}$ with probability $1/2$, we obtain that each x_e is not used in the sum with $\boldsymbol{u}[i]$ with probability $p < 1/2 + 1/2 \cdot 1/\pi$.

We now compute the probability that there aren't enough samples of x_e to construct the circuit. Let $X_e \in \{0, 1\}$ be a random variable such that $X_e = 0$ with probability p and $X_e = 1$ with probability $1 - p$. Furthermore, define $\overline{X} = \frac{1}{16\kappa}\sum_{e=1}^{16\kappa} X_e$. Since there are 16κ samples of X_e to begin with and 5κ are required to complete the circuit construction successfully, we need to determine $\Pr\left[\sum_{e=1}^{16\kappa} X_e < 5\kappa\right] = \Pr\left[\overline{X} < \frac{5}{16}\right]$. Observe that furthermore $E\left[\overline{X}\right] = p < 1/2 + 1/2\pi$ and therefore $\Pr\left[\overline{X} < \frac{5}{16}\right] \le \Pr\left[\overline{X} - E[\overline{X}] < \frac{5}{16} - \frac{1}{2} - \frac{1}{2\pi}\right] = \Pr\left[E[\overline{X}] - \overline{X} > \frac{3}{16} + \frac{1}{2\pi}\right]$.

Using the Hoeffding bound we obtain

$$\begin{aligned}\Pr\left[\sum_{e=1}^{16\kappa} X_e < 5\kappa\right] &\le \exp\left(-2 \cdot 16\kappa \cdot \left(\frac{3\pi + 8}{16\pi}\right)^2\right) \\ &\le \exp(-\kappa)\end{aligned}$$

where the last inequality holds for any $\pi \ge 1$.

Simulatability. Recall that the definition of in Section 4.1, we need to show that a simulator who receives x, w and $\tau_{\mathcal{V}}$ as its input can output $\tau_{\mathcal{P}}$ and $C(\overline{\boldsymbol{w}})$ that are indistinguishable from an output of a real execution. As we have seen, the aforementioned "throwing away" probability is actually independent of the value $\boldsymbol{u}[i]$ as long as $\boldsymbol{u}[i] \in [-m \cdot \beta, m \cdot \beta]$. One can therefore simulate $\tau_{\mathcal{P}}$ by flipping a biased coin. To simulate $C(\overline{\boldsymbol{w}})$, each value $v_1, \ldots, v_{\ell+5\kappa}$ is simply chosen from its respective interval. This is obviously a perfect simulation for $v_1, \ldots, v_\ell$ whereas $v_{\ell+1}, \ldots, v_{\ell+5\kappa}$ is uniformly random in its interval by the choice of x_i in $\mathsf{Response}$.

α-Soundness. Let $\overline{\boldsymbol{w}}$ be fixed and assume that the prover succeeds with probability $> \alpha = \max\{1/|\mathbb{F}|, 2^{-\kappa}\}$. As in previous arguments, this particularly implies that the committed $\overline{\boldsymbol{w}}$ contains a correct preimage of $\boldsymbol{t}$ under multiplication with $\mathbf{A}$ and it remains to show that this preimage is of correct size.

By the fact that each x_e is chosen to be in the set E with probability $1/2$, it follows that all but κ of the unopened x_e must be within the interval $[-\pi \cdot \beta \cdot m, \pi \cdot \beta \cdot m]$, as otherwise the success probability of the prover must be lower than $\frac{1}{2^\kappa}$.

Let's consider the vector $\boldsymbol{u}$ which the prover computes. By Corollary 2 we know that if $\mathcal{P}$ uses $\boldsymbol{s}$ such that $||\boldsymbol{s}||_\infty \ge (4\pi - 2)m \cdot \beta + 2$ then $\boldsymbol{u} \in S_\kappa^{(4\pi-2)m\beta+2}$ (i.e., there exist $> \kappa$ values $\boldsymbol{u}[j]$

for which $|\boldsymbol{u}[j]| \geq (2\pi - 1)m \cdot \beta + 1)$, except with probability $2^{-\kappa}$. Thus for the output of the circuit to be in Y each $\boldsymbol{u}[j]$ with $|\boldsymbol{u}[j]| \geq (2\pi - 1)m\beta + 1 > (2\pi - 1)m\beta$ must be paired with a value x_e with $|x_e| > \pi m\beta$ so that $|v_j| = |\boldsymbol{u}[j] + x_e| \leq (\pi - 1)m\beta$. As there are at most κ many such "bad" x_i, the prover can make at most κ bad sums and at least one generated v_j will be outside of the interval and thus be noticed, except with probability $2^{-\kappa}$. Therefore, if the success probability of the prover is higher than $2^{-\kappa}$, then $\overline{\boldsymbol{w}}$ must contain a preimage of $\boldsymbol{t}$ of bound at most $4\pi m\beta \geq (4\pi - 2) \cdot m \cdot \beta + 2$. □

