ZKBoo: Faster Zero-Knowledge for Boolean Circuits

Irene Giacomelli Jesper Madsen Claudio Orlandi

Computer Science Department, Aarhus University,
{giacomelli,sobuno,orlandi}@cs.au.dk

Abstract. In this paper we describe ZKBoo[1], a proposal for practically efficient zero-knowledge arguments especially tailored for Boolean circuits and report on a proof-of-concept implementation. As an highlight, we can generate (resp. verify) a non-interactive proof for the SHA-1 circuit in approximately 13ms (resp. 5ms), with a proof size of 444KB.
Our techniques are based on the "MPC-in-the-head" approach to zero-knowledge of Ishai et al. (IKOS), which has been successfully used to achieve significant asymptotic improvements. Our contributions include:
∘ A thorough analysis of the different variants of IKOS, which highlights their pro and cons for practically relevant soundness parameters;
∘ A generalization and simplification of their approach, which leads to faster Σ-protocols (that can be made non-interactive using the Fiat-Shamir heuristic) for statements of the form "I know x such that $y = \phi(x)$" (where ϕ is a circuit and y a public value);
∘ A case study, where we provide explicit protocols, implementations and benchmarking of zero-knowledge protocols for the SHA-1 and SHA-256 circuits;

1 Introduction

Since their introduction in the 80s [GMR85], zero-knowledge (ZK) arguments have been one of the main building blocks in the design of complex cryptographic protocols. However, due to the lack of practically efficient solutions for proving generic statements, their application in real-world systems is very limited. In particular, while there is a large body of work considering the efficiency of ZK protocols for algebraic languages (following the seminal work of Schnorr for discrete logarithm [Sch89]), things are quite different when it comes to general purpose ZK.

A notable exception is the recent line of work on *succinct non-interactive arguments of knowledge* (SNARKs) (*e.g.* Pinocchio [PHGR13], `libsnark` [BCTV14], etc.). SNARKs are an extremely useful tool when the size of the proof and the verification time matters: SNARKs are less than 300 bytes and can be verified in the order of 5ms, which makes them perfect for applications such as ZeroCash [BCG$^+$14]. However, on the negative side, SNARKs require very large parameters (which must be generated in a trusted way) and the time to generate proofs are prohibitive for many applications. As an example, the running time of the prover for generating a proof for SHA-1 is in the order of 10 seconds. There is an inherent reason for this inefficiency: current SNARKs technology requires to perform expensive operations (in pairing friendly groups) for each gate in the circuit.

Jawurek et al. [JKO13] proposed a different approach to efficient ZK, namely using garbled circuits (GC). Using GC, it is possible to prove any statement (expressed as a Boolean circuit) using only a (low) constant number of symmetric key operations per gate in the circuit, thus decreasing the proving time by more than an order of magnitude. On the flip-side, GC-based ZK are inherently interactive, and they still require a few public-key operations (used for implementing the necessary oblivious transfers).

In this paper we describe efficient ZK protocols for circuits based on the "MPC-in-the-head" paradigm of Ishai et al. [IKOS07] (IKOS). In IKOS, a prover simulates an MPC protocol between a number of "virtual" servers and then commits to the views and internal state of the individual servers. Now the verifier challenges the prover by asking to open a subset of these commitments. The privacy guarantee of the underlying MPC

[1] Sounds like Peekaboo.

protocol guarantees that observing the state of a (sufficiently small) subset of servers does not reveal any information. At the same time, the correctness of the MPC protocol guarantees that if the prover tries to prove a false statement, then the joint views of some of the server must necessarily be inconsistent, and the verifier can efficiently check that. By plugging different MPC protocols into this approach, [IKOS07] shows how to construct ZK protocols with good asymptotic properties. However, to the best of our knowledge, no one has yet investigated whether the IKOS approach can be used to construct practically efficient ZK protocols. This paper is a first step in this direction.

First (in Section 3), we describe the different variants of the IKOS framework. IKOS presents two strategies to achieve a negligible soundness error: either repeating a passive secure MPC protocol with few parties, or using a single instance of an active secure MPC protocol with a large number of parties. While IKOS only provides asymptotic estimates of the soundness parameters, we concretely estimate the soundness of IKOS with different kind of MPC protocols and show that, if one is interested in a (reasonable) soundness error of 2^{-80}, then the version of IKOS without repetition does not (unfortunately) lead to any practical advantage. Then (in Section 4) we present a new interpretation of the IKOS framework when instantiated with a 2-private 3-party version of the GMW [GMW87] protocol, where each pair of parties is connected with an OT-channel. We observe that in general the OT-channels can be replaced with arbitrary 2-party functionalities. Since those ideal functionalities do not have to be implemented using cryptographic protocols (remember, they are executed between pair of virtual servers in a simulation performed by the prover), this increases the degrees of freedom of the protocol designer and allows to construct more efficient MPC protocols (or, as we prefer to call them, functions decompositions) that can be used for constructing ZK protocols. (Note that this class of protocol has not been studied before, since it does not lead to any advantage in the standard MPC setting, and therefore we expect future work to improve on our approach by designing better MPC protocols for this special setting.) All resulting protocols are Σ-protocols (3-move honest-verifier zero-knowledge protocols with special soundness) which can therefore be made *non-interactive* in the random oracle model using the Fiat-Shamir heuristic.

Finally (in Section 5) we describe how our approach can be used to construct very efficient ZK protocols for proving knowledge of preimages for SHA-1 and SHA-256. The resulting proofs are incredibly efficient: the verification time is essentially the same as the verification time for SNARKs, but the prover runs approximately 1000 times faster. On the negative side the size of our proofs scales linearly with the circuit size, but we believe that in some applications this is a desirable trade-off.

2 Preliminaries

Standard notations: For an integer n, we write $[n] = \{1, 2, \ldots, n\}$ and, given $A \subseteq [n]$, $|A|$ denotes the cardinality of A. We say that a function ϵ is *negligible* in n, $\epsilon(n) = negl(n)$, if for every polynomial p there exists a constant c such that $\epsilon(n) < \frac{1}{p(n)}$ when $n > c$. Given two random variables X ad Y with support S, the statistical distance between X and Y is defined as $\mathsf{SD}(X, Y) = \frac{1}{2}\sum_{i \in S} \mid \Pr[\,X = i\,] - \Pr[\,Y = i\,] \mid$. Two families $X = \{X_{\mathbf{k}}\}$ and $Y = \{Y_{\mathbf{k}}\}$, $\mathbf{k} \in \{0,1\}^*$ of random variables are said to be *statistically indistinguishable* if there exists a negligible function $\epsilon(\cdot)$ such that for every $\mathbf{k} \in \{0,1\}^*$, $\mathsf{SD}(X_{\mathbf{k}}, Y_{\mathbf{k}}) \le \epsilon(|\mathbf{k}|)$. They are said to be *computationally indistinguishable* if for every efficient non-uniform distinguisher D there exists a negligible function $\epsilon(\cdot)$ such that for every $\mathbf{k} \in \{0,1\}^*$, $\mid Pr[D(X_{\mathbf{k}}) = 1] - Pr[D(Y_{\mathbf{k}}) = 1] \le \epsilon(|\mathbf{k}|)$.

2.1 Multi-Party Computation (MPC)

Consider a public function $f : (\{0,1\}^k)^n \to \{0,1\}^\ell$ and let $P_1, \ldots, P_n$ be n players modelled as PPT machines. Each player P_i holds the value $\mathbf{x}_i \in \{0,1\}^k$ and wants to compute the value $\mathbf{y} = f(\mathbf{x})$ with $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ while keeping his input private. The players can communicate among them using point-to-point secure channels $\mathsf{CH}_{i,j}$ in the synchronous model. These can be classical secure channels (*i.e.* encrypted channels) or more powerful channels (*e.g.* OT-channel [EGL85,Rab05]). If necessary, we also allow the players to use a broadcast channel. To achieve their goal, the players jointly run a n-party MPC protocol Π_f. The latter is a protocol for n players that is specified via the next-message functions: there are several rounds of communication and in each round the player P_i sends into the channel $\mathsf{CH}_{i,j}$ (or in the broadcast channel) a message that is computed as a deterministic function of the internal state of P_i (his initial input $\mathbf{x}_i$ and his random tape $\mathbf{k}_i$) and the messages that P_i has received in the previous rounds of communications. The *view* of the player P_j, denoted by $\mathrm{View}_{P_j}(\mathbf{x})$, is defined as the concatenation of the private input $\mathbf{x}_j$, the random tape $\mathbf{k}_j$ and all the messages received by P_j during the execution of Π_f. Each channel $\mathsf{CH}_{i,j}$ defines a relation of *consistency* between views. For instance, in a plain channel *two views are consistent if* the messages reported in $\mathrm{View}_{P_j}(\mathbf{x})$ as incoming from P_j are equal to the outgoing message implied by $\mathrm{View}_{P_i}(\mathbf{x})$ $(i \neq j)$. More powerful channels (such as OT channels), are defined via some function φ and we say that *two views are consistent if* the view of the sender contains an input $\mathbf{x}$ to the channel and the view of the receiver contains an input $\mathbf{y}$ and an output $\mathbf{z}$ such that $\mathbf{z} = \varphi(\mathbf{x}, \mathbf{y})$. For instance, in OT channels $\mathbf{x} = (m_0, m_1)$, $\mathbf{y}$ is a bit and $\mathbf{z} = m_{\mathbf{y}}$.

