SoK: Formal security analysis of MPC-in-the-head zero-knowledge protocols

Nikolaj Sidorenco[1], Sabine Oechsner[1], and Bas Spitters[2]

[1] Aarhus University, Aarhus, Denmark
[2] Concordium Blockchain Research Center, Aarhus, Denmark
{sidorenco, oechsner, spitters}@cs.au.dk

Abstract

Zero-knowledge proofs allow a *prover* to convince a *verifier* of the veracity of a statement without revealing any other information. An interesting class of zero-knowledge protocols are those following the MPC-in-the-head paradigm (Ishai et al., *STOC '07*) which use secure multiparty computation (MPC) protocols as basis. Efficient instances of this paradigm has emerged as an active research topic in the last years, starting with ZKBoo (Giacomelli et al., *USENIX '16*). Zero-knowledge protocols are a vital building block in the design of privacy-preserving technologies as well as cryptographic primitives like digital signature schemes that provide post-quantum security.

This work investigates the security of zero-knowledge protocols following the MPC-in-the-head paradigm. We provide the first machine-checked security proof of such a protocol on the example of ZKBoo. Our proofs are checked in the EasyCrypt proof assistant. To enable a modular security proof, we develop a new security notion for the MPC protocols used in MPC-in-the-head zero-knowledge protocols. This allows us to recast existing security proofs in a black-box fashion which we believe to be of independent interest.

1 Introduction

Zero-knowledge proofs [1] allow a party, the prover, to convince another party, acting as the verifier, of the veracity of some statement without revealing anything else. This seemingly paradoxical primitive lies at the heart of many modern privacy-preserving technologies, and more generally is a crucial cryptographic building block for applications like digital signature schemes.

One approach to constructing zero-knowledge proofs has gained particular attention over the last years: the MPC-in-the-head paradigm of Ishai et al. [2] which uses secure multiparty computation (MPC) protocols in a surprising way as building block. Consider the setting where a prover holds the pre-image x of a public one-way function f and has published $y = f(x)$. To convince the verifier that they indeed know x corresponding to y,

the prover will first split the secret x into random shares $x_1, \ldots, x_n$ such that $\sum_i x_i = x$. The prover then emulates an MPC protocol "in their head", with the catch that the protocol performs a distributed computation of $f(x)$ with shares $x_1, \ldots x_n$ as inputs. This emulation yields one transcripts of the protocol execution per party. Prover and verifier can then interact to reveal a subset of transcripts, which the prover can check for consistency. If the consistency check succeeds, then the verifier will be convinced that the prover knows x. Intuitively, this does not leak any information about x if the MPC protocol is secure against insider corruption of some parties and not too many transcripts are revealed.

While at first believed to be of purely theoretical interest, the MPC-in-the-head paradigm was subsequently shown to be of practical relevance [3]. Combined with the Fiat-Shamir heuristic [4], one can moreover obtain efficient digital signature schemes from such zero-knowledge proofs. In fact, Picnic, a successful contender for the NIST post-quantum cryptography standardization competition [5], follows this design pattern. Moreover, multiple efficiency improvements have been proposed recently [6, 7]. Given the standardization potential of this approach, it is natural to ask to formally verify such constructions.

1.1 Our Contributions

In this work, we investigate the security of MPC-in-the-head type zero-knowledge proofs like ZKBoo [3], Picnic [5,8], KKW [9], and Banquet [7].

- We provide the first machine-checked security proof of a zero-knowledge protocol following the MPC-in-the-head paradigm. Our mechanization studies the ZKBoo protocol [3] and is done in the EasyCrypt proof assistant [10]. Interestingly, protocols following the MPC-in-the-head paradigm use MPC protocols as building block in a bigger construction rather than as goal, and we are not aware of any other machine-checked proof with this property.

- To enable a modular security proof, we develop a new security notion for the MPC protocols in question which is of independent interest. The new notion enables us to give black-box security proofs of MPC-in-the-head zero-knowledge protocols.

Our starting point is the ZKBoo protocol by Giacomelli et al. [3] as a representative of this protocol class. From a technical perspective, this class of protocols is an interesting challenge due to the unconventional combination of complex primitives like MPC and zero-knowledge proofs. Based on the observation that modularity of existing constructions currently does not carry over to modularity of proofs, we propose to use a refined notion of the MPC protocol (called *decomposition* protocol, to keep with the ZKBoo terminology). This new decomposition notion then allows us to define black-box transformations from decomposition to Σ-protocols, a special class of zero-knowledge protocol. To demonstrate the generality of this approach, we recast existing protocols in this style. On a conceptual level, this clear separation between decomposition and

transformation to Σ-protocol improves the understanding of the different optimization strategies, and can hopefully help find new ones. With a clear proof strategy set up, we then proceed to mechanize the security proof in EasyCrypt. The EasyCrypt code is in the attached zip-file.

1.2 Outline

Section 2 presents the necessary background for the rest of this work. The MPC-in-the-head paradigm is presented in Section 3, and we discuss moreover the ZKBoo protocol and its existing security proof as an example. In Sections 4 and 5 we present our new decomposition notion and demonstrate the black-box construction of a Σ-protocol from it. Further protocols and how they fit into our formalization are discussed in Section 6. Section 7 presents our EasyCrypt formalization of the ZKBoo protocol. Related work is discussed in Section 8 before we discuss future work and conclude in Section 9 and 10.

2 Preliminaries

This section presents some cryptographic concepts that are necessary to understand the rest of this work.

2.1 Commitments

A *commitment scheme* is a cryptographic primitive that allows a committer holding message m to convince a verifier of the following. Firstly, that some m was fixed at some point in time without revealing the value of m. This is done by sending a *commitment*, i.e. some token derived from m, to the verifier. Second, the committer can later open the commitment to reveal m and convince the verifier that the message was not modified in the meantime.

Definition 2.1 (Commitment scheme)**.** *A* commitment scheme *consists of a tuple* $(\texttt{setup}, \texttt{com}, \texttt{cverify})$ *of probabilistic algorithms with the following properties:*

- *Correctness: Let* $ck \leftarrow \texttt{setup}(1^\kappa)$*. For all* m *and* $(c, r) \leftarrow \texttt{com}(ck, m)$*,* $\texttt{cverify}(m, c, r) = \top$*.*
- *Perfect hiding: Let* $ck \leftarrow \texttt{setup}(1^\kappa)$*. For all* m, m'*, with* $m \neq m'$*, the distributions* $\texttt{com}(ck, m)$ *and* $\texttt{com}(ck, m')$ *are identical.*
- *Computational binding: Let* $ck \leftarrow \texttt{setup}(1^\kappa)$*, and* c *a commitment. Then for any adversary and message* m*, the probability of finding* r, r' *such that* $\texttt{cverify}(m, c, r) = \texttt{cverify}(m, c, r') = \top$ *is negligible.*

Note that we limit ourselves to the above definition of perfectly hiding and computationally binding commitments. There are other notions that will not be discussed here.