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 7. 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 actual argument (neglecting the length of $\tau_{\mathcal{P}}$) when using $\Pi_{\mathsf{sac}}^{\mathrm{samp}}$ is

$$|\mathsf{hash}| \cdot 2 + |\mathsf{sd}| \cdot (2 + M \log N) + |\mathtt{com}| \cdot M + \log_2 q \cdot M(m + 29\kappa + 1)$$

2^{nd} Approach: The Power of Random Bits. The circuit sampler from Fig. 7 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.

To reduce the slack, we will again rely on Lemma 7. But instead of performing rejection sampling on the output, we perform a range proof for each element of the matrix product $\boldsymbol{u} = \mathbf{C}\boldsymbol{s}$. 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 $||\boldsymbol{s}||_\infty \leq \beta$, then since $\mathbf{C} \in \{0,1\}^{\kappa \times m}$ it must hold that $||\mathbf{C}\boldsymbol{s}||_\infty \leq m \cdot \beta$. Thus, letting r be the smallest integer such that $m \cdot \beta < 2^r$, it suffices for the prover to show that $\boldsymbol{u}[i] \in [0, 2^r - 1]$ (we can also shift the interval as done in Section 5.1). To show the inclusion $\mathcal{P}$ can add random bits $x_1^i, \ldots, x_r^i$ to the witness. Then, once the challenge is received from $\mathcal{V}$ and $\boldsymbol{u}$ is known to $\mathcal{P}$, it can compute the bit decomposition $\boldsymbol{u}[i] = \sum_{j=1}^{r} 2^j h_j^i$ for each $i \in [\kappa]$ and tell $\mathcal{V}$ for each $j \in [r]$ 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 Section 5.3, which uses polynomial evaluation and requires a single non-linear gate. We provide the full circuit sampler in Fig. 8.