Finally, the output $\mathbf{y}$ can be computed from any of the view $\mathrm{View}_{P_i}(\mathbf{x})$, *i.e.* there are n functions $\Pi_{f,1}, \ldots, \Pi_{f,n}$ such that $\mathbf{y} = \Pi_{f,i}(\mathrm{View}_{P_i}(\mathbf{x}))$ for all $i \in [n]$. In order to be private, the protocol Π_f needs to be designed in such a way that a curious player P_i can not infer information about $\mathbf{x}_j$ with $j \neq i$ from his view $\mathrm{View}_{P_i}(\mathbf{x})$. An additional security property, robustness, assures that a cheating player P_i (who may not follow the instructions in the protocol) can not mislead the honest players, who still compute the correct output $\mathbf{y}$. More precisely, we have the following definition.

Definition 1. – ***(Correctness)*** *We say that the protocol Π_f realizes f with perfect (resp. statistical) correctness if for any input $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, it holds that* $\Pr[\, f(\mathbf{x}) \neq \Pi_{f,i}(\mathrm{View}_{P_i}(\mathbf{x}))\,] = 0$ *(resp. negligible) for all $i \in [n]$. The probability is over the choice of the random tapes $\mathbf{k}_i$.*
– ***(Privacy)*** *Let $1 \leq t < n$, the protocol Π_f has* perfect t-privacy *if it is correct and for all $A \subsetneq [n]$ satisfying $|A| \leq t$ there exists a PPT algorithm S_A such that the joint views* $(\mathrm{View}_{P_i}(\mathbf{x}))_{i\in A}$ *have the same distribution as $S_A(f, (\mathbf{x}_i)_{i\in A}, \mathbf{y})$, for all* $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$.
We will speak about statistical (resp. computational) t-privacy if the two distributions $S_A(f, (\mathbf{x}_i)_{i\in A}, \mathbf{y})$ and $(\mathrm{View}_{P_i}(\mathbf{x}))_{i\in A}$ *are statistically (resp. computationally) indistinguishable.*
– ***(Robustness)*** *Let $0 \leq r < n$, the protocol Π_f has perfect (resp. statistical) r-robustness if it is correct and for all $A \subsetneq [n]$ satisfying $|A| \leq r$ even assuming that all the players in A have been arbitrarily corrupted, then* $\Pr[\, f(\mathbf{x}) \neq \Pi_{f,i}(\mathrm{View}_{P_i}(\mathbf{x}))\,] = 0$ *(resp. negligible) for all $i \in A^c$.*

3 Zero Knowledge

In this section we review the notion of zero-knowledge and Σ-protocols, we review the IKOS construction [IKOS07] for zero-knowledge, and we discuss different possible instantiations.

3.1 Definitions

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation representing some computational problem (*e.g.* $R = \{(\mathbf{y}, \mathbf{x}) \mid \mathbf{y} = \text{SHA-256}(\mathbf{x})\}$). We will interpret R as a binary function from $\{0,1\}^* \times \{0,1\}^*$ to $\{0,1\}$ (*i.e.* $R(\mathbf{y}, \mathbf{x}) = 1 \Leftrightarrow (\mathbf{y}, \mathbf{x}) \in R$) and we will assume that:

– $\forall\, \mathbf{y}$ and $\forall\, \mathbf{x}$, $R(\mathbf{y}, \mathbf{x})$ can be computed in polynomial-time by a probabilistic Turing machine;
– there exists a polynomial p such that if $R(\mathbf{y}, \mathbf{x}) = 1$ then the length of $\mathbf{x}$ is less or equal to $p(|\mathbf{y}|)$.

Such relation is called NP relation. With L we indicate the set of the yes-instances of the relation R, *i.e.* $L = \{\mathbf{y} \mid \exists\, \mathbf{x} \text{ s.t. } R(\mathbf{y}, \mathbf{x}) = 1\}$.

An *argument* for L is a cryptographic protocols between two players: the prover P and the verifier V with the following features. We assume that both P and V are probabilistic polynomial time (PPT) machines and that they know $\mathbf{y}$, an instance of the relation R. The situation is that P wants to convince V that $\mathbf{y} \in L$. This clearly makes sense only if the prover has some advantage over the verifier. Thus, we allow the prover to have an extra private input (for example P knows $\mathbf{x}$ such that $R(\mathbf{y}, \mathbf{x}) = 1$). The protocol is described by instructions for the players and has different rounds of communication. At the end of the protocol, the verifier outputs accept if he is convinced or reject otherwise. If $\mathbf{y} \in L$, we require that an honest verifier convinces an honest prover with probability 1 (the protocol is complete). On the other hand, we say that the protocol has *soundness error* ϵ if for all $\mathbf{y} \notin L$ $\Pr[\,V(\mathbf{y}) = \mathsf{accept}\,] \leq \epsilon$, no matter what the prover does. In other words, ϵ is an upperbound of the probability that a cheating prover makes an honest verifier outputs accept for a false instance.

However, in many interesting cryptographic applications, the language L is trivial and therefore the soundness property gives absolutely no guarantees: for every string $\mathbf{y}$ there exist a $\mathbf{x}$ s.t., $\mathbf{y}$ =SHA-256($\mathbf{x}$). In this case we need a stronger property, namely *proof-of-knowledge* (PoK), which informally states that the verifier should output accept only if the prover *knows* the value $\mathbf{x}$.

Finally, ZK protocols get their name from the *zero-knowledge* property: Here, we want to express the requirement that whatever strategy a cheating verifier follows, he learns nothing except for the truth of the prover's claim. In particular, he can not obtain information about the private input of P. This is captured using the simulation-paradigm and saying that the messages received by the verifier during the protocol can be efficiently simulated by knowing only the public input $\mathbf{y}$. More precisely, we have the following requirement: for any corrupted PPT verifier V^*, there is a PPT algorithm S (the "simulator") with access to V^* such that the output of $S(\mathbf{y})$ and the real conversation between P and V^* on input $\mathbf{y}$ are indistinguishable.

In the rest of the paper we will be concerned with public-coin two-party protocols with a specific communication pattern as known as Σ-protocols.

Definition 2 (Σ-protocol). *A protocol Π_R between two players P and V is a* Sigma Protocol *for the relation R if it satisfies the following conditions:*

- *Π_R has the following communication pattern:*
  1. (Commit) *P sends a first message $\mathbf{a}$ to V;*
  2. (Challenge) *V sends a random element $\mathbf{e}$ to P;*
  3. (Prove) *P replies with a second message $\mathbf{z}$.*
- *(**Completeness**) If both players P and V are honest and $\mathbf{y} \in L$, then* $\Pr[\,(P,V)(\mathbf{y}) = \mathsf{accept}\,] = 1$*;*
- *(s-**special soundness**) Fixed $\mathbf{y} \in L$, a witness $\mathbf{x}$ can be efficiently computed from s accepting conversations $(\mathbf{a}, \mathbf{e}_i, \mathbf{z}_i)$ for $i \in [s]$ with $\mathbf{e}_i \neq \mathbf{e}_j$ if $i \neq j$;*
- *(**Special honest-verifier ZK**) There exists a PPT simulator S such that on input $\mathbf{y} \in L$ and $\mathbf{e}$ outputs a triple $(\mathbf{a}', \mathbf{e}, \mathbf{z}')$ with same probability distribution of real conversations $(\mathbf{a}, \mathbf{e}, \mathbf{z})$ of the protocol.*

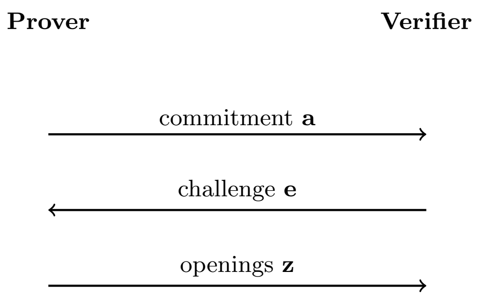

Fig. 1. The communication pattern of a Σ-protocol.

Σ-protocols have several properties (*e.g.* parallel composition, witness indistinguishability) that make them a useful building block for many other cryptographic primitives (identification schemes, signatures, etc). See [Dam] or [HL10, Chapter 6] for more details on this. Here we are mainly interested in the following facts: First, Σ-protocols are public-coin protocols and thus they can be made non-interactive in the random oracle model using the Fiat-Shamir heuristic [FS86]. Second, there exist efficient transformations from Σ-protocols to fully-fledged ZK and PoK: indeed, it is possible to efficiently transform a Σ-protocol into a zero-knowledge argument (resp. zero-knowledge proof of knowledge) with the addition of one additional round (resp. two additional rounds). Note finally that s-special soundness implies a bound of $1 - s^{-1}$ on the soundness error of the protocol: if $\mathbf{y} \notin L$, then there exist no $\mathbf{x}$ s.t.$R(\mathbf{x}, \mathbf{y}) = 1$, and therefore fixed any $\mathbf{a}$ there must be at least one challenge e such that no accepting conversation for that e exists.