2.2 MPC

A secure multiparty computation (MPC) protocol allows a set of n mutually distrusting parties $P_1, \ldots, P_n$ to compute a public function f of their private inputs $x_1, \ldots, x_n$. The function f is typically assumed to be represented as an arithmetic circuit for the sake of the protocol. Security can be studied with respect to different corruption models. In this work, we focus on passive security (also called honest-but-curious) where all protocol participants are assumed to follow the protocol specification, but might try to derive additional information from the messages they receive. An MPC protocol is deemed passively secure if it provides

- Correctness: Parties learn the correct output $f(x_1, \ldots, x_n)$, and
- Privacy: Parties do not learn anything about the inputs of honest parties beyond what $f(x_1, \ldots, x_n)$ reveals.

We will denote by *view* the transcript of a protocol execution from the point of view of a party P_i, consisting of the input x_i, all messages P_i receives, as well as its random choices.

2.3 Zero-knowledge protocols

Zero-knowledge protocols [1] are a cryptographic primitive that allows a prover P to convince a verifier V of the veracity of a public statement, without revealing anything beyond that fact.

2.3.1 Σ-protocols

An important subclass of zero-knowledge protocols are Σ-protocols [11]. A Σ-protocol is a zero-knowledge proof of knowledge for a relation R, i.e. it allows a prover to prove knowledge of a witness x for a public statement h in relation R.

Definition 2.2 (Σ-protocol)**.** *Let R be a relation. A* Σ*-protocol for R is an interactive protocol between a prover P and a verifier V, where P and V hold a common input h and P has additional secret input x with $R(h, x)$, with the following properties:*

- *The protocol has a special 3-move form (a, e, z) as shown in Fig 1.*
- *Completeness: If prover P is honest, i.e. $R(h, x)$ and P follows the protocol, then an honest verifier V will always accept.*
- *s-special soundness: Given s transcripts $(a, e_1, z_1), \ldots, (a, e_s, z_s)$, an x' with $R(h, x')$ can be extracted from the transcripts.*
- *Special honest-verifier zero-knowledge: Assuming that the verifier is honest, there exists a simulator S that simulates transcripts such that real and simulated transcripts are statistically indistinguishable.*

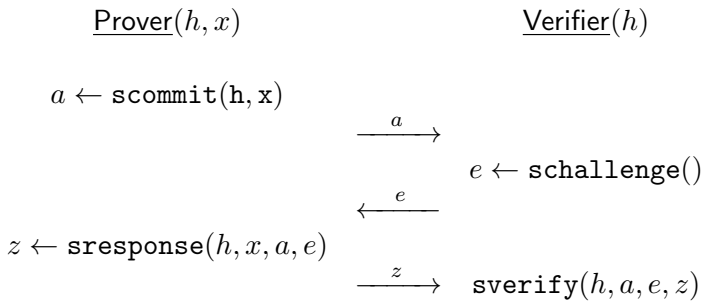

Figure 1: Σ-Protocol overview

Interactive Σ-protocols can be made non-interactive and turned into digital signature schemes via the Fiat-Shamir transform [4]. The idea is to replace the random challenge by the verifier by the output of a hash function on input the statement to be proved as well as the first protocol message, which ensures that the prover chooses the first message before seeing the challenge. This transformation from proof of knowledge to signature was proven secure in the random oracle model by Pointcheval and Stern [12].

3 The MPC-in-the-head paradigm

Since the invention of the zero-knowledge concept, many approaches to constructing protocols were proposed. In recent years, the *MPC-in-the-head paradigm* (Ishai et al. [2]) has gained popularity. In this section, we briefly revisit the MPC-in-the-head paradigm as well as the ZKBoo protocol.

3.1 MPC-in-the-head-based zero-knowledge

To obtain a zero-knowledge protocol from an MPC protocol, the MPC-in-the-head paradigm proposes the following idea. Assume there is a public function ϕ and value y, and we want to prove knowledge of a witness x such that $\phi(x) = y$ in zero knowledge. The value y could for example be the output of the SHA-256 hash function ϕ. As is standard in the MPC literature, we assume that ϕ is given in the form of a circuit.

- The prover P starts by secret sharing the private input x into inputs $x_1, \ldots, x_n$ to virtual parties $P_1, \ldots, P_n$. Assume that the circuit representation of ϕ is chosen such that it evaluates the function on such a shared input. The prover then runs an MPC protocol for evaluating ϕ on those shares "in their head". As a result, P obtains one protocol transcript for each party, also referred to as *views*. The prover then commits to all views and sends the commitments to the verifier V.
- The prover and verifier engage in an interactive protocol to select and open a random subset of committed views.
- The prover opens those commitments to reveal the requested views.

- The verifier checks consistency of the opened views and accepts if they are consistent as well as valid openings of the commitments, and otherwise rejects.

The crucial observation is that if the MPC protocol allows for local verifiability of views, then the above idea yields zero-knowledge protocols. While the MPC-in-the-head paradigm was initially believed to be of mostly theoretical interest, a series of recent works, starting with ZKBoo [3], showed it to be of practical relevance.

3.2 ZKBoo

We will now study the ZKBoo protocol as a concrete instance of the MPC-in-the-head paradigm. The ZKBoo protocol [3] was the first construction to show that the MPC-in-the-head paradigm [2] could actually be instantiated to yield a practically efficient protocol. The idea is to use a secret-sharing-based MPC protocol with three parties and a particular communication pattern as basis: Each party P_i only sends messages to one of the other parties, namely their neighbor P_{i-1}. This pattern ensures that meaningful consistency checks can be performed given a pair of views of a protocol execution. The protocol operates on arithmetic circuits over a finite field $\mathbb{Z}_p$.

3.2.1 The Construction

For convenience, and to separate the MPC protocol from the Σ-protocol construction, the authors define *(2,3)-decomposition*. This is the view generation for an MPC protocol with three parties and privacy against passive corruption of two parties. This decomposition can then be combined with any commitment scheme to obtain a Σ-protocol for proving knowledge of a pre-image of a value y under a function ϕ.

(2,3)-Decomposition Let ϕ be a function which is represented as circuit with N gates. A (2,3)-decomposition for ϕ is defined as follows:

Definition 3.1 ([3])**.** *A* (2,3)-decomposition *for a function ϕ is the set of functions $\mathcal{D} = \{\mathsf{Share}, \mathsf{Rec}, \phi_1^{(1)}, \ldots, \phi_1^{(N+1)}, \ldots, \phi_3^{(1)}, \ldots, \phi_3^{(N+1)}, \mathsf{Output}_1, \mathsf{Output}_2, \mathsf{Output}_3\}$ such that* Share *is a surjective function and $\phi_m^{(i)}$,* Output_i *and* Rec *are functions as described before. Let Π_ϕ^* be the algorithm in Fig. 2, then we say that $\mathcal{D}$*

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

The decomposition functions are implemented by ZKBoo as:

- $\mathsf{Share}(\mathbf{x}; \mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3)$ performs an additive secret sharing of $\mathbf{x}$ into three random shares $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ such that $\mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3$.