Theorem 6. *Assume that $(q - 1)/2 > 4m\beta$. The algorithms in Fig. 8 yield a perfectly correct, α-sound and perfectly simulatable circuit sampler for the relation $R_{\mathtt{SIS}}^{m,n,q,2m \cdot \beta + 4}$ where $\alpha = \max\{\frac{2(r+1)\kappa - 1}{|\mathbb{F}|}, 2^{-\kappa}\}$ and r is the smallest integer such that $m \cdot \beta \leq 2^r - 1$.*

```
We will have $S_{\mathcal{V}} = \{(\boldsymbol{\gamma}, \hat{x}, \mathbf{C}) \in \mathbb{F}_q^n \times \mathbb{F}_q \times \{0,1\}^{\kappa \times m}\}$.
Let $\ell_j^T(x) = \prod_{\substack{1 \le i \le h \\ i \ne j}} \frac{x - x_i}{x_j - x_i}$ be Lagrange coefficients, where $T = \{(x_i, y_i)\}_{i \in h}$ is the set of points used for polynomial evaluation.

ExtWitness: On input $(x = (\boldsymbol{t}[1], \ldots, \boldsymbol{t}[n]), w = (\boldsymbol{s}[1], \ldots, \boldsymbol{s}[m])) \in R_{\mathtt{SIS}}^{m,n,q,\beta}$ do the following:
 1. For $i \in [\kappa]$ sample the random bits $x_1^i, \ldots, x_{r+1}^i$.
 2. Compute the unique polynomial $D(X)$ of degree $(r+1)\kappa - 1$ where $D((i-1)(r+1)+j) = x_j^i$ for $i \in [\kappa], j \in [r+1]$. Furthermore, compute the polynomial $B(X) = (D(X))^2$ and set $S_D = \{(i, D(i))\}_{i \in [(r+1)\kappa]}$ and $S_B = S_D \cup \{(i, B(i))\}_{i \in [2(r+1)\kappa - 1] \setminus [(r+1)\kappa]}$ as the evaluation points of the polynomials $D, B$.
 3. Set
      $$\overline{\boldsymbol{w}} = (\boldsymbol{s}[1], \ldots, \boldsymbol{s}[m], x_1^1, \ldots, x_{r+1}^1, \ldots, x_1^\kappa, \ldots, x_{r+1}^\kappa, z_1, \ldots, z_{(r+1)\kappa-1}) \in \mathbb{F}_q^{m+(r+1)\kappa}$$
    where $z_i = B(i + (r+1)\kappa) = (D(i + (r+1)\kappa))^2$.

Response: On input $x, v, \overline{\boldsymbol{w}}, \tau_{\mathcal{V}}$ do the following:
 1. Set $\boldsymbol{u} \leftarrow \mathbf{C}\boldsymbol{s}$. Then for all $i \in [\kappa]$ do the following:
    (a) Set $v_i \leftarrow \boldsymbol{u}[i] + m \cdot \beta$.
    (b) Find $h_0^i, \ldots, h_r^i \in \{0,1\}$ such that $v_i = \sum_{j=0}^r h_j^i 2^j$.
    (c) For each $j \in \{0, \ldots, r\}$ set
          $$b_j^i \leftarrow \begin{cases} 0 & \text{if } h_j^i = x_j^i \\ 1 & \text{otherwise} \end{cases}$$
 2. Output $\tau_{\mathcal{P}} = (\{b_0^i, \ldots, b_r^i\}_{i \in [\kappa]})$.

SampCircuit: On input $x, \tau_{\mathcal{P}} = (\{b_0^i, \ldots, b_r^i\}_{i \in [\kappa]}), \tau_{\mathcal{V}} = (\boldsymbol{\gamma}, \hat{x}, \mathbf{C})$ the circuit $C$ performs the following:
 1. Compute $\overline{\boldsymbol{y}} \leftarrow \mathbf{A}\boldsymbol{s}$.
 2. Set $y_1 \leftarrow \sum_{j \in [n]} \boldsymbol{\gamma}[j] \cdot (\overline{\boldsymbol{y}}[j] - \boldsymbol{t}[j])$.
 3. Set $\boldsymbol{u} \leftarrow \mathbf{C}\boldsymbol{s}$.
 4. $z_1 \leftarrow \sum_{i=1}^\kappa \sum_{j=1}^{r+1} \ell_{(i-1)(r+1)+j}^{S_D}(\hat{x}) \cdot x_j^i$
 5. $z_2 \leftarrow \sum_{i=1}^\kappa \sum_{j=1}^{r+1} \ell_{(i-1)(r+1)+j}^{S_B}(\hat{x}) \cdot x_j^i + \sum_{j=1}^{(r+1)\kappa-1} \ell_{j+(r+1)\kappa}^{S_B}(\hat{x}) \cdot z_j$
 6. $z_3 \leftarrow z_1^2$
 7. $y_2 \leftarrow z_3 - z_2$
 8. For each $i \in [\kappa], j \in \{0, \ldots, r\}$ set
          $$h_j^i \leftarrow \begin{cases} x_{j+1}^i & \text{if } b_j^i = 0 \\ 1 - x_{j-1}^i & \text{otherwise} \end{cases}$$
 9. For $i \in [\kappa]$ set $y_{i+2} \leftarrow \boldsymbol{u}[i] + m \cdot \beta - \left(\sum_{j=0}^r h_j^i 2^j\right)$.
10. Output $(y_1, \ldots, y_{\kappa+2})$.
We output the set
          $$Y = \left\{(0, \cdots, 0) \in \mathbb{F}_q^{\kappa+2}\right\}.$$
```

Fig. 8: Sampling of a circuit for $R_{\mathtt{SIS}}^{m,n,q,3m\cdot\beta}$.

Proof. Correctness: Most of the correctness follows simply by linearity as in the other constructions, so we will focus on the bit-decomposition part.

The multiplication of $\boldsymbol{s}$ with $\mathbf{C}$ trivially yields a bound $||\boldsymbol{u}||_\infty \le m \cdot ||\boldsymbol{s}||_\infty = m \cdot \beta$. By shifting each $\boldsymbol{u}[i]$ with the constant $m \cdot \beta$ we will have that $\boldsymbol{u}[i] + m \cdot \beta \in [0, 2^{r+1} - 1]$. This can always be represented with $r+1$ bits as in the protocol.

α-Soundness: The prover has three ways it can get away with using an invalid witness. The first is to use a input vector $\boldsymbol{s}$ which is *not* a preimage of $\boldsymbol{t}$ under multiplication with $\mathbf{A}$. But as in previous constructions this will only succeed with probability $1/|\mathbb{F}|$ (due to the random linear combination of outputs). A second option to cheat is to use x_j^i that are *not* bits. Following the same idea as in the proof of Theorem 4, this prover can succeed here with probability of at most $\frac{2(r+1)\kappa-1}{|\mathbb{F}|}$. The third way is to use an input vector with norm that is not in allowed range. We will show now that the success probability in this case is at most $2^{-\kappa}$. Observe that $\frac{2(r+1)\kappa-1}{|\mathbb{F}|} > \frac{1}{|\mathbb{F}|}$ and so the overall soundness error is as defined in the Theorem.