3.2 IKOS Construction

In 2007 Ishai *et al.* show how to use any MPC protocol and the commitment-hybrid (Com) model[2] to obtain a ZK proof for an arbitrary NP relation R with asymptotically small soundness error. Here we briefly recall their construction and moreover we explicitly analyse its soundness error.

Let Π_f be an MPC protocol that realizes any n-party function f with perfect correctness (Definition 1). Depending on the features of Π_f (privacy, robustness, communication channels used), [IKOS07] presents slightly different ZK protocols. However, the general structure is always the same and is the structure of a Σ-protocol, see Figure 1. The high-level idea is the following: assume that $\mathbf{y} \in L$ is the public input of the ZK protocol, while $\mathbf{x}$ is the private input of the prover (*i.e.* $R(\mathbf{y}, \mathbf{x}) = 1$). The prover first takes n random values $\mathbf{x}_1, \ldots, \mathbf{x}_n$ such that $\mathbf{x} = \mathbf{x}_1 \oplus \cdots \oplus \mathbf{x}_n$, then he considers the n-input function $f_{\mathbf{y}}$ defined as

$$f_{\mathbf{y}}(\mathbf{x}_1, \cdots, \mathbf{x}_n) := R(\mathbf{y}, \mathbf{x}_1 \oplus \cdots \oplus \mathbf{x}_n)$$

and emulates "in his head" the protocol $\Pi_{f_{\mathbf{y}}}$ on inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n$. After the emulation, he computes the commitments to each of the n produced views (*i.e.* $\mathsf{Com}(\mathrm{View}_{P_i}(\mathbf{x}))$ for $i = 1, \ldots, n$) and sends them to the verifier. After having received all the commitments, the verifier challenges the prover to open some of them (*i.e.* the challenge is a random subset of $[n]$ of a given size). Finally, the prover opens the requested commitments and the verifier outputs accept if and only if all the opened views are consistent with each other and with output 1.

Here we focus on the ZK protocols presented in [IKOS07] that assume a perfectly correct (and eventually perfectly robust) MPC protocol and we collect them in two versions. Version 1 considers the case of an MPC protocol with t-privacy and perfect r-robustness with $t > 1$.[3] Version 2 shows that 2-privacy is not necessary condition and indeed considers the case of an MPC protocol with 1-privacy only.

Version 1: Let t and r be two integers, $2 \leq t < n$ and $0 \leq r \leq t$. We assume that the protocol $\Pi_{f_{\mathbf{y}}}$ is perfectly correct and satisfies two more properties: perfect, statistical or computational t-privacy and perfect r-robustness. In this version of the IKOS protocol (Figure 2) the verifier is allowed to ask for the openings of t of the commitments $\mathsf{Com}(\mathrm{View}_{P_i}(\mathbf{x}))$ he received. In this way, the zero-knowledge property follows easily by the t-privacy of the protocol $\Pi_{f_{\mathbf{y}}}$.

For the analysis of the soundness error of this protocol we use the so-called *inconsistency graph* G. Given an execution of $\Pi_{f_{\mathbf{y}}}$, the graph G has n nodes and there is an edge (i, j) if and only if the views of the players P_i and P_j are inconsistent. Assume that $\mathbf{y} \notin L$ and that the execution of $\Pi_{f_{\mathbf{y}}}$ is not a correct one (otherwise $\Pr[\, V(\mathbf{y}) = \mathsf{accept}\,] = 0$ because of the checks in step 1 of the procedure **Verify**). Then we have two cases:

1. There is in G a vertex cover set[4] B of size at most r. Intuitively, this means that in the current execution of $\Pi_{f_{\mathbf{y}}}$ only the players in B have been actively corrupted. Indeed, if we remove the nodes in B, we obtain a graph without edges. That is, all the players not in B have views consistent among them and we can consider these players honest. Since the size of B is less or equal to the parameter r, the robustness property assures that for all the players not in B (honest players) the view implies a 0 output (the correct

[2] In the *commitment-hybrid model* the two parties have access to an idealized implementation of commitments, which can be imagined as a trusted third party which stores the messages of the sender and only reveals them if told so by the sender.

[3] This is a generalization of [IKOS07] as they only consider the case $t = r$.

[4] B is a vertex cover set for the graph G if each edge in G is incident to at least 1 node in B.

IKOS Protocol (Version 1)

The verifier and the prover have input $\mathbf{y} \in L$. The prover knows $\mathbf{x}$ such that $R(\mathbf{y}, \mathbf{x}) = 1$. A perfectly correct and t-private n-party MPC protocol $\Pi_{f_{\mathbf{y}}}$ is given $(2 \leq t < n)$.

Commit: The prover does the following:
1. Sample random vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$ s.t. $\mathbf{x}_1 \oplus \cdots \oplus \mathbf{x}_n = \mathbf{x}$;
2. Run $\Pi_{f_{\mathbf{y}}}(\mathbf{x}_1, \dots, \mathbf{x}_n)$ and obtain the views $\mathbf{w}_i = \mathrm{View}_{P_i}(\mathbf{x})$ for all $i \in [n]$;
3. Commits to $\mathbf{w}_1, \dots, \mathbf{w}_n$.

Prove: The verifier chooses a subset $E \subseteq [n]$ such that $|E| = t$ and sends it to the prover. The prover reveals the value $\mathbf{w}_e$ for all $e \in E$.

Verify: The verifier runs the following checks:
1. If $\exists\, e \in E$ s.t. $\Pi_{f,e}(\mathrm{View}_{P_e}(\mathbf{x})) \neq 1$, output reject;
2. If $\exists\, \{i, j\} \subset E$ s.t. $\mathrm{View}_{P_i}(\mathbf{x})$ is not consistent with $\mathrm{View}_{P_j}(\mathbf{x})$, output reject;
3. Output accept;

Fig. 2. The IKOS zero-knowledge protocol for the relation R in the commitment-hybrid model.

output of the protocol $\Pi_{f_{\mathbf{y}}}$). The probability that the verifier will not see one of these views choosing t of them uniformly at random is less or equal to[5]

$$p_1(n, t, r) = \binom{r}{t}\binom{n}{t}^{-1}$$

2. If the size of the minimum vertex cover is $> r$, then the graph G has a matching[6] of size $> r/2$. The probability that the verifier accepts the wrong proof is equal to the probability that between the t nodes that he chooses there are no edges of G and this is less or equal to the probability that there are no edges from the matching. Clearly, this probability reaches the maximum when the matching is the smallest possible, that is it has size $k = \lfloor r/2 \rfloor + 1$. In this situation the aforementioned probability is

$$p_2(n, t, r) = \begin{cases} 0 & \text{otherwise} \\ \left(\sum_{j=0}^{k} 2^j \binom{k}{j}\binom{n-2k}{t-j}\right)\binom{n}{t}^{-1} & \text{if } n - 2k > 0 \end{cases}$$

In general, the soundness error is equal to the value $p(n, t, r) = \max\{p_1(n, t, r), p_2(n, t, r)\}$.

Version 2: A second version of the protocol was proposed in [IKOS09] to show that 2-privacy is not a necessary condition for the IKOS construction. In other words, we can construct ZK proofs in from 1-private MPC protocols. Notice that in this case the MPC protocol is allowed to use only standard point-to-point secure channels. The idea of the construction is very similar to the previous one, but now the prover commits to all the $\binom{n}{2}$ channels in addition to committing to the n views. The verifier picks a random $i \in [n]$ and challenges the prover to open the view of the player P_i and all the $n-1$ channels CH_{ij} incident to him. Finally, the verifier accepts if the opened view is consistent with the channels and with the output 1. Again, the ZK property follows from the privacy property of the MPC protocol: the information revealed to the verifier is implied by the view of a single player. To compute the soundness error in this case, observe that for any incorrect execution of $\Pi_{f_{\mathbf{y}}}$ there is at least one player P_i such that $\mathrm{View}_{P_i}(\mathbf{x})$ is inconsistent with a channel CH_{ij}. The probability $\Pr[\, V(\mathbf{y}) = \mathsf{accept}\,]$ is less or equal to the probability that V does not choose this index i. Therefore, the soundness error of this version is $1 - 1/n$.

[5] $\binom{r}{t}$ is 0 if $r < t$.
[6] A matching is a set of edges without common nodes.

3.3 Our choice of version and parameters

In this section we discuss and motivate some of our design choices.