Protocol Π^*_ϕ

Let $\phi\colon X \to Y$ be a function and $\mathcal{D}$ a related (2,3)-decomposition as defined in Def. 3.1.
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 $i \in \{1,2,3\}$ compute:
     $\mathbf{w}_i[j] = \phi_i^{(j)}((\mathbf{w}_m[0..j-1], \mathbf{k}_m)_{m \in \{i,i+1\}})$.
4. Compute $\mathbf{y}_i = \mathsf{Output}(\mathbf{w}_i, \mathbf{k}_i)$ for $i \in \{1,2,3\}$.
5. Compute $\mathbf{y} = \mathsf{Rec}(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3)$.

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

Figure 2: Protocol Π^*_ϕ describing how to use decomposition, used in Def. 3.1. Reproduced from [3].

- $\mathsf{Rec}(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3)$ outputs $\mathbf{y} = \mathbf{y}_1 + \mathbf{y}_2 + \mathbf{y}_3$.
- The gate evaluation functions $\phi_i^{(j)}$ are defined in the following way. Consider the j-th gate, and let a and b be its left and right input gates, resp. Then for $i \in [3]$, $\phi_i^{(j)}$ is defined as:
  - unary addition of α:
    $$\mathbf{w}_i[j] = \phi_i^{(j)}(\mathbf{w}_i[a]) = \begin{cases} \mathbf{w}_i[a] + \alpha & \text{if } i = 1 \\ \mathbf{w}_i[a] & \text{otherwise} \end{cases}$$
  - unary multiplication by α:
    $\mathbf{w}_i[j] = \phi_i^{(j)}(\mathbf{w}_i[a]) = \alpha \cdot \mathbf{w}_i[a]$
  - binary addition:
    $\mathbf{w}_i[j] = \phi_i^{(j)}(\mathbf{w}_i[a], \mathbf{w}_i[b]) = (\mathbf{w}_i[a] + \mathbf{w}_i[b])$
  - binary multiplication:
    $$\begin{aligned} \mathbf{w}_i[j] =& \phi_i^{(j)}(\mathbf{w}_i[a,b], \mathbf{w}_{i+1}[a,b]) \\ =& \mathbf{w}_i[a] \cdot \mathbf{w}_i[b] + \mathbf{w}_{i+1}[a] \cdot \mathbf{w}_i[b] \\ & + \mathbf{w}_i[a] + \mathbf{w}_{i+1}[b] + R_i(j) - R_{i+1}(j) \end{aligned}$$
    where $R_i(j)$ is sampled uniformly at random using $\mathbf{k}_i$.
- $\mathsf{Output}_i(\mathbf{w}_i, \mathbf{k}_i)$ selects the shares of the output wires of the circuit.

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 Π^*_ϕ 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 $\mathbf{c}_i = \mathtt{com}(\mathbf{k}_i, \mathbf{w}_i)$ 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 chooses 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 $\mathbf{z} = (\mathbf{k}_e, \mathbf{w}_e, \mathbf{k}_{e+1}, \mathbf{k}_{e+1})$.

Verify: The verifier runs the following checks:

1. If the openings of commitments $\mathbf{c}_e, \mathbf{c}_{e+1}$ do not verify, output reject.
2. If $\mathsf{Rec}(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3) \neq \mathbf{y}$, output reject.
3. If $\exists i \in \{e, e+1\}$ such that $\mathbf{y}_i \neq \mathsf{Output}_i(\mathbf{w}_i)$, output reject.
4. If $\exists j$ such that $\mathbf{w}_e[j] \neq \phi_e^{(j)}(\mathbf{w}_e, \mathbf{w}_{e+1}, \mathbf{k}_e, \mathbf{w}_{e+1})$, output reject.
5. Otherwise output accept.

Figure 3: ZKBoo protocol, reproduced from [3].

ZKBoo protocol Given the (2,3)-decomposition described above and a commitment scheme, the ZKBoo protocol proceeds to construct a Σ-protocol as shown in Fig. 3, following the MPC-in-the-head paradigm. The protocol is shown to be a Σ-protocol assuming the security of the commitment scheme and the (2,3)-decomposition.

3.2.2 Black-Box Security

We will now revisit the security proof of the ZKBoo construction. The proof of [3, Prop. 4.2] is not black-box as it relies on implementation specifics rather than on the security guarantees given by the decomposition and the commitment scheme.

Revisiting the ZKBoo security proof To prove that the ZKBoo construction is a Σ-protocol, it is necessary to prove three properties: Completeness, 3-special soundness and special honest-verifier zero-knowledge.

The *completeness* property is derived from the correctness of the commitment scheme in combination with correctness of the decomposition. There is, however, a subtle issue that prevents this proof step from being fully black-box: Correctness of the decomposition itself does not guarantee anything about the verifier's ability to verify the opened views. More specifically, correctness is a property of the protocol Π^*_ϕ derived from a decomposition $\mathcal{D}$ that computes all three views, whereas the verifier can only recompute the view corresponding to challenge e. Hence the security proof needs to revisit the concrete implementation of verification (recomputing the views in this case) and conclude that verification is indeed possible. To be fully precise, one would also need to reprove that each computation step is performed correctly, as the standard correctness property in the MPC literature only guarantees correctness of the end result and not the intermediate computation steps[1].

The *3-special soundness* property is a modified special soundness property that proves witness extraction given 3 transcripts (instead of the usual 2). The proof relies on multiple assumptions: First, the binding property of the commitment scheme is used to argue that the opened views are identical in the overlapping indices except with negligible probability. The next step invokes the reconstruction property of the specific secret sharing scheme used by ZKBoo to extract a potential input. This non-black-box step is necessary due to the lack of an explicit extractability guarantee of the decomposition notion. Finally, correctness of the decomposition ensures that the extracted input is actually valid.

Finally, *special honest-verifier zero-knowledge* follows directly from 2-privacy of the decomposition and the hiding property of the commitment scheme, so this part is actually black-box.

Conclusion As explained above, the ZKBoo security proof is not black-box, which seems to stem from an incomplete formalization of the required properties of the under-

[1] Correctness of intermediate steps is of course shown during the proof, but this information is usually dropped in the final statement as it is not necessary for many applications.

lying MPC protocol. In the next sections we will make a black-box construction and proof. To do so we modify the notion of decomposition. This formalization is not limited to ZKBoo, but captures a range of other protocols, as we will discuss in Section 6

4 Decomposition protocols

Now that we understand why the decomposition notion of Giacomelli et al. [3] is not sufficient for a black-box security proof of the ZKBoo protocol, we will attempt to improve the situation. This section proposes a new decomposition notion and explains how the (2,3)-decomposition of Giacomelli et al. relates to it before we will show a black-box construction of the ZKBoo protocol from our decomposition notion in Section 5.