Thus, for the rest of the proof we assume that the committed values x_j^i are bits. Assume that $\boldsymbol{s}$ as committed in the witness $\overline{\boldsymbol{w}}$ is such that $||\boldsymbol{s}||_\infty \geq 2m\beta + 4$. Then by Lemma 7 we have that $\Pr[||\boldsymbol{u}||_\infty < m \cdot \beta + 2] \leq 2^{-\kappa}$. If $||\boldsymbol{u}||_\infty \geq m \cdot \beta + 2$ then there exists an index i such that either $\boldsymbol{u}[i] \in [-(q-1)/2, -m \cdot \beta - 2]$ or $\boldsymbol{u}[i] \in [m \cdot \beta + 2, (q-1)/2]$. In the first case we have $v_i = \boldsymbol{u}[i] + m \cdot \beta$ which results in $v_i \in [-(q-1)/2 + m \cdot \beta, -2]$. But no such v_i can be represented as $v_i = \sum_{j=0}^r h_j^i 2^j$ using bits h_j^i only. If on the other hand $\boldsymbol{u}[i] \in [m \cdot \beta + 2, (q-1)/2]$ then $v_i \in [2m \cdot \beta + 2, (q-1)/2] \cup [-(q-1)/2, -(q-1)/2 + m\beta - 1]$. But the largest number which the sum $\sum_{j=0}^r h_j^i 2^j$ can express with $h_j^i \in \{0, 1\}$ is $2^{r+1} - 1 = 2m\beta + 1$ and the values from $[-(q-1)/2, -(q-1)/2 + m\beta - 1]$ are not expressible by the bound on q. As the prover succeeds with probability strictly larger than $2^{-\kappa}$ to represent all v_i as bits, it must hold by Lemma 7 that $||\boldsymbol{s}||_\infty < 2m\beta + 4$ and the claim follows.

Simulatable: The set Y is fixed for all instances. $\tau_{\mathcal{P}}$ consists of bits b_j^i which are essentially the XOR of the bit decomposition of the secret $\boldsymbol{u}[i] + m\beta$ with uniformly random bits x_j^i and thus perfectly simulatable. □

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

$$|\mathsf{hash}| \cdot 2 + |\mathsf{sd}| \cdot (2 + M \log N) + |\mathtt{com}| \cdot M + \log_2 q \cdot M(m + \kappa(r+2) + 5).$$

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 10Gpbs. For our implementation we used only the baseline construction for the Binary-SIS problem presented in Section 5.1. Nevertheless, this includes the three general optimizations described in Section 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, $|\mathsf{hash}| = |\mathtt{com}| = 256$ bits and $|\mathsf{sd}| = 128$ bits.

Binary-SIS Problem parameters. We used five sets of parameters for our experiments:

1. $\log_2 |\mathbb{F}| = 15$, $n = 256$ and $m = 1024$.
2. $\log_2 |\mathbb{F}| = 15$, $n = 256$ and $m = 4096$.
3. $\log_2 |\mathbb{F}| = 31$, $n = 512$ and $m = 2048$.
4. $\log_2 |\mathbb{F}| = 59$, $n = 1024$ and $m = 4096$.
5. $\log_2 |\mathbb{F}| = 61$, $n = 1024$ 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

N	**Cut-and-Choose**						**Sacrificing**			
	$\xi \leq 2^{-40}$			$\xi \leq 2^{-80}$			$\xi \leq 2^{-40}$		$\xi \leq 2^{-80}$	
	M	τ	**Argument size**	M	τ	**Argument Size**	M	**Argument Size**	M	**Argument Size**
2	75	34	$31 + 0.123 \cdot \rho m$	145	63	$61.1{+}0.246{\cdot}\rho m$	40	$26.2{+}0.16{\cdot}\rho m$	80	$51.8{+}0.32{\cdot}\rho m$
4	55	32	$22.4{+}0.069{\cdot}\rho m$	105	57	$44.8{+}0.144{\cdot}\rho m$	20	$16{+}0.08{\cdot}\rho m$	40	$31.3{+}0.16{\cdot}\rho m$
8	55	38	$20.7{+}0.051{\cdot}\rho m$	95	57	$42{+}0.114{\cdot}\rho m$	14	$13.2{+}0.056{\cdot}\rho m$	27	$24.8{+}0.108{\cdot}\rho m$
16	45	26	$23.4{+}0.057{\cdot}\rho m$	95	63	$41.5{+}0.096{\cdot}\rho m$	10	$10.9{+}0.04{\cdot}\rho m$	20	$21.2{+}0.08{\cdot}\rho m$
32	45	28	$23.8{+}0.051{\cdot}\rho m$	85	47	$50.4{+}0.114{\cdot}\rho m$	8	$9.9{+}0.032{\cdot}\rho m$	16	$19.1{+}0.064{\cdot}\rho m$
64	45	28	$26{+}0.051{\cdot}\rho m$	85	49	$53{+}0.108{\cdot}\rho m$	7	$9.6{+}0.028{\cdot}\rho m$	14	$18.6{+}0.056{\cdot}\rho m$

Table 1: Summary of Parameters used in the experiments for each protocol and the argument size in Kbits for each set of parameters as a function of the SIS problem parameters $\rho = \log_2 |\mathbb{F}|$ and m.