Which MPC protocol? As discussed, IKOS can be instantiated with a large number of MPC protocols. In particular, using MPC protocols with good asymptotic properties (such as [DI06,DIK10], etc.), one can obtain ZK protocols with equally good asymptotic properties. However in this paper we are concerned with concrete, constant size circuits, and we do not want to put any restriction on the shape or width of the circuits. Thus, the best two choices are BGW [BGW88] style protocols with $t = r = \lfloor \frac{n-1}{3} \rfloor$ which use simple point-to-point channels and GMW [GMW87] style protocols with $t = n-1, r = 0$ which use OT channels between each pair of parties. Then we have the following two cases:

1. (GMW [GMW87]:) In this case the soundness error is $\frac{2}{n}$ and we open $n-1$ views. Note that in these protocols each party must communicate with every other party, thus the size of the proof for soundness $2^{-\sigma}$ is given by
$$c \cdot \frac{(n-1)^2}{\log_2(n) - 1} \cdot \sigma$$
where c is a constant which depends on the exact protocol. It is easy to see that the function grows with n and therefore smallest proofs are achieved with $n = 3$. Looking ahead, our protocol in Section 4 has $c = 1/2$ and $\sigma = 80$ and therefore the size of the proof is 274 bits per multiplication gate.
2. (BGW [BGW88]:) In this case the soundness error is given by $p_2(n, \lfloor \frac{n-1}{3} \rfloor, \lfloor \frac{n-1}{3} \rfloor)$. To get soundness error $\leq 2^{-80}$, we get that $n \geq 1122$ and therefore the number of opened views is $\lfloor \frac{n-1}{3} \rfloor = 373$. Thus, even if each party only had to store a single bit for each multiplication gate, the size of the proof would already be larger than in the previous case.
3. (Future Work:) Our analysis shows that using an MPC protocol with $t = r = \lceil \frac{2}{3}n \rceil$ it would be enough to use $(n, t, r) = (92, 64, 64)$ to achieve soundness 2^{-80}. The existence of such a protocol, where in addition each party only needs to store ≤ 4 bits per multiplication gate, would give rise to ZK proofs of size smaller than the one we construct. We are not aware of any such protocols, however we cannot rule out their existence. In particular, we note that such protocols have *not* been considered in the literature, since they give rise to poor MPC protocols in practice (note that such a protocol necessarily uses advanced channels, which in the standard MPC protocol need to be implemented using expensive cryptographic operations), and we believe that the quest for "MPC protocols" optimized for the ZK applications has just begun. Figure 3 shows how, for a fixed number of parties n, the soundness error decreases as a function of $t = r$. Note that the soundness error for $\frac{2}{3}n$ is much smaller than $\frac{1}{3}n$.

Why only perfect correctness and robustness? [IKOS07] presented also two extensions of the basic construction that allow to use MPC protocol with statistical correctness or with statistical robustness, but we are not considering those cases here for two reasons: first, the resulting ZK protocols have higher round complexity (and are therefore not Σ-protocols); second, perfectly secure MPC protocols are *more efficient*: practically efficient MPC protocols which only achieve statistical security (even when allowing arbitrary two-party channels, such as in [BDOZ11,NNOB12]) require parties to store *tags* or *MACs* together with their shares, and to make sure that the statistical error is negligibly small these tags need to be at least as long as the security parameter[7], whereas in perfectly secure MPC protocols the share size can be made constant.

Why not Version 2? Note that the soundness error of Version 1 with $(n, t, r) = (3, 2, 0)$ is the same as the soundness error of Version 2 with $(n, t, r) = (3, 1, 0)$, thus the number of required rounds is exactly the same. However (i) Version 2 requires to compute and open more commitments and (ii) Version 2 only works with plain channels, while Version 1 allows to use arbitrary channels which helps in constructing more efficient protocols.

[7] This can be avoided for SIMD computations [DZ13]

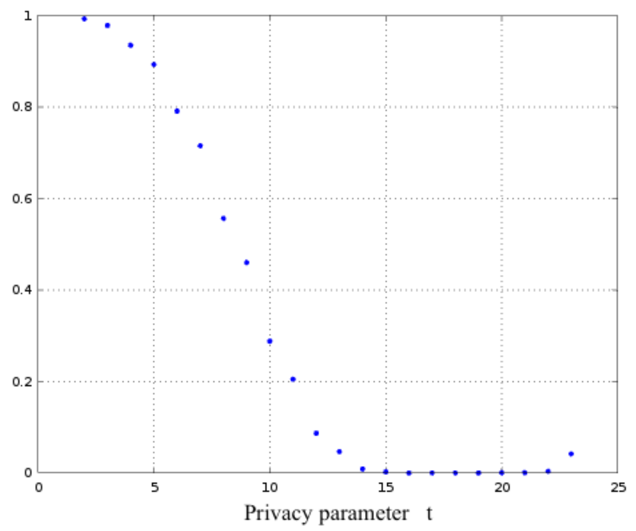

Fig. 3. The soundness error $p(n, t, r)$ in function of t when $t = r$ and $n = 24$.

4 Generalizing IKOS

This section contains a generalized and optimized version of the IKOS protocol that works for any relation defined by a function, $\phi : X \to Y$ which can be decomposed in the "right way". In particular, in Section 4.2 we will describe a ZK Σ-protocol for the relation R_ϕ defined by $R_\phi(\mathbf{y}, \mathbf{x}) = 1 \Leftrightarrow \phi(\mathbf{x}) = \mathbf{y}$, while the decomposition used to construct it is formalized in the following section.

Protocol Π^*_ϕ

Let $\phi : X \to Y$ be a function and $\mathcal{D}$ a related $(2, 3)$-decomposition as defined in Definition 3.

Input: $\mathbf{x} \in X$

```
1. Sample random tapes $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3$;
2. Compute $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \leftarrow \mathsf{Share}(\mathbf{x}; \mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3)$;
3. Let $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$ be vectors with $N + 1$ entries;
   – Initialize $\mathbf{w}_i[0] = \mathbf{x}_i$ for all $i \in \{1, 2, 3\}$;
   – For $j = 1, \ldots, N$, compute:
       • For $i = 1, 2, 3$, compute
             $\mathbf{w}_i[j] = \phi_i^{(j)}\big((\mathbf{w}_m[0..j-1], \mathbf{k}_m)_{m \in \{i,i+1\}}\big)$
4. Compute $\mathbf{y}_i = \mathsf{Output}_i(\mathbf{w}_i, \mathbf{k}_i)$ for $i \in \{1, 2, 3\}$;
5. Compute $\mathbf{y} = \mathsf{Rec}(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3)$;
```

Output: $\mathbf{y} \in Y$

Fig. 4. Given a correct decomposition $\mathcal{D}$, the protocol Π^*_ϕ can be used to evaluate the function ϕ.

4.1 (2, 3)-Function Decomposition

Given an arbitrary function $\phi : X \to Y$ and an input value $\mathbf{x} \in X$ we want to compute the value $\phi(\mathbf{x})$ splitting the computation in 3 branches such that the values computed in 2 branches reveals no information about the input $\mathbf{x}$. In order to achieve this, we start by "splitting" the value $\mathbf{x}$ in three values $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ (called *input shares*) using a surjective function that we indicate with Share. Those input shares as well as all the intermediate values are stored in 3 string $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$ called the *views*. More precisely, $\mathbf{w}_i$ contains the values computed in the computation branch i. In order to achieve the goal and compute the value $\mathbf{y} = \phi(\mathbf{x})$, we use a finite family of efficiently computable functions that we indicate with $\mathcal{F} = \bigcup_{j=1}^{N} \{\phi_1^{(j)}, \phi_2^{(j)}, \phi_3^{(j)}\}$. The function $\phi_m^{(j)}$ takes as inputs specific values from the views $\mathbf{w}_m, \mathbf{w}_{m+1}$ with $m = \{1, 2, 3\}$ and where $3 + 1 = 1$. The functions are used in the following way: we use functions $\phi_1^{(1)}, \phi_2^{(1)}, \phi_3^{(1)}$ to compute the next value to be stored in each view $\mathbf{w}_m$: The function $\phi_m^{(1)}$ takes as input $\mathbf{w}_m, \mathbf{w}_{m+1}$ (which at this point contain only the shares $\mathbf{x}_m, \mathbf{x}_{m+1}$) and outputs one value which is saved in position 1 of the views $\mathbf{w}_m$. We continue like this for all N functions, with the difference that in step $j > 1$, the function $\phi_m^{(j)}$ can receive as input (any subset of) the current views $\mathbf{w}_m, \mathbf{w}_{m+1}$. The initial function Share and all subfunctions $\phi_m^{(j)}$ are allowed to be randomized, and they get their coins from $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3$, three random tapes which correspond to the three branches. Finally, after the N steps described, the 3 functions $\mathsf{Output}_1, \mathsf{Output}_2, \mathsf{Output}_3$ are used to compute the values $\mathbf{y}_i = \mathsf{Output}_i(\mathbf{w}_i)$ that we call *output shares*. From these three values we compute the final output $\mathbf{y} = \phi(\mathbf{x})$ using the function Rec. The entire procedure is described in detail in Figure 4 (Protocol Π_ϕ^*).