4.1 Syntax and Security

Let us first consider the syntax. First of all, we combine the Share and $\phi_i^{(j)}$ functions into one `decompose` algorithm since they are no longer used separately. Next, remember that the black-box proof issues we discussed relate to the extractability of a witness from views as well as a lack of understanding of verification of the views. To mitigate these issues, we add a new `verify` algorithm to the decomposition notion. Finally, we observe that optimizations of the ZKBoo protocol which we investigate in Section 6 improve in efficiency by not sending the full views in the last message of the Σ-protocol, but they perform a reversible compression step. For this reason, we add a `compress` algorithm to our formalization. So, the syntax of a decomposition looks as follows:

Definition 4.1 (Decomposition protocols)**.** *Let n denote the number of parties. A decomposition π is a collection of algorithms:* $(\texttt{decompose}, \texttt{compress}, \texttt{verify}, \texttt{out}, \texttt{rec})$ *and distributions $\mathcal{C}$ and $\mathcal{R}$. We let $\leftarrow_R \mathcal{R}$ denote uniformly random picking an element from the distribution. where,*

- $\texttt{decompose}(\phi, x, ks)$ *takes a circuit ϕ with input x and a collection of random values and returns n views. We fix the distribution $\mathcal{R}$ as the universe of all random value inputs accepted by* `decompose`*.*
- $\texttt{compress}(v)$ *is a compression function that transforms a view w into an alternative representation. For convenience, we define a compression function* $\texttt{compress}(ws, \mathcal{I}(e)) := (\texttt{compress}(ws[i]))_{i \in \mathcal{I}(e)}$ *for a full set of n views and a list of challenged views, produced by $\mathcal{I}$. We denote the universe of possible challenges $e \in \mathcal{C}$.*
- $\texttt{uncompress}(w)$ *is the inverse of* `compress`*.*
- $\texttt{verify}(\phi, ws', e, ys)$ *takes a circuit, d compressed views, a challenge, and n output shares and returns true/false,*
- $\texttt{out}(w[i])$ *takes a view and returns the output share,*
- $\texttt{rec}(ys)$ *takes a list of output shares and returns the output value of the circuit.*

After defining the syntax of a decomposition protocol, we now turn to expressing its security. We identify four properties of interest: verifiability, privacy, special soundness, and losslessness of compression.

Verifiability The first property, verifiability, captures that the views of a subset of parties in an honest execution of the protocol can be verified. Note that this property subsumes and extends correctness of the underlying MPC protocol.

Definition 4.2 (Verifiability)**.** *For any fixed ϕ accepted by the decomposition we say π is* verifiable *if for all challenges $e \in \mathcal{C}$ and inputs x,*

$$\Pr[verifiability\_game(\phi, x, e)] = 1$$

where

$$\begin{aligned}
&verifiability_game(\phi, x, e) = \{ \\
&\quad rs \leftarrow_R \mathcal{R}; \\
&\quad ws \leftarrow \texttt{decompose}(c, x, ks); \\
&\quad ys \leftarrow \texttt{map out } ws; \\
&\quad y \leftarrow \texttt{rec}(ys); \\
&\quad ws' \leftarrow \texttt{uncompress}(\texttt{compress } ws\ e); \\
&\quad valid \leftarrow \texttt{verify}(c, ws', ys); \\
&\quad \textbf{\textit{return}}\ valid \wedge \phi(x) = y \\
&\}
\end{aligned} \tag{1}$$

d-Privacy The next property, d-privacy, captures the fact that a subset of views of size d does not reveal the input to the decomposition protocol. As is common in cryptography, this privacy property is stated using simulators. Note that the simulator is required to simulate not the parties' views obtained from the `decompose` function, but their compressed versions. Moreover, the simulator should be able to produce the output shares for all n parties which are indistinguishable from real output shares.

Definition 4.3 (d-Privacy)**.** *A decomposition π is said to be* d-private *if for all challenges $e \in \mathcal{C}$ and accepted circuits ϕ there exists a PPT simulator S_e such that*

$$\forall \phi, x, e \colon real(\phi, x, e) \sim \mathtt{S_e}(\phi, c(x)) \tag{2}$$

where

$$\begin{aligned}
&real(\phi, x, e) = \{ \\
&\quad rs \leftarrow_R \mathcal{R}; \\
&\quad ws \leftarrow \texttt{decompose}(c, x, ks); \\
&\quad ys \leftarrow \texttt{map out } ws; \\
&\quad \textbf{return } (\texttt{compress } ws\ e, ys); \\
&\}
\end{aligned} \tag{3}$$

k-Special Soundness Moreover, we require k-special soundness, meaning that given multiple partial (compressed) protocol views that are consistent with each other and verify, it is possible to extract a valid input to the protocol. In particular, given any subset of views of size k, a valid input to the protocol can be extracted.

Definition 4.4 (k-Special Soundness)**.** *A decomposition π has* k-special soundness *if there exists a PPT extractor* `witness_extractor` *such that for any k tuples of $(ws'_1, es_1, ys_1), \ldots, (ws'_k, es_k, y$*

- *If $es_1, \ldots, es_k$ are pairwise different, and*
- *if the compressed views are pairwise consistent, i.e. $\forall i, i', j, i \neq i' \colon j \in \mathcal{I}(es_i) \cap \mathcal{I}(es_{i'}) \implies ws'_i[j] = ws'_{i'}[j]$. (in particular $ys_1 = \cdots = ys_k$).*
- *if each set of compressed views verifies, i.e. $\forall i, \texttt{verify}(\phi, ws'_i, es_i, ys_i) = true$,*
- *then* $\Pr[\phi(\texttt{witness\_extractor}(c, \{ws'_i, es_i\}_{\forall i}) = \texttt{rec}(\texttt{map out}(ys_1)] = 1$.

Losslessness of compression Finally, we require the compression function to be lossless and hence completely reversible.

Definition 4.5 (Lossless Compression)**.** *Let* `compress` *be a compression function with domain D.* `Compress` *is* lossless *if there exists an efficiently computable function* `uncompress` *such that for all $x \in D$,* $\texttt{uncompress}(\texttt{compress}(x)) = x$.

Decomposition Security Combining the properties above, we obtain the following security definition for decomposition protocols:

Definition 4.6 (Secure decomposition protocol)**.** *Let $k, d \in \mathbb{N}$. A decomposition protocol ϕ is* (k, d)-secure *if it has verifiability, d-Privacy, k-Special Soundness, and its decompression is lossless.*

4.2 Example: ZKBoo Decomposition Protocol

We now show that our new definition of a secure decomposition captures existing protocols on the example of ZKBoo. Further examples will be discussed in Section 4. The construction, recast in our syntax, looks as follows:

- $\mathcal{R}$ is the universe of all three element tuples (ks_1, ks_2, ks_3) where ks_i is a list of N random values.
- $\mathcal{C} = \{1, 2, 3\}$
- `out` and `rec` work exactly as before.
- `compress` selects the appropriate views from a list of views according to the challenge.