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].

Protocol parameters. We ran experiments for 40 and 80 bits of statistical security κ. For the parameter N, which is the number of parties in the underlying MPC protocol, we used 6 different values: 2,4,8,16,32 and 64. Then, given the desired level of security and the number of parties, we searched for the set of parameters in each of the two protocols, that minimizes 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 the minimal 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 Table 1. In addition, for each set of parameters we give the size of the argument in Kbits as a formula of the SIS problem parameters $\rho = \log_2 |\mathbb{F}|$ and m. 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.

Running times. In Table 2 and Table 3 we present the running times (in Msec.) of the two protocols for 40 and 80 bits of security respectively. For each SIS problem set of parameters, 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. When compared with the data presented in Table 1 which shows that the sacrificing protocol has lower argument size and that argument size usually becomes smaller as the number of

ρ	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

Table 2: Best running times in MSec for different sets of SIS problem parameters with 40-bit security.

ρ	n	m	Cut-and-Choose				Sacrificing		
			N	M	τ	**time**	N	M	**time**
15	1024	256	2	145	63	146	4	40	118.6
15	256	4096	2	145	63	589.2	4	40	507.7
31	512	2048	2	145	63	505.4	4	40	436.8
59	1024	4096	2	145	63	2000.4	2	80	2138.8
61	1024	4096	2	145	63	2396.8	2	80	2432.3

Table 3: Best running times in MSec for different sets of SIS problem parameters with 80-bit security.

parties grows (which is to be expected), this indicates that, in the current state, computation time is the bottleneck of the protocol.

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 $\boldsymbol{s}$ (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% in our results.

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. In Fig. 9 we present the improvement in the running time when utilizing more threads. This experiment was run for the most "toughest" instance of the SIS problem, with $\rho = \log|\mathbb{F}| = 61$, $n = 1024$ and $m = 4096$ and with the MPC protocol parameters who yielded the best running time in Table 2 and Table 3. As can be seen, using many threads speeds-up the performance by more than 80%. As a consequence, we obtain a lattice ZKAoK that runs in less than 0.5 seconds 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 & Structured Lattices. In this work we solely focus on unstructured matrices A for SIS. By micro-benchmarking the results, we observe that as the size of the matrix 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