Definition 3. *A* (2, 3)-decomposition *for the function ϕ is the set of functions*

$$\mathcal{D} = \{\mathsf{Share}, \mathsf{Output}_1, \mathsf{Output}_2, \mathsf{Output}_3, \mathsf{Rec}\} \cup \mathcal{F}$$

such that Share *is a surjective function and* $\phi_m^{(j)}$, Output_i *and* Rec *are functions as described before. Let Π_ϕ^* be the algorithm described in Figure 4, we have the following definitions.*

- *(**Correctness**) We say that $\mathcal{D}$ is correct if* $\Pr[\,\phi(\mathbf{x}) = \Pi_\phi^*(\mathbf{x})\,] = 1$ *for all $\mathbf{x} \in X$. The probability is over the choice of the random tapes $\mathbf{k}_i$.*
- *(**Privacy**) We say that $\mathcal{D}$ has* 2*-privacy if it is correct and for all* $e \in [3]$ *there exists a PPT simulator S_e such that*

$$(\{\mathbf{k}_i, \mathbf{w}_i\}_{i \in \{e, e+1\}}, \mathbf{y}_{e+2}) \text{ and } S_e(\phi, \mathbf{y})$$

have the same probability distribution for all $\mathbf{x} \in X$.

The Linear Decomposition We present here an explicit example of a convenient (2,3)-decomposition. Let $\mathbb{Z}$ be an arbitrary finite ring such that $\phi : \mathbb{Z}^k \to \mathbb{Z}^\ell$ can be expressed by an arithmetic circuit over the ring using addition by constant, multiplication by constant, binary addition and binary multiplication gates[8]. The total number of gates in the circuit is N, the gates are labelled with indices in $[N]$. The *linear (2,3)-decomposition* of ϕ is defined as follows:

- $\mathsf{Share}^{\mathbb{Z}}(\mathbf{x}; \mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3)$ samples random $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ such that $\mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3$;
- The family $\mathcal{F}^{\mathbb{Z}} = \bigcup_{c=1}^{N} \{\phi_1^{(c)}, \phi_2^{(c)}, \phi_3^{(c)}\}$ is defined in the following way. Assume that the c-th gate has input wires coming from the gate number a and the gate number b (or only gate number a in the case of a unary gate), then the function $\phi_i^{(c)}$ is defined as follows: If the c-th gate is a ($\forall \alpha \in \mathbb{Z}$)
  - unary "add α" gate, then $\forall\, i \in [3]$:

$$\mathbf{w}_i[c] = \phi_i^{(c)}(\mathbf{w}_i[a]) = \begin{cases} \mathbf{w}_i[a] + \alpha \text{ if } i = 1 \\ \mathbf{w}_i[a] \text{ else} \end{cases}$$

[8] Note that Boolean circuits are a special case of this, with the XOR, AND and NOT gate.

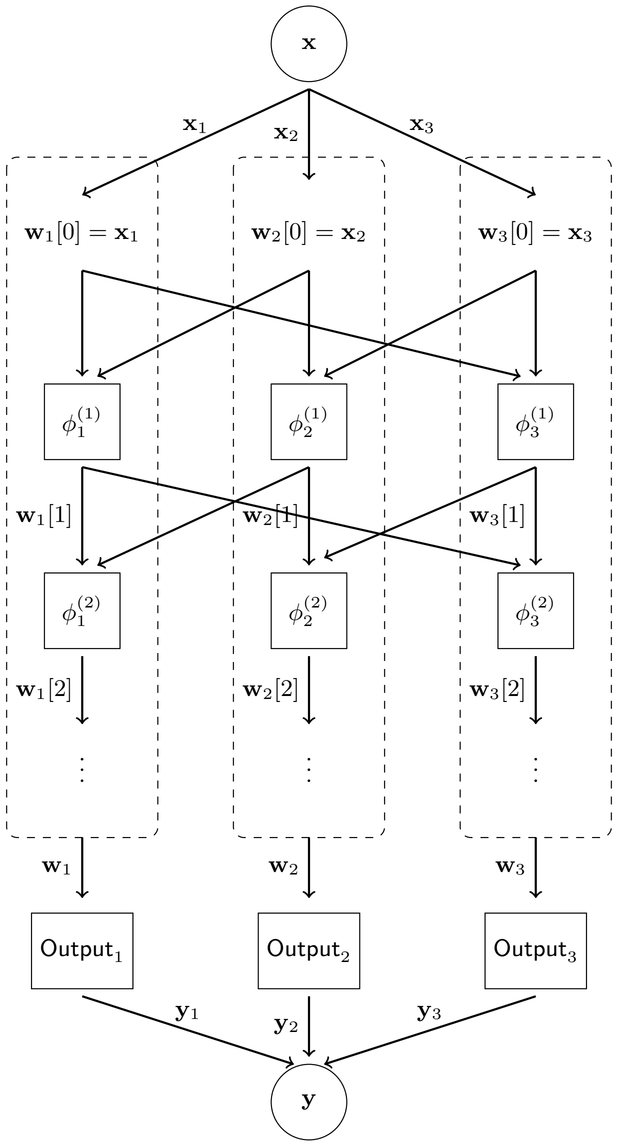

Fig. 5. Pictorial representation of a (2,3)-decomposition of the computation $\mathbf{y} = \phi(\mathbf{x})$ showing the three branches.

– unary "mult. α" gate, then $\forall\, i \in [3]$:

$$\mathbf{w}_i[c] = \phi_i^{(c)}(\mathbf{w}_i[a]) = \alpha \cdot \mathbf{w}_i[a]$$

– binary addition gate, then $\forall\, i \in [3]$:

$$\mathbf{w}_i[c] = \phi_i^{(c)}(\mathbf{w}_i[a], \mathbf{w}_i[b]) = (\mathbf{w}_i[a] + \mathbf{w}_i[b])$$

– binary multiplication gate, then $\forall\, i \in [3]$:

$$\begin{aligned} \mathbf{w}_i[c] = \phi_i^{(c)}\big(&\mathbf{w}_i[a,b], \mathbf{w}_{i+1}[a,b]\big) \\ &= \mathbf{w}_i[a] \cdot \mathbf{w}_i[b] + \mathbf{w}_{i+1}[a] \cdot \mathbf{w}_i[b] \\ &+ \mathbf{w}_i[a] \cdot \mathbf{w}_{i+1}[b] + R_i(c) - R_{i+1}(c) \end{aligned}$$

where $R_i(c)$ is a uniformly random function sampled using $\mathbf{k}_i$.
- For all $i \in [3]$, $\mathsf{Output}_i^{\mathbb{Z}}(\mathbf{w}_i, \mathbf{k}_i)$ simply selects all the shares of the output wires of the circuit;
- Finally, $\mathsf{Rec}^{\mathbb{Z}}(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3)$ outputs $\mathbf{y} = \mathbf{y}_1 + \mathbf{y}_2 + \mathbf{y}_3$

Proposition 1. *The decomposition $\mathcal{D}^{\mathbb{Z}} = \{\mathsf{Share}^{\mathbb{Z}}, \mathsf{Rec}^{\mathbb{Z}}, \mathsf{Output}_1^{\mathbb{Z}}, \mathsf{Output}_2^{\mathbb{Z}}, \mathsf{Output}_3^{\mathbb{Z}}\} \cup \mathcal{F}^{\mathbb{Z}}$ defined above is a (2,3)-decomposition. Moreover, the length of each view in $\mathcal{D}^{\mathbb{Z}}$ is $(k + N + \ell) \log |\mathbb{Z}| + \kappa$ bits.*

Correctness of the decomposition follows from inspection. Privacy can be shown by constructing an appropriate simulator as shown in Appendix A

In the linear decomposition just presented, the parameter N is equal to the total number of gates (unary and binary) in the circuit computing ϕ. It is easy to slightly modify the definition of the functions $\phi_i^{(c)}$ in $\mathcal{D}^{\mathbb{Z}}$ in such a way that N results equal to the number of multiplication gates only. In particular, note that the evaluation of *addition gates* (both unary and binary) only requires computation on values from the same branch, thus they can be embedded in a generalized multiplication gates which take as input arbitrary subsets of wires A, B, contains constants α, β, γ and computes the value:

$$\mathbf{w}[c] = \left(\sum_{a \in A} \alpha[a]\mathbf{w}[a]\right) \cdot \left(\sum_{b \in B} \beta[b]\mathbf{w}[b]\right) + \gamma$$

4.2 ZKBoo Protocol

Following the idea of [IKOS07], we turn a $(2,3)$-decomposition of a function ϕ into a zero-knowledge protocol for statements of the form "I know $\mathbf{x}$ such that $\phi(\mathbf{x}) = \mathbf{y}$". We indicate with L_ϕ the language $\{\mathbf{y} \mid \exists\, \mathbf{x} \text{ s.t. } \phi(\mathbf{x}) = \mathbf{y}\}$.

Assume that a (2,3)-decomposition of the function ϕ is known (see Section 4.1). The structure of the resulting protocol (Figure 6) is very similar to the structure of the IKOS protocol. If $\mathbf{y} \in L_\phi$ is the public input of the proof, then the prover P uses his private input $\mathbf{x}$ (with $\phi(\mathbf{x}) = \mathbf{y}$) to run "in his head" the protocol Π_ϕ^*. After the emulation of the protocol, P commits to each of the 3 produced views $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$. Now the verifier challenges the prover to open 2 of the commitments. Finally, the verifier accepts if the opened views are consistent with the decomposition used and with output $\mathbf{y}$.