- `decompose` is a combination of Share and the gate computation functions $\phi_i^{(j)}$ from Section 3.2. Concretely, the function corresponds to steps 2 and 3 in Fig. 2.
- `verify` performs the following checks:
  - The views are well-formed.
  - The output shares $ys[e]$, $y[e+1]$ are consistent with the output gate shares in the corresponding views $ws'[e+1]$, $ws'[e+1]$
  - For $j = 1, \ldots, N$ and $i = 1, \ldots, 3$, $\phi_i^{(j)}(ws'[a], ws'[b], ws'[a], ws'[b]) = ws'[e][j]$. Here $ws'[e][j]$ denotes the share of gate j in view e.

Lemma 4.7. *The construction described above is a secure decomposition for $d = 2$ and $k = 3$.*

Proof. To show security, we need to prove verifiability, 3-special soundness, 2-privacy and losslessness of the compression. Verifiability is an extension of the original correctness proof. We observe that the well-formedness and output share consistency checks performed by `verify` are trivially true in an honest execution. The last check performs the same computations as `decompose`, just on a subset of views, which is possible given the communication pattern. 3-special soundness follows from the security of the additive secret sharing scheme that is used by `decompose`. The proof of 2-privacy carries over directly, and finally losslessness is trivial since `compress` is a projection and does not modify the individual views. □

We conclude that ZKBoo fits our general framework.

5 From Decomposition to Σ-Protocol

In this section, we show an example of a black-box construction of a Σ-protocol from the decomposition notion presented in Section 4. We focus on one of the simplest constructions based on the Σ-protocol by Giacomelli et al. [3]. As we will discuss in Section 6, this construction forms the basis for a family of secure transformations. Note that we actually obtain a stronger result than Giacomelli et al.: Our construction works for *any* secure decomposition. Moreover, we add the `compress` function for view compression to capture a greater variety of decompositions.

5.1 Example: ZKBoo Σ-Protocol

Let π be a secure decomposition and Com be a secure commitment scheme. The transformation into a Σ-Protocol is shown in Fig. 4. Observe how in comparison to Fig. 3, all references to the internal structure of the decomposition or even the circuit are removed.

For the sake of completeness, we will briefly outline the security of this protocol.

Lemma 5.1. *Let π be a secure (k,d)-decomposition, and Com a secure commitment scheme. Then the protocol described in Fig. 4 is a secure Σ-protocol.*

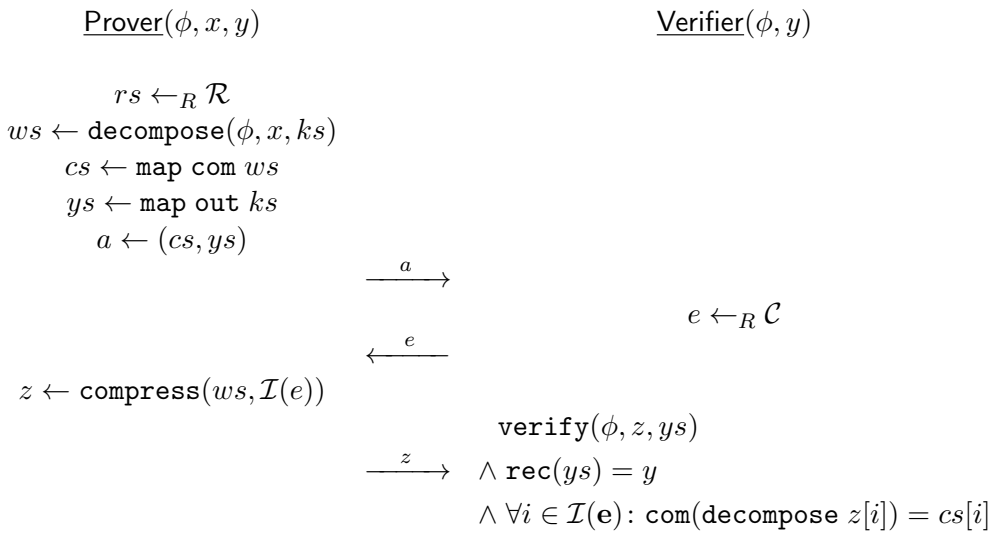

Figure 4: ZKBoo Σ-protocol construction based on secure decomposition.

Proof. Completeness of the Σ-protocol follows from the correctness of the commitment correctness, the verifiability of the decomposition, and the losslessness of view compression. Given the binding property of the commitment scheme, k-special soundness follows directly from k-special soundness. Special honest-verifier zero-knowledge is a direct consequence of d-privacy in combination with the hiding property of the commitment scheme which allows to simulate commitments. □

6 Further MPC-in-the-Head Protocols

Numerous other implementations of the MPC-in-the-head paradigm for zero-knowledge exists, in particular as optimizations of ZKBoo. We will briefly discuss in this section how they fit within our definition of a decomposition and how the corresponding transformation to a Σ-protocol changes. Note that we consider ZKBoo as the base protocol and explain how the differences of the alternative protocols fit within our framework.

6.1 ZKB++

The first protocol is ZKB++ [13]. It offers numerous optimisation for reducing the size of the messages communicated in the Σ-Protocol. The underlying MPC protocol is kept as a three party protocol with 2-Privacy, just like in ZKBoo. Optimisations are then observed in the work of the `compress` functions as well as the randomness space. Instead of sampling a long random string at the beginning, the protocol starts by sampling a short seed and expanding it into a long pseudo-random one. View compression works as follows: Given a view, the input share and the random seed used to generate all further

randomness is projected out. Since all randomness is fixed by the seed it is possible to recompute all shares of the views given the input share. The remaining algorithms for computing the decomposition and verification remain unchanged.

6.2 KKW

Another optimisation vector that was explored by Katz, Kolesnikov and Wang [9] is to replace the traditional MPC protocol by one with preprocessing. This approach splits the MPC protocol into an input-independent offline phase and an online phase where parties use their respective inputs. Essentially, correlated randomness [14] is generated during the offline phase for use in the online phase. The main observation is that since the offline phase is input-independent, revealing it completely does not compromise input privacy. Of course such a revealed offline phase cannot be used in an online phase. The work hence resorts to a trick and uses the repetition of Σ-protocol executions in their favor: Instead of repeating the protocol execution multiple times to reduce the soundness error like ZKBoo, KKW directly run m copies of the MPC protocol. The correct execution of the offline phase is verified via a cut-and-choose approach, i.e. some of the offline phase instances are completely revealed. For the remaining instances, the online phase can then be verified following the ZKBoo template, where the verifier requests the opening of a subset of party views for each instance.

In our terminology, we let $\mathcal{R}$ be the universe of all sets of size m of preprocessed data from the protocol. `decompose` then executes the online phase for each set of provided preprocessed data. `decompose` returns the input of each party, masked under the preprocessing. The challenge set $\mathcal{C}$ is then all tuples of challenges to open a subset of the preprocessing, and challenges to open all but an individual party from the MPC protocols. `compress` selects the subset of runs chosen in the challenge and reveals all preprocessing. For the remaining runs all preprocessing and views are sent, barring the view of party p.