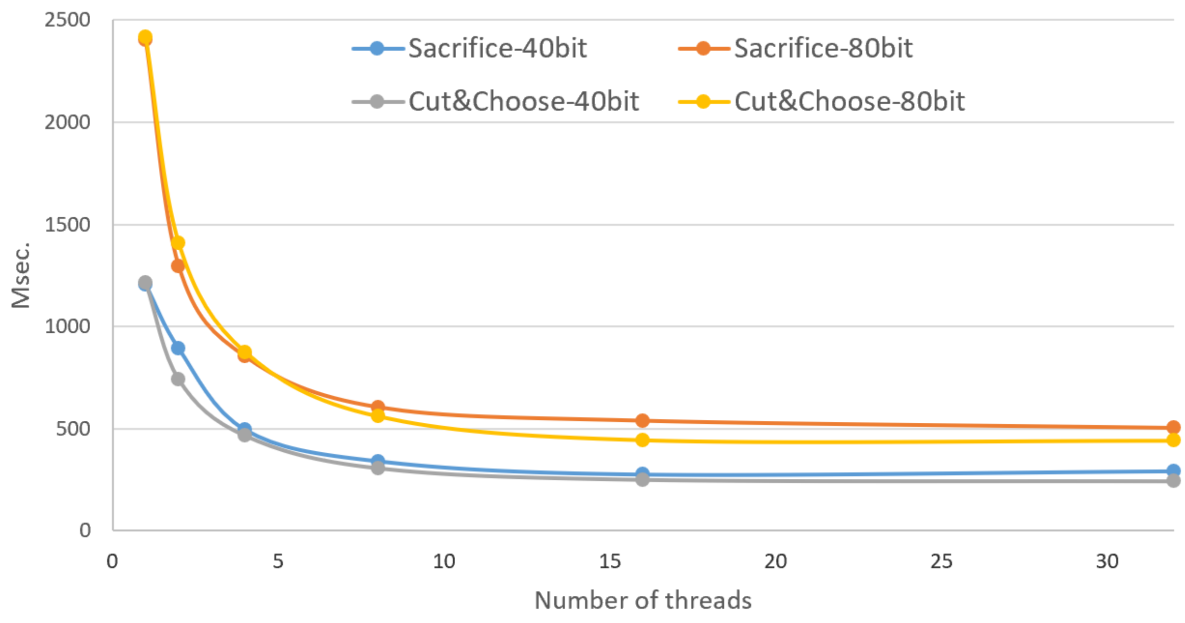

Fig. 9: Running time in Msec. as a function of the number of threads used, for the instance with the parameters: $\rho = 61$, $n = 1024$, $m = 4096$ and $N = 2$

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 Comparison to Previous Works

We now describe how our protocol compares in terms of proof size with other state-of-the-art arguments of knowledge. For this comparison, we consider only our second "sacrificing"-based protocol. We stress that comparing only the proof size does not provide a complete picture, since communication is not necessarily the bottleneck (as can be observed from the experimental results in the previous section). As we focus on practicality, the comparison focuses on two instances of the SIS problem: the Binary-SIS problem and a more general instance where $||\boldsymbol{s}||_\infty \leq 15$ (i.e., $\beta = 15$ and so the secrets are of 5-bit size). For each instance, we consider three set of parameters: 40-bit and 80-bit soundness security for $\log_2(|\mathbb{F}|) \approx 32$ and $\mathbf{A} \in \mathbb{F}^{512\times 2048}$ as well as 128-bit soundness for $\log_2(|\mathbb{F}|) \approx 61$ and $\mathbf{A} \in \mathbb{F}^{1024\times 4096}$. As in the previous subsection, for seed-size we use 128 bits, whereas length of commitments and outputs of hash function-calls will be 256 bits. Whenever necessary, we will use the same values for comparable parameters in other protocols. Since our focus here is on communication only, we use $N = 16$ simulated parties in the underlying MPC protocol, and so we set $M = 11, 21$ and 33 to achieve $40, 80$ and 128 bit of soundness security.

Protocol	$\log_2(\lvert\mathbb{F}\rvert) = 32$, Binary	$\log_2(\lvert\mathbb{F}\rvert) = 32, \beta = 15$	$\log_2(\lvert\mathbb{F}\rvert) = 61$, Binary	$\log_2(\lvert\mathbb{F}\rvert) = 61, \beta = 15$
Stern [Ste96]	971 KB	7285 KB	3703 KB	27775 KB
Ligero [AHIV17]	1158 KB	1174 KB	2821 KB	2860 KB
Ours, baseline	357KB	2138 KB	1359KB	8148 KB
Ours, amortized	179 KB	1069 KB	680 KB	4075 KB

Table 4: Proof sizes for Binary-SIS and secrets of 5-bit size, small constant slack, 40-bit soundness.

Protocol	$\log_2(\lvert\mathbb{F}\rvert) = 32$, Binary	$\log_2(\lvert\mathbb{F}\rvert) = 32, \beta = 15$	$\log_2(\lvert\mathbb{F}\rvert) = 61$, Binary	$\log_2(\lvert\mathbb{F}\rvert) = 61, \beta = 15$
Stern [Ste96]	1457 KB	10928 KB	5555 KB	41663 KB
Ligero [AHIV17]	2218 KB	2245 KB	6009 KB	6085 KB
Ours, baseline	682 KB	4082 KB	2595KB	15557 KB
Ours, amortized	342 KB	2042 KB	1299 KB	7780 KB

Table 5: Proof sizes for Binary-SIS and secrets of 5-bit size, small constant slack, 80-bit soundness.

Protocol	$\log_2(\lvert\mathbb{F}\rvert) = 61$, Binary	$\log_2(\lvert\mathbb{F}\rvert) = 61, \beta = 15$
Stern [Ste96]	11851 KB	88882 KB
Ligero [AHIV17]	10173 KB	10309 KB
Ours, baseline	4077 KB	24461 KB
Ours, amortized	2041 KB	12225 KB

Table 6: Proof sizes for Binary-SIS and secrets of 5-bit size, small constant slack, 128-bit soundness.

Protocols with Small Constant Slack. We subsume all protocols that achieve constant slack here. These are either based on Stern-type arguments [LNSW13], direct applications of MPC-in-the-head [AHIV17] or IOP-based constructions [BSCR$^+$18]. Though STARKs [BSBHR18] fall into the third category, we do not consider those here as they are rather tailored to computations with looping components.