Proposition 2. *The ZKBoo protocol (Figure 6) is a Σ-protocol for the relation R_ϕ with* 3*-special soundness.*

Proof. Clearly, the ZKBoo protocol has the right communication pattern and it is complete given that the decomposition $\mathcal{D}$ is correct. Moreover, the protocol satisfies the 3-special soundness property: consider 3 accepting conversations $(\mathbf{a}, i, \mathbf{z}_i)$, $i \in [3]$: first note that thanks to the binding properties of the commitment, the view $\mathbf{w}_1$ contained in $\mathbf{z}_1$ and the one contained in $\mathbf{z}_3$ are identical, and the same holds for the other views $\mathbf{w}_2, \mathbf{w}_3$ and random tapes $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3$. Then, we can traverse the decomposition of ϕ backwards from the output to the input shares: since the three conversations are accepting, we have that $\mathsf{Rec}(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3) = \mathbf{y}$, that $\mathbf{y}_i = \mathsf{Output}_i(\mathbf{w}_i) \forall i$, and finally that every entry in all of $\mathbf{w}_i$ was computed correctly. Therefore, since the Share function is surjective, we can compute $\mathbf{x}' = \mathsf{Share}^{-1}(\mathbf{w}_1[0], \mathbf{w}_2[0], \mathbf{w}_3[0])$. Thanks to the prefect correctness of the composition we thus have that $\phi(\mathbf{x}') = \mathbf{y}$, which is what we wanted to prove. Note that the protocol does not satisfy 2-special soundness, even if two accepting conversation actually contain all three views: in this case, since one of the branches of the computation has not been checked, $\exists i s.t.$ $\mathbf{w}_i$ might not be equal to $\phi_i^{(j)}(\mathbf{w}_i, \mathbf{w}_{i+1}, \mathbf{k}_i, \mathbf{k}_{i+1})$.

To prove the special honest-verifier ZK property, we consider the simulator S defined by the following steps. The input are $\mathbf{y} \in L_\phi$ and $e \in [3]$: run the 2-privacy simulator (which is guaranteed to exist thanks to the 2-privacy property of the decomposition $\mathcal{D}$ as in Definition 3), which returns $(\{\mathbf{k}_i, \mathbf{w}_i\}_{i \in \{e, e+1\}}, \mathbf{y}_{e+2})$, sets $\mathbf{w}_{e+2} = 0^{|\mathbf{w}|}, \mathbf{k}_{e+2} = 0^{|\mathbf{k}|}$ and then constructs $\mathbf{a}$ by committing to the three views and tapes.

ZKBoo Protocol

The verifier and the prover have input $\mathbf{y} \in L_\phi$. The prover knows $\mathbf{x}$ such that $\mathbf{y} = \phi(\mathbf{x})$. A (2,3)-decomposition of ϕ is given. Let Π^*_ϕ be the protocol related to this decomposition.

Commit: The prover does the following:
1. Sample random tapes $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3$;
2. Run $\Pi^*_\phi(\mathbf{x})$ and obtain the views $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$ and the output shares $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3$;
3. Commit to $\mathbf{c}_i = \mathsf{Com}(\mathbf{k}_i, \mathbf{w}_i)$ for all $i \in [3]$;
4. Send $\mathbf{a} = (\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3)$.

Prove: The verifier choose an index $\mathbf{e} \in [3]$ and sends it to the prover. The prover answers to the verifier's challenge sending opening $\mathbf{c}_e, \mathbf{c}_{e+1}$ thus revealing $\mathbf{z} = (\mathbf{k}_e, \mathbf{w}_e, \mathbf{k}_{e+1}, \mathbf{w}_{e+1})$.

Verify: The verifier runs the following checks:
1. If $\mathsf{Rec}(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3) \neq \mathbf{y}$, output reject;
2. If $\exists\, i \in \{e, e+1\}$ s.t. $\mathbf{y}_i \neq \mathsf{Output}_i(\mathbf{w}_i)$, output reject;
3. If $\exists j$ such that
$$\mathbf{w}_e[j] \neq \phi_e^{(j)}\big(\mathbf{w}_e, \mathbf{w}_{e+1}, \mathbf{k}_e, \mathbf{k}_{e+1}\big)$$
output reject;
4. Output accept;

Fig. 6. ZKBoo protocol for the language L_ϕ in the commitment-hybrid model.

Efficiency. Given a function $\phi : \mathbb{Z}^k \to \mathbb{Z}^\ell$ that can be expressed by a circuit over the finite ring $\mathbb{Z}$ with N multiplication gates, if we repeat repeat $\sigma(\log_2 3 - 1)^{-1}$ copies of the above protocol instantiated with the linear decomposition described before, and where we generate the random tapes pseudo-randomly with security parameter κ, we get a Σ-protocol with soundness $2^{-\sigma}$ and size

$$2 \cdot \sigma \cdot (\log_2(|\mathbb{Z}|)(k + N + \ell) + \kappa)$$

5 Zero-Knowledge for SHA-1/SHA-256

In this section we describe our case study, in which we implemented the protocol described in Section 4 for proving knowledge of preimages of SHA-1 and SHA-256. We start describing the choices we made in our implementation, describe the result of our empirical validation and finally compare with state-of-the-art protocols for the same task. Our implementation is available at `https://github.com/Sobuno/ZKBoo`.

5.1 Circuits For SHA-1/SHA-256

The linear-decomposition protocol described in Section 4 can be used with arithmetic circuits over arbitrary rings. Our first choice is picking a ring in which to express the computation of SHA-1/SHA-256. The two functions are quite similar, and they both use vectors of 32 bits for internal representation of values. Three kind of operations are performed over these bit-vectors: bitwise XORs, bitwise ANDs, and additions modulo 2^{32}. Implementing the two algorithms (after some simple optimization to reduce the number of bitwise ANDs) requires the following number of operations:

	AND	XOR	ADD
SHA-1	40	372	325
SHA-256	192	704	600

Hence, the two natural choices for the ring are $\mathbb{Z}_2$ (where XOR gates are for free but AND/ADD require a 32 multiplication gates) and $\mathbb{Z}_{2^{32}}$ (where ADD is free but bitwise operations require a linear number of multiplication gates). Since the number of XORs dominates in both algorithms, we opted for an implementation over the ring $\mathbb{Z}_2$.

5.2 Implementation of Building Blocks

We wrote our software in C, using the OpenSSL[9] library. We instantiated the building blocks in our protocol in the following way:

RNG: We generate the random tapes pseudorandomly using AES in counter mode, where the keys are generated via the OpenSSL secure random number generator. In the linear decomposition of multiplication gates, we use a random function $R : [N] \to \mathbb{Z}_2$. We implement this function by picking a bit from the stream generated using AES. In particular, we compute

$$R(i) = \mathrm{AES}(K, \lfloor i/128 \rfloor)[i \bmod 128]$$

which means that 3 calls to AES are sufficient to evaluate 128 individual AND gates. Note that since N (the number of AND gates) is known in advance, we can precompute all calls to AES at the beginning of the protocol. These two optimizations, together with the native support for AES in modern processors, proved very effective towards decreasing running times.

Commitments: In the first step of the protocol the prover commits to the three views $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$. Those commitments have been implemented using SHA-256 as the commitment function i.e., $\mathsf{Com}(x, r) = \text{SHA-256}(x, r)$. Under the (mild) assumptions that SHA-256 is collision resistant and that SHA-256$(\cdot, r)$ is a PRF (with key r) the commitments are binding and hiding.

The Fiat-Shamir Oracle. To make the proofs non-interactive, we need a random oracle $H : \{0,1\}^* \to \{1,2,3\}^r$ where r is the number of repetitions of our basic protocol. We instantiate this using SHA-256 as a random oracle and by performing rejection sampling. In particular, we compute the first output coordinate of $H(x)$ by looking at the first two output bits of SHA-256$(0, x)$ and mapping $(a, b) \to 2a + b + 1$. In case that $(a, b) = (1, 1)$ we look at the third-fourth bit instead and repeat. If there are no more bits left in the output of the hash function, we evaluate SHA-256$(1, x)$ and so on. In our experiments the maximum number of repetition is $r \in \{69, 137\}$, thus we call the hash function once or twice (on expectation).

5.3 Experimental Setup

We report on the results of the implementation of SHA-1 and SHA-256 for 69 and 137 repetitions each. Those correspond to soundness errors 2^{-40} and 2^{-80}. While the security level 2^{-40} is not sufficient for the case of non-interactive zero-knowledge, it offers reasonable security guarantees in the interactive case – note however that in this case our timings are only indicative of the local computation as they do not account for the necessary network communication.