Since this protocols requires the prover to execution multiple decomposition protocols with the same secret input, but with different initial randomness the authors added optimisations not only to the MPC protocol that is used, but also to the Σ-protocol construction. First, all randomness (preprocessing) is committed to, but instead of sending all commitments to the verifier a hash of all preprocessing concatenated is computed. Moreover, all messages of the online phase are concatenated and hashed. The hash of these two hash values are then sent to the verifier. When the prover then responds to a challenge, `compress` is used to send the preprocessing and online phase. Additionally, the commitments to the preprocessing of the unrevealed party is sent. `Uncompress` is the identity.

To verify an execution of the Σ-Protocol, the verifier first ensures that the offline and online phase are executed correctly by calling `verify`. Next, the verifier commits to all preprocessing revealed, concatenated it with the commitment of the unrevealed party (when applicable) and computes the hash. The verifier then runs `decompose`, and commits to all messages of the online phase. Lastly, the two hash values and hashed again and then compared to the value sent by the prover.

6.3 Picnic

The zero-knowledge protocol underlying the Picnic signature scheme [13] is a combination of the optimizations described above and hence fits nicely within our approach.

6.4 SNI-in-the-head

Seker et al. [15] showed ZKBoo to be susceptible to probing attacks on the exposed views of the decomposition. To mitigate this attack, the authors then proposed a change to the protocol, in particular how multiplication gates are evaluated. It is clear that the entire extension conforms to our definitions, since only the internal implementation of the `decompose` function is changed compared to ZKBoo. Since the attack vector is the exposed views of the decomposition the Σ-Protocol does not need to change.

6.5 BBQ and Banquet

BBQ [6] and Banquet [7] continue the line of optimizations of ZKB++ and KKW and adapt their approach to work with the AES blockcipher as function for the relation to be proved, i.e. the public statement is an AES ciphertext. Using AES is desirable as it is a well-studied and standardized cipher. BBQ uses an MPC protocol in the preprocessing model and can thus be expressed similarly to the KKW protocol. Banquet observes that it is sufficient to compute the verification circuit for correct AES evaluation instead of computing the AES evaluation itself. This change does not affect the applicability of our security notion. Banquet further shows how to improve in efficiency by removing the preprocessing again and using an MPC protocol specifically tailored to evaluating the AES evaluation verification, which again is a modification to the decomposition used with modifications to verification.

7 EasyCrypt Formalization

In this section we present how we checked our security proof of ZKBoo (Section 3.2) in EasyCrypt. The formalization consists of several parts: We formalize our decomposition notion introduced in Section 4.1 as well as Σ-protocols as the final security objective. Moreover, we implement our version of the ZKBoo decomposition from Section 4.2 and prove it to be a secure decomposition. Finally, we implement and prove the security of a Σ-protocol based on *any* secure decomposition to obtain a complete machine-checked security proof of ZKBoo. Assume for the rest of this section that some relation $\mathbf{R}$ is fixed, and that the Σ-protocol we construct wants to prove knowledge of a witness for a statement in the relation.

7.1 EasyCrypt

EasyCrypt [10] is a proof assistant designed specifically to capture the code-based game-playing approach to cryptographic proofs [16]. In EasyCrypt, protocols are modeled as

probabilistic programs. The tool provides an ambient higher order logic and an embedded probabilistic relational Hoare logic to reason about a probabilistic `while` language. It offers powerful automation through its interaction with SMT solvers. Proving security of a cryptographic protocol proceeds by proving a series of game transformations. Each transformation either moves a procedure call or substitutes them. This reduction is captured in the relational Hoare logic. Additionally, EasyCrypt has support for defining abstract (ML-style) modules. With abstract modules, one can formulate security specification by quantifying over all possible implementations of a module. This makes black-box style security proofs possible. In such proofs, one only relies on abstract security notions as opposed to on concrete implementation details of the protocol.

7.2 Σ-Protocol

We start by explaining the target of our formalization: Σ-protocols. As is common in EasyCrypt, we model this primitive as an abstract module. Similar to the work of Butler et al. in CryptHOL [17], we choose four procedures corresponding to the generation of the three messages exchanged as well as the final verification step. Note that we generalize their security definitions to encapsulate s-Special Soundness, rather than 2-Special Soundness. The security properties are then expressed as follows:

- Completeness:

$$\begin{aligned}&\forall h, x, e \colon \mathbf{R}\ h\ x \\ &\quad \implies \Pr[\mathit{completeness\_game}(h, x, e) = \mathit{true}] = 1.\end{aligned}$$

- s-Special Soundness:

$$\forall h, x \colon \mathbf{R}\ h\ x \implies \mathit{real}(h, x, e) \sim \mathit{ideal}(h, e)$$

- Special Honest-Verifier Zero-Knowledge:

$$\begin{aligned}&\forall h, a, es, vs \colon \\ &\quad (\forall i, 0 \le i < |es| \colon \Pr[\mathtt{sverify}(h, a, es[i], vs[i])] = 1) \\ &\quad \wedge |es| = |vs| \wedge (\forall (e, e') \in es \colon e \neq e') \\ &\quad \implies \Pr[\mathit{soudness\_game}(h, a, es, vs)] = 1\end{aligned}$$

The programs used to express the game-based security can be seen in Figure 5.

7.3 Commitments

To implement the Σ-protocol we are interested in, we need two components: a commitment scheme and a decomposition. The commitment scheme notion we use is an adaptation of the work of Butler et al. [17], and Metere and Dong [18], but we altered some game-based definitions to ones defined as relational Hoare statements. This affects the hiding property which is more conveniently stated directly as a property of

$completeness\_game(h, x, e) =$
  $a \leftarrow \texttt{scommit}(h, x);$
  $z \leftarrow \texttt{sresponse}(h, x, a, e);$
  return $\texttt{sverify}(h, a, e, z);$

$ideal(h, e) =$
  $(a, z) \leftarrow S(h, e);$
  return $(a, e, z);$

$real(h, x, e) =$
  $a \leftarrow \texttt{scommit}(h, x);$
  $z \leftarrow \texttt{sresponse}(h, x, a, e);$
  return $(a, e, z);$

$soundness\_game(h, a, es, zs) =$
  $x' \leftarrow \texttt{extractor}(h, a, es, za);$
  return $\mathbf{R}\ h\ w\ x'$

Figure 5: Σ-Protocol games

the output distribution of com for the purpose of our proofs. Again, we formalize the commitment scheme as an abstract module with procedures for the different algorithms according to Def. 2.1 and the security properties as:

- Correctness:

$$\forall m\colon \Pr[\texttt{cverify}(m, \texttt{com}(m)) = true] = 1.$$

- Hiding:

$$\forall m, m'\colon \texttt{com}(m) \sim \texttt{com}(m').$$

- Binding: $\forall c, m, m'$:

$$\Pr[\texttt{cverify}(m, c) \wedge \texttt{cverify}(m', c)] = 1 - \epsilon.$$

7.4 Decomposition

The next part is the heart of our formalization, the formalization of our decomposition notion from Section 4.