While [LNSW13] is a specific technique tailored to problems such as SIS, [AHIV17,BSCR$^+$18] require an arithmetic circuit (similar to us) for the verification of the statement. Recall that in our system, all linear gates of these circuits are for free whereas in [AHIV17,BSCR$^+$18] they necessarily contribute to the size of the proof. The circuits for the two different problems follow directly from Section 5.1: for the Binary-SIS proof we have $m \cdot (n + 2)$ arithmetic gates with m inputs and $m + n$ outputs, whereas the proof for the general SIS problem has $m \cdot (n + 2r)$ arithmetic gates, $m \cdot r$ inputs and $n + m \cdot r$ outputs (recall that r is the smallest integer such that $\beta \leq 2^r - 1$).

In Table 4 we present the proof size for 40 bits of soundness, whereas Table 5 and Table 6 deal with 80 bits and 128 bits, respectively. For our protocol, we present the proof size when using the baseline protocol (Section 5.1) and when using the ‘amortizing bit tests’ optimization from Section 5.3.

Observe that the tables do not include proof sizes for the recent Aurora protocol [BSCR$^+$18], as the authors there did not provide any general expression for the proof size, but rather only experimental results for the binary field $\mathbb{F}_{2^{192}}$. However, even for this large field, their proof size is around 250KB for a circuit with the same size as ours. Though [BSCR$^+$18]’s current implementation

Protocol	Slack	$\log_2(\lvert\mathbb{F}\rvert) = 32$ 40 bits	$\log_2(\lvert\mathbb{F}\rvert) = 32$ 80 bits	$\log_2(\lvert\mathbb{F}\rvert) = 61$ 128 bits
Sigma-protocol [Lyu12]	$288m$	184KB	368KB	1241KB
Ours, Approach 1 ($\kappa = 8$)	$400m$	100KB	191KB	1079KB
Ours, Approach 2 ($\kappa = 8$)	$< 3m$	96KB	183KB	1057KB
Ours, Exact	1	179KB	342KB	2041KB

Table 7: Proof sizes for Binary-SIS non-constant slack.

Protocol	Slack	$\log_2(\lvert\mathbb{F}\rvert) = 32$ 40 bits	$\log_2(\lvert\mathbb{F}\rvert) = 32$ 80 bits	$\log_2(\lvert\mathbb{F}\rvert) = 61$ 128 bits
Sigma-protocol [Lyu12]	$288m$	223KB	447KB	1494KB
Ours, Approach 1 ($\kappa = 8$)	$400m$	100KB	191KB	1079KB
Ours, Approach 2 ($\kappa = 8$)	$< 3m$	97KB	184KB	1061KB
Ours, Exact	1	1069KB	2042KB	12225KB

Table 8: Proof sizes for SIS with $\beta = 15$ and non-constant slack.

is only for binary fields and they require to use field extensions (whereas we can work over the prime field itself), it is fair to assume that the proof size for smaller fields will be smaller than the one obtained by our protocols. Nevertheless, the prover running time according to their experiments is approximately 200sec, and so is expected to be at least one order of magnitude bigger than the prover's running time in our protocols (the same applies to [AHIV17] whose prover's running time is even higher than of [BSCR$^+$18]). Thus, in applications where only the proof size matters, their protocol might be preferable, whereas in applications where the overall running time of the interactive protocol is more important, our argument system will outperform theirs.

Protocols with Non-constant Slack. Here, we compare with argument system from the signature scheme of [Lyu12].

We present the comparison in Tables 7 and 8. The scheme of Lyubashevsky consists of 3 steps in each round i: (i) $\mathcal{P}$ sends a value $\boldsymbol{a}_i = \mathbf{A}\boldsymbol{x}_i$ where $\boldsymbol{x}_i$ is sampled according to a discrete Gaussian distribution; (ii) $\mathcal{V}$ sends a challenge bit e_i; and (iii) $\mathcal{P}$ finishes by either sending $\boldsymbol{z}_i = \boldsymbol{x}_i + e_i \cdot \boldsymbol{s}$ or aborting. For the sake of comparison we consider as message complexity only the messages that are sent in the last of the three rounds, as the first message can be compressed into a commitment and then the opening be obtained from the third message. A further optimization, which we do not consider, is that the third message itself must only be sent whenever $e_i = 1$. Whenever $e_i = 0$, it would instead be feasible to just send a PRG seed which was used to generate the vector $\boldsymbol{x}_i$. But this vector is sampled according to a discrete Gaussian distribution and re-running such a sampler would be rather time-consuming. The vector $\boldsymbol{x}_i$ could alternatively be sampled from a bounded uniform distribution as in our case, but that would increase the slack proportionally. We could arguably also further decrease the size of our new arguments by increasing the number of parties N, which would also lead to higher computational cost on our side.