Our experiments were run on a machine with an AMD FX-8350 CPU, running 8 cores at 4.00 GHz. The programs were run under Windows 10 Pro version 1511 (OS Build 10586.14) on a Seagate Barracuda 7200 RPM SATA 3.0 Gb/s hard drive with 16MB cache. Note that computing and verifying our proofs is an embarrassingly parallel task, thus it was possible to effortlessly take advantage of our multi-core architecture using OpenMP[10], an API useful for making a C program multi-threaded. We note that we have only done this for the main loop of the program, which iterates over the individual repetitions of the proofs (which are clearly independent from each other), thus it is likely that there is room for further parallelization. Timings were done using C native clock() function and are measured in milliseconds.

[9] https://www.openssl.org

[10] http://openmp.org

5.4 Experimental Results

Breakdown. In Table 1 we report on the timings we obtained for both SHA-1 and SHA-256, with 69 and 137 rounds, both enabling and disabling parallelization. In this table we also present a breakdown of the running time. In particular we measure the following phases for the prover:

- *Commit*: This is the time to run the **Commit** procedure (Figure 6) to produce $\mathbf{a}$. It is further divided into the following sub-timings: (*Rand. gen.*) Generation of all needed randomness using OpenSSL RNG as well as preprocessing of the PRF; (*Algorithm exec.*) Time taken to run the algorithm Π_ϕ^*. This is the total time for all 69/137 rounds; (*Commitment*) Generating commitments of the views;
- *Gen. challenge*: Using the random oracle to generate the challenge vector as $\mathbf{e} = H(\mathbf{y}, \mathbf{a})$;
- *Prove*: Building the vector $\mathbf{z}$;
- *Output to disk*: Writing $(\mathbf{a}, \mathbf{e}, \mathbf{z})$ to disk;[11]

For the verifier:

- *Input from disk*: Reading the proof from file;
- *Gen. challenge*: Regenerate the challenge vector using the random oracle;
- *Verify*: The time to run all the rounds of the **Verify** procedure;

Finally, with *proof size* we indicate the size of the string $\pi = (\mathbf{y}, \mathbf{a}, \mathbf{z})$ on disk in KB.

	SHA-1				SHA-256			
	69 rounds		137 rounds		69 rounds		137 rounds	
	Serial	Paral.	Serial	Paral.	Serial	Paral.	Serial	Paral.
Prover (ms)	18.98	8.12	31.73	12.73	30.81	12.45	54.63	15.95
Commit	13.45	3.68	26.73	6.59	24.47	5.86	48.25	10.07
- Rand. gen.	1.35	0.60	2.47	0.88	2.28	0.80	4.46	1.13
- Algorithm exc.	10.41	2.69	21.55	5.06	19.60	4.44	38.68	7.87
- Commitment	1.37	0.39	2.71	0.64	2.56	0.62	5.09	1.07
Gen. challenge	0.06	0.05	0.10	0.13	0.05	0.06	0.09	0.09
Prove	0.12	0.18	0.28	0.32	0.32	0.39	0.07	0.53
Output to disk	5.35	4.21	4.62	5.70	5.08	5.39	4.76	4.43
Verifier (ms)	11.68	2.35	22.85	4.39	34.16	6.77	67.74	13.20
Input from disk	0.09	0.11	0.13	0.16	0.15	0.16	0.29	0.25
Gen. challenge	0.06	0.05	0.10	0.10	0.06	0.05	0.10	0.11
Verify	11.53	2.20	22.63	4.12	33.95	6.56	67.35	12.85
Proof size (KB)	223.71		444.18		421.01		835.91	

Table 1. Breakdown of times and proof size for 69/137 rounds of SHA-1/SHA-256, average of 1000 runs

Parallelization. Figure 7 and 8 show how the running time of the prover (resp. verifier) changes when we change the number of rounds (from 1 to 137) and the number of threads (from 1 to 8). We include the graphs for SHA-256 only. It is easy to see that the running time increases linearly with the number of rounds, and that the improvement due to multithreading is significant. The graph indicates that there is some fluctuation in the algorithm's run time for all number of threads when using up to about 85 rounds. While we have no definitive answer as to why this happens, we think it might be related to the memory management of the program and operating system. We note that the runtime of the verifier benefits more from parallelisation. This is consistent with Amdahl's law since, as shown in Table 1, the prover spends signficantly more time performing tasks which do not benefit from parallelization (e.g., writing to disk).

[11] We observed that the timings of writing to disk are very noisy, and not always monotone in the size of the written file.

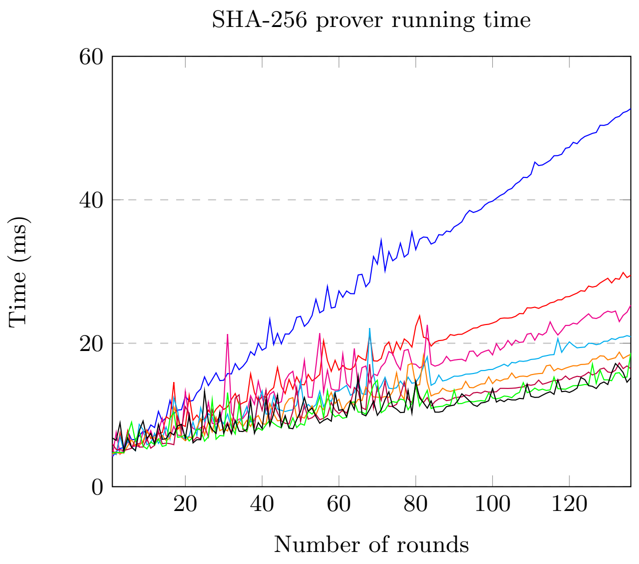

Fig. 7. Relation between running time and number of rounds for the SHA-256 prover, average over 100 runs.

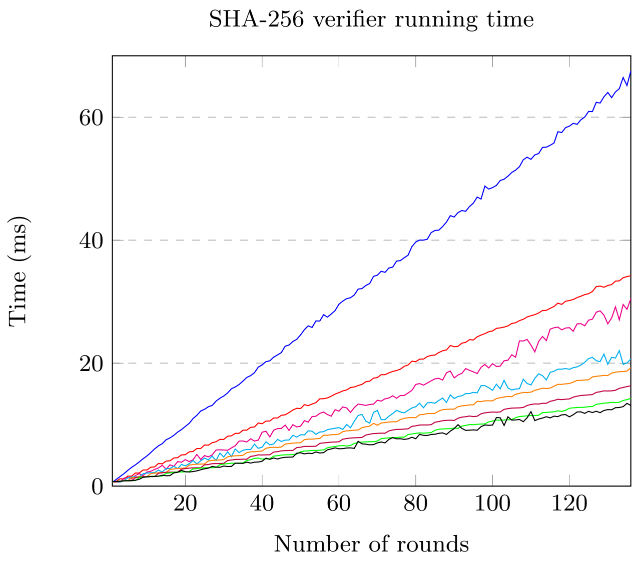

Fig. 8. Relation between running time and number of rounds for the SHA-256 verifier, average over 100 runs.

	Preproc. (ms)	Prover (ms)	Verifier (ms)	Proof size (B)
ZKBoo	0	13	5	454840
ZKGC (Estimates)	0	$>$ 19 (OT only)	$>$ 25 (OT only)	186880
Pinocchio	9754	12059	8	288

Table 2. Comparison of approaches for SHA-1

5.5 Comparison

Here we compare the performances of ZKBoo with some of the state-of-the-art protocol for the same task. In particular, we compare the performances of proving/verifying knowledge of SHA-1 preimages across ZKBoo, Pinocchio [PHGR13] and ZKGC [JKO13].

Pinocchio [PHGR13] is an implementation of SNARKs for verifiable outsourcing of computation. While not its main purpose, it can generate zero-knowledge proofs at a negligible extra cost over sound-only proofs. The choice of benchmarking SHA-1 only (and not SHA-256) is due to the fact that the Pinocchio library only contains SHA-1. The runtime reported for Pinocchio are obtained on the same machine as our implementation. The results shows that ZKBoo is faster at both proving and verifying, with an incredible 10^3 factor for the prover. Note here that if the underlying circuit had been larger, the proof size and the verification time of Pinocchio would not change, while its preprocessing and proving time would grow accordingly. We note also that Pinocchio has a large preprocessing time where some prover/verifier key are generated. Those keys are circuit dependent, and for SHA-1 the prover key is 6.5 MB and the verifier key is 1.1 MB. To Pinocchio's defence, it must be noted that Pinocchio is a general purpose system that can generate proofs for any circuit (provided as an input file) while our implementation contains the SHA circuit hard-coded. Moreover according to [PHGR13], Pinocchio has not been parallelised. While it is conceivable that Pinocchio could be made faster in a similar way, we do not believe that Pinocchio could ever reach proving times similar to ZKBoo, due to the use of heavy public-key technology (exponentiations in a pairing-friendly group) for each gate in the circuit.