7.4.1 Circuits and Views

First, we choose representations for both the circuit and the state of each individual party. To deal with circuit evaluation, we need a method for associating gates and intermediate computations. This is similar to MPC protocols. We chose to represent both our circuit and views as lists, as this gives us a one-to-one correspondence between gates and shares: the intermediate value for circuit[i] can then be found at view[i]. Furthermore, lists allow convenient induction proofs.

7.4.2 Security

The security properties are stated as statements in (relational) Hoare logic.

- Verifiability:

$$\begin{aligned}&\forall(\phi : Circuit)(e \in \mathcal{C})(x : Input):\\ &\quad \texttt{valid\_circuit}(\phi) \implies\\ &\quad \Pr[verifiability\_game(\phi, x, e) = true] = 1.\end{aligned}$$

- d-Privacy:

$$\begin{aligned}&\forall(\phi : Circuit)(e \in \mathcal{C})(x : Input):\\ &\quad \texttt{real}(\phi, x, e) \sim \texttt{simulator}(\phi, \phi(x), e).\end{aligned}$$

  where `real` and `simulator` are defined in definition 4.3.

- s-Special Soundness:

$$\begin{aligned}&\forall(\phi : Circuit)(es \in \textbf{list}\ \mathcal{C})(vs : \textbf{list}\ view)(ys : \textbf{list}\ shares):\\ &\quad (\forall i, 0 \leq i < n\colon \Pr[\texttt{verify}(\phi, es[i]), vs[i], ys = true] = 1.)\\ &\wedge |vs| = |es| \wedge \forall (e, e') \in es\colon e \neq e' \wedge |ys| = n\\ &\wedge \texttt{valid\_circuit}(\phi) \wedge \texttt{fully\_consistent}(vs, es)\\ &\quad \implies \Pr\begin{bmatrix} c(\texttt{witness\_extractor}(\phi, \texttt{vs}, \texttt{es})) = \\ \texttt{rec}\ (\texttt{map}\ \texttt{out}\ ys)\end{bmatrix} = 1.\end{aligned}$$

- Losslessness of compression:
  $\forall(w : View), \texttt{uncompress}(\texttt{compress}(w)) = w.$

The third property uses a helper predicate $\texttt{fully\_consistent}(\{vs_1, \ldots, k\}, \{es_1, \ldots es_k\})$. A collection of list of views with respective challenges are fully consistent if the view of party constrained within two different list of views vs_a, vs_b are equivalent.

7.5 ZKBoo Decomposition

With the primitives in place, we can now describe our implementation of ZKBoo as well as the security proofs, starting with the decomposition part.

7.5.1 Computation and "communication"

The implementation of most procedures of the decomposition is straightforward, the only part that requires a bit of thought is `decompose`. This is where the gate computation function ϕ from the original ZKBoo work comes in handy. While we removed it from the decomposition notion itself, it plays a useful role in the implementation. We thus fix a procedure

$$\texttt{compute} : \textbf{list view} \times \textbf{gate} \to \textbf{list share}$$

that updates the views of all parties for gate **gate**. This updating of all shares simultaneously models the emulation of communication as required.

7.5.2 Randomness sampling

When implementing a probabilistic program there are two ways to sample randomness: lazy and eager sampling. Both are equivalent, and both are possible in EasyCrypt. *Eager sampling* samples all randomness at the start of the execution. When a new random value is needed the next unused value is used. In the case of ZKBoo, that means sampling randomness outside of the `decompose` procedure. This is necessary for the construction of a Σ-protocol as that protocol needs some control over the random choices. *Lazy sampling* on the other hand samples randomness at the moment it is needed in the protocol, and has the advantage that it enables to reason about random choices locally. In the case of proving a relational statement, one often wants to relate random choices in the two programs via a coupling, which is easier with lazy sampling. For this reason, we define two versions of `decompose`, one that takes all randomness as input, and one that samples internally, and prove them equivalent. The former is more convenient to describe the construction itself while the latter simplifies the security proof.

7.5.3 Security

We prove verifiability by showing the views produces by `decompose` are computed following the procedure outlines in Section 3.2 and reconstruct to the value of circuit evaluation. This is achieved by induction on the structure of the circuit. With this in mind, showing that `verify` will always succeed following `compress` $\circ$ `decompose` follows immediately. In particular, since `compress` is a projection, we can directly apply the invariant proven on the views of `decompose`.

Privacy is proven using a relational statement. For any valid circuit, we show that view e and $e + 1$ are identically distributed to the two simulated views. By induction on the structure of the circuit, we show that any gate can be simulated. To facilitate the proof we rewrite the procedures to use lazy sampling. With lazy sampling, we can easily manipulate the random shares in both the simulator and decomposition to make the computed shares indistinguishable. Last, we reuse the proof from verifiability that the views reconstruct to circuit evaluation. This fixes the output share of the party not simulated to be the simulated output value subtracted from the circuit evaluation.

To prove k-Special Soundness we use `fully_consistent` to derive knowledge of each view in the decomposition. Moreover, the assumption of all revealed views verifying allow us to derive that all gates of all views were computed as defined by the decomposition. To show that the input share of the revealed views gives us the secret input for the circuit evaluation we run the decomposition again. We then show by induction on the circuit that each gate computed, starting from our guess at the secret input, are equal to the shares computed in the revealed views. In particular, the output shares computed form our guess will be equal to the output shares revealed. By the reconstruction property proven during verifiability, we can conclude that our guess at the input leads to the correct reconstructed output, which is equal to the output of circuit evaluation.

7.6 Transformation to Σ-protocol

Finally, we arrive at the Σ-protocol that is our main interest. Due to the security definitions of decompositions (Def. 4.1), the transformation is black-box and can be constructed formally independent of implementation details. In the sense of our EasyCrypt formalization, this means that the construction is parameterized by an arbitrary decomposition and can be instantiated with the ZKBoo decomposition described above to yield the ZKBoo protocol.