We compare the proof size of [Lyu12] with our baseline protocol and with the two solutions described in Section 5.4. We see that in particular the 2nd protocol of Section 5.4 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] stays the same.

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].

Protocols not based on post-quantum assumptions. 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 1.5KB), we think that comparing our work to theirs would not be fair. First, the soundness of their construction relies on the Discrete-Log assumption and is therefore not post-quantum secure. Moreover, their computational efficiency relies on using structured lattices, which we do not need.

For the same reason, we excluded Hyrax [WTS$^+$18] and Sonic [MBKM19] from the comparison. As [dPLS19] these constructions rely on the Discrete Logarithm-assumption for their security. Older ZK-SNARKs such as [PHGR16,BSCTV14] would offer low argument size and verification time but in addition to large keys and a high prover runtime also rely on very strong assumptions. Similarly,the work of [BCC$^+$16] is also in the DLog setting. Its lattice-based variant [BBC$^+$18] is so far not implemented, may have large hidden constants and itself uses ZKAoKs for SIS as building blocks.

Conclusions. We conclude that in terms of achieving both small proof size and low prover's running time, our scheme is the most efficient compared to other proof systems which rely on similar assumptions. We remark again that if the goal is only to minimize communication, it is possible to further increase the number of parties in the underlying MPC scheme and reduce the proof size even more.

Acknowledgments

We thank Roey Sefi and Assi Barak for their help with the implementation as well as Carmit Hazay, Yehuda Lindell and Avishay Yanay for helpful comments.

References

AHIV17. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 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. Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafael del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In *Advances in Cryptology – CRYPTO 2018*. Springer International Publishing, 2018.

BCC$^+$16. Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*. Springer Berlin Heidelberg, 2016.

BD10. Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In *Theory of Cryptography Conference*. Springer, 2010.

BDL$^+$18. Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. More efficient commitments from structured lattice assumptions. In Dario Catalano and Roberto De Prisco, editors, *Security and Cryptography for Networks*, Cham, 2018. Springer International Publishing.

BDLN16. Carsten Baum, Ivan Damgård, Kasper Green Larsen, and Michael Nielsen. How to prove knowledge of small secrets. In *Annual Cryptology Conference*. Springer, 2016.

Bea91. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*. Springer, 1991.

BGV14. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3), 2014.

BHR12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012.

BL17. Carsten Baum and Vadim Lyubashevsky. Simple amortized proofs of shortness for linear relations over polynomial rings, 2017. `https://eprint.iacr.org/2017/759`.

BSBHR18. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. `https://eprint.iacr.org/2018/046`.

BSCR$^+$18. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for r1cs. Cryptology ePrint Archive, Report 2018/828, 2018. `https://eprint.iacr.org/2018/828`.

BSCTV14. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security Symposium*, 2014.

dPLS19. Rafael del Pino, Vadim Lyubashevsky, and Gregor Seiler. Short discrete log proofs for fhe and ring-lwe ciphertexts. Cryptology ePrint Archive, Report 2019/057, 2019. `https://eprint.iacr.org/2019/057`, to appear at PKC 2019K.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference*, 2012.

FS86. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86*, 1986.

GMO16. Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX Security Symposium*, 2016.

GMR89. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1), 1989.

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. ACM, 2007.

KKW18. Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. 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. Akinori Kawachi, Keisuke Tanaka, and Keita Xagawa. Concurrently secure identification schemes based on the worst-case hardness of lattice problems. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2008.

LLNW17. Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Zero-knowledge arguments for lattice-based prfs and applications to e-cash. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China*, 2017.

LN17. Yehuda Lindell and Ariel Nof. 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. San Ling, Khoa Nguyen, Damien Stehlé, and Huaxiong Wang. Improved zero-knowledge proofs of knowledge for the isis problem, and applications. In *Public-Key Cryptography–PKC 2013*. Springer, 2013.

Lyu09. Vadim Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*. Springer Berlin Heidelberg, 2009.

Lyu12. Vadim Lyubashevsky. Lattice signatures without trapdoors. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012.

MBKM19. Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updateable structured reference strings. Cryptology ePrint Archive, Report 2019/099, 2019. `https://eprint.iacr.org/2019/099`.

PHGR16. Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. *Communications of the ACM*, 59(2), 2016.

Ste96. Jacques Stern. A new paradigm for public key identification. *IEEE Transactions on Information Theory*, 42(6), 1996.

WTS$^+$18. Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings*, 2018.