ZKGC [JKO13]. For the case of ZKGC, we could not directly compare implementations, since the source code for [JKO13] is not publicly available. In addition, since the publication of [JKO13], several significant improvements have been proposed but have not been implemented yet. Therefore, in Table 2, we give an accurate estimate of the size of the proofs generated using ZKGC but only a lower-bound for its runtime. The estimates are computed using the following tools: *(GC)* we estimate the proof size using the communication complexity of the most efficient (in terms of communication complexity) garbled circuits, namely *privacy-free garbled circuits* [FNO15,ZRE15] that can be instantiated with as little as one ciphertext (128 bits using AES) per AND gate in the circuit; (OT) we plug the size and runtime given by the most efficient OT available [CO15]. Since the input size of SHA-1 is quite large (512 bits), it might be that using OT extension would prove useful. Therefore, to make the comparison even more favourable towards ZKGC, we only count the runtime of 190 base OTs necessaries for active secure OT extensions [ALSZ15] and we do not account at all for the runtime of the OT extension protocol nor the generation/verification of the GC. The resulting estimates show that even when *counting the base OTs alone*, the runtime of ZKGC is already larger than the runtime of ZKBoo for the SHA-1 circuit. As for proof size, we note that ZKGC produces shorter proofs. However, the approach of ZKGC cannot be made non-interactive which is a qualitative drawback and it is likely to introduce significant slow-downs due to network latency.

6 Conclusions

In this paper we described ZKBoo, the first attempt to make general purpose zero-knowledge practical using the "MPC-in-the-head" approach of Ishai et al. [IKOS07]. We discussed how to generalize their protocol using the idea of $(2, 3)$-function decompositions, we showed simple linear decompositions for arithmetic circuits over any ring and we leave it as a future work to find compact decompositions for other interesting functions.

Our experimental results show that for practically relevant circuits (such as SHA-1), our protocol is the fastest in terms of proving time, and where the verification time is comparable even with SNARKs technology.

References

[ALSZ15] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 673–701, 2015.

[BCG$^+$14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474, 2014.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 781–796, 2014.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188. Springer, 2011.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[CO15] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, pages 40–58, 2015.

[Dam] Ivan Damgaard. On σ-protocols (2010). *Lecture on Cryptologic Protocol Theory (Aaurhus Unvivesrity, course notes).*

[DI06] Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In *Advances in Cryptology-CRYPTO 2006*, pages 501–520. Springer, 2006.

[DIK10] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *Proceedings of EuroCrypt*, pages 445–465, Springer Verlag 2010.

[DZ13] Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *TCC*, pages 621–641, 2013.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 191–219, 2015.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO'86*, pages 186–194. Springer, 1986.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304, 1985.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[HL10] Carmit Hazay and Yehuda Lindell. *Efficient secure two-party protocols: Techniques and constructions.* Springer Science & Business Media, 2010.

[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*, STOC '07, pages 21–30. ACM, 2007.

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM Journal on Computing*, 39(3):1121–1152, 2009.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 955–966, 2013.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 681–700, 2012.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 238–252, 2013.

[Rab05] Michael O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, pages 239–252, 1989.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 220–250, 2015.

A Appendix

Proof of Proposition 1

Proof. In order to prove that the decomposition $\mathcal{D}^{\mathbb{Z}}$ is correct is enough to prove that for any $c \in [N]$ the following holds.

(1) if the c-th gate is an "add α" gate, then:

$$\sum_{i=1}^{3} \mathbf{w}_i[c] = \left(\sum_{i=1}^{3} \mathbf{w}_i[a]\right) + \alpha$$

(2) if the c-th gate is an "mult. α" gate, then:

$$\sum_{i=1}^{3} \mathbf{w}_i[c] = \left(\sum_{i=1}^{3} \mathbf{w}_i[a]\right) \cdot \alpha$$

(3) if the c-th gate is an addition gate, then:

$$\sum_{i=1}^{3} \mathbf{w}_i[c] = \left(\sum_{i=1}^{3} \mathbf{w}_i[a]\right) + \left(\sum_{i=1}^{3} \mathbf{w}_i[b]\right)$$

(4) if the c-th gate is a multiplication gate, then:

$$\sum_{i=1}^{3} \mathbf{w}_i[c] = \left(\sum_{i=1}^{3} \mathbf{w}_i[a]\right) \cdot \left(\sum_{i=1}^{3} \mathbf{w}_i[b]\right)$$

Indeed, using (1), (2), (3) and (4) iteratively for all the gates in the circuit we can prove that $\sum_{i=1}^{3} \mathbf{w}_i[N] = \phi(\sum_{i=1}^{3} \mathbf{x}_i)$ and from this it follows that

$$\begin{aligned} \Pi^*_\phi(\mathbf{x}) &= \mathsf{Rec}^{\mathbb{Z}}(\mathbf{y}_1, \ldots, \mathbf{y}_n) = \sum_{i=1}^{3} \mathbf{y}_i \\ &= \sum_{i=1}^{3} \mathbf{w}_i[N] = \phi\left(\sum_{i=1}^{3} \mathbf{x}_i\right) = \phi(\mathbf{x}) \end{aligned}$$

The first three follow trivially by the definition of the function $\phi^{(c)}_{A_i}$ when the c-th gate is a an "add α", "mult. α" and addition gate, respectively. Now assume that the c-th gate is a multiplication gate. Then, using the definition for the function $\phi^{(c)}_{A_i}$ for this case and recalling that the index values are computed modulo 3, we have that

$$\begin{aligned} \sum_{i=1}^{3} \mathbf{w}_i[c] &= \sum_{i=1}^{3} \left(\mathbf{w}_i[a] \cdot \mathbf{w}_i[b] + \mathbf{w}_{i+1}[a] \cdot \mathbf{w}_i[b] + \mathbf{w}_i[a] \cdot \mathbf{w}_{i+1}[b] + R_i(c) - R_{i+1}(c)\right) \\ &= \sum_{i=1}^{3} \mathbf{w}_i[a] \cdot (\mathbf{w}_i[b] + \mathbf{w}_{i+1}[b]) + \sum_{i=1}^{3} \mathbf{w}_i[a] \cdot \mathbf{w}_{i+2}[b] + \sum_{i=1}^{3} R_i(c) - \sum_{i=1}^{3} R_i(c) \\ &= \left(\sum_{i=1}^{3} \mathbf{w}_i[a]\right) \cdot \left(\sum_{i=1}^{3} \mathbf{w}_i[b]\right) \end{aligned}$$

We now pass to prove the 2-privacy property. Given $e \in [3]$, we define the simulator S_e on input $\mathbf{y}$ with the following instructions:

1. Sample random tapes $\mathbf{k}'_e, \mathbf{k}'_{e+1}$;
2. Sample uniformly at random the values $\mathbf{w}'_e[0]$ and $\mathbf{w}'_{e+1}[0]$. Then, for all $c \in [N]$: If the c-th gate is an "add α", "mult. α" or addition gate then define $\mathbf{w}'_e[c]$ and $\mathbf{w}'_{e+1}[c]$ using the functions $\phi_e^{(c)}$ and $\phi_{e+1}^{(c)}$, respectively. If the c-th gate is a multiplication gate then sample uniformly at random the value $\mathbf{w}'_{e+1}[c]$ and compute the value $\mathbf{w}'_e[c]$ using $\phi_e^{(c)}$; In this way define the entire views $\mathbf{w}'_e$ and $\mathbf{w}'_{e+1}$;
3. Compute $\mathbf{y}'_e = \mathsf{Output}_e(\mathbf{w}'_e)$ and $\mathbf{y}'_{e+1} = \mathsf{Output}_{e+1}(\mathbf{w}'_{e+1})$;
4. Compute $\mathbf{y}'_{e+2} = \mathbf{y} - (\mathbf{y}'_e + \mathbf{y}'_{e+1})$;
5. Output $(\{\mathbf{k}'_i, \mathbf{w}'_i\}_{i\in\{e,e+1\}}, \mathbf{y}'_{e+2})$

It is easy to verify that the output of the simulator S_e has the same distribution of the string $(\{\mathbf{k}_i, \mathbf{w}_i\}_{i\in\{e,e+1\}}, \mathbf{y}_{e+2})$ produced by the protocol Π^*_ϕ. Indeed, all the elements in the output of S_e are computed using the same commands used in Π^*_ϕ, except for the element $\mathbf{w}'_{e+1}[c]$ when the c-th gate is a multiplication gate. In this case $\mathbf{w}'_{e+1}[c]$ is sample uniformly at random, while $\mathbf{w}_{e+1}[c]$ in the protocol is computed using the function $\phi_{e+1}^{(c)}$. In particular, $\mathbf{w}_{e+1}[c]$ is computed by subtracting to determined value the element $R_{i+2}(c)$. Since R_{i+2} is an uniformly random function sampled using an independent tape $\mathbf{k}_{e+2}$, the distribution of $\mathbf{w}_{e+1}[c]$ in the protocol is the uniform one, that is it has the same distribution of $\mathbf{w}'{}_{e+1}[c]$ in the output of S_e. Therefore, we can conclude that S_e is a correct simulator for the decomposition $\mathcal{D}^{\mathbb{Z}}$.

Finally, by inspection we have that $|\mathbf{w}_i| = (k + N + \ell)\log|\mathbb{Z}| + \kappa$ for all $i \in [3]$.