We fix the relation of the Σ-protocol as $\mathbf{R}\ (\phi, y)\ x \iff \phi(x) = y$. The procedure implementations are seen in Figure 6.

$$
\begin{array}{l}
\texttt{scommit}((\phi, y), x) = \\
\quad ks \leftarrow \mathcal{R}; \\
\quad ws \leftarrow \texttt{decompose}(\phi, x, ks); \\
\quad cs \leftarrow \texttt{map com } ws; \\
\quad ys \leftarrow \texttt{map out } ws; \\
\quad \textbf{return } (ys, cs);
\end{array}
$$

$$
\begin{array}{l}
\texttt{sresponse}((\phi, y), (cs, ys), e) = \\
\quad z \leftarrow \texttt{compress}(ws, \mathcal{I}(e)); \\
\quad \textbf{return } z;
\end{array}
\qquad
\begin{array}{l}
\texttt{sverify}((\phi, y), (cs, ys), e, z) = \\
\quad ws \leftarrow \texttt{uncompress}z; \\
\quad v \leftarrow \forall i \in \mathcal{I}(e) \colon \texttt{cverify}(ws[i], cs[i]); \\
\quad \textbf{return } v \wedge \texttt{verify}(\phi, z, e, ys);
\end{array}
$$

Figure 6: Σ-Protocol transformation procedures

7.6.1 Security

Lemma 7.1 (Completeness)**.** *If the underlying decomposition satisfies verifiability and the commitment scheme is correct, then*

$$
\begin{array}{l}
\forall (\phi : Circuit)(e \in \mathcal{C})(x : Input) \colon \\
\quad \mathbf{R}\ h\ x \implies \Pr[Completeness(\phi, x, e) = true] = 1.
\end{array}
$$

To prove Completeness we consider the decomposition and commitment scheme parts separately. By applying verifiability of the decomposition, it is clear that the verification check will pass, since the views originate from a call to `decompose`. For the commitment scheme, we first use losslessness of the decomposition to derive that the views considered by the verifier are, in fact, identical to the ones produced by the prover. We then apply correctness of the commitment scheme to conclude that the commitments always verify given the view that was committed to.

Lemma 7.2 (SHVZK)**.** *If the underlying decomposition is d-Private, for any d, and the commitment scheme is perfectly hiding, then*

$$\begin{aligned}&\forall h, (e \in \mathcal{C}), x\colon\\ &\mathbf{R}\ h\ x \implies real(h, x, e) \sim ideal(h, e).\end{aligned}$$

Where the simulator is defined as:

$$\begin{aligned}&\texttt{simulator}((\phi, y), e) = \{\\ &\quad (z', ys) \leftarrow S_e(\phi, y);\\ &\quad ws' \leftarrow \texttt{uncompress}(z');\\ &\quad cs \leftarrow \texttt{map}\ \left(\begin{array}{l}\lambda i : \texttt{if}\ i \in \mathcal{I}(e)\\ \qquad\quad \texttt{then com}(ws'[i])\\ \qquad\quad \texttt{else com}([\,])\end{array}\right)[0..N];\\ &\quad \textbf{return}\ (ys, cs)\}\end{aligned}$$

In proving Special Honest-Verifier Zero-Knowledge, we first use d-Privacy of the decomposition to show the simulated views revealed by `compress` under challenge e are indistinguishable from the real views. The indistinguishability also implies that both `verify` and `cverify` will succeed, since their inputs are indistinguishable from the honestly generated inputs which are known to succeed. Lastly, we use the hiding property of the commitment scheme to conclude that commitments to empty lists are indistinguishable from the commitments to the unrevealed views of the decomposition.

Lemma 7.3 (s-Special Soundness)**.** *If the underlying decomposition has k-Special Soundness and the commitment scheme is binding with probability* $1 - \epsilon$*, then*

$$\begin{aligned}&\forall(\phi : Circuit), (es \in \mathcal{C})y, a, es, zs\colon\\ &\quad (\forall(e, e') \in es\colon e \neq e')\\ &\wedge |es| = |vs| = s \wedge \texttt{valid\_circuit}(c)\\ &\wedge (\exists(a \in es, b \in es, i)\colon a \neq b \wedge i \in vs[a] \wedge i \in vs[b])\\ &\wedge (\forall i, i < |es|\colon \Pr[\texttt{sverify}(c, y, a, es[i], vs[i]) = true] = 1)\\ &\quad \implies \Pr[soundness\_game((\phi, y), a, e, z) = true] = (1 - \epsilon).\end{aligned}$$

From k-Special Soundness of the decomposition, it follows that we can extract a valid witness for the relation. The assumptions of s-Special Soundness must therefore imply the assumptions of k-Special Soundness from the decomposition. Concretely, this is achieved by proving:

$$\begin{aligned}&(\forall i, i < |es|\colon \Pr[\texttt{sverify}(\phi, y, a, es[i], vs[i]) = true] = 1)\\ &\implies \texttt{fully\_consistent}(\texttt{vs}, \texttt{es}).\end{aligned}$$

To show this, we use the binding property of the commitment scheme. We assume that we are given enough responses, such that at least two responses will overlap on at

least one view. With this overlap in mind, it follows from the binding property that the two different openings are equivalent.

From the overlap and the proof of equivalence, we derive that the responses are fully consistent.

8 Related Work

We list work on formal verification of zero-knowledge protocols.

Computational Analysis One approach is to formalize security proofs of zero-knowledge protocols, which is also the focus of this work. Previous work in this direction includes ZKCrypt by Almeida et al. [19] which automatically generates CertiCrypt proofs [20] of the resulting protocols. The work of Butler, Aspinall and Gascón [17] focuses on formalizing Σ-protocols in CryptHOL [21]. Both have in common that they focus on simpler algebraic protocols, like proving knowledge of pre-images under group homomorphisms. This limits usability to problems which exhibit this simpler algebraic structure. The present work formalizes more sophisticated protocols in which security is reduced to the security of complex building blocks like MPC protocols. The zero-knowledge protocols that we study use secret-sharing-based MPC as building block. This type of MPC protocol has been formalized previously by Butler, Aspinall and Gascón [22] and Haagh et al. [23]. Our MPC protocol formalization is close in spirit to the passive security construction of Haagh et al., yet it differs in that we directly formalize a simulation-based security notion which is more familiar to cryptographers than the non-interference used there.

Symbolic Analysis An orthogonal line of work studies the symbolic security of protocols that use zero-knowledge protocols as primitives [24, 25]. In this setting, the zero-knowledge proofs themselves are treated as abstract objects that can be manipulated according to fixed rules modeled as equational theory. Symbolic security of a protocol then rules out any attack that follows only those allowed manipulations. This approach cannot capture the security of a concrete zero-knowledge protocol, but only of another protocol that uses it.

9 Discussion and Future Work

This present work shows how formal verification cannot only recreate existing proofs, but also foster a deeper understanding of the object in question. In our case, we set out to formalize the ZKBoo security proof and found out that what looked like a modular proof structure was actually not as modular as it could be. Obvious future work includes extending our efforts to more efficient protocols following the MPC-in-the-head paradigm, in particular in case any of them becomes standardized. As mentioned, Picnic was recently announced as alternate in the third round of the NIST post-quantum

cryptography standardization competition. The reason Picnic is an alternate and not a candidate is because of the variety of proposals that were published after the submission of Picnic. Once the line of research converges to one or more efficient constructions ready for standardization, we expect our work to form the basis of further formal verification efforts, possibly even connecting our work with an actual implementation. On the other hand, the structures that we identified in this work can hopefully enhance the understanding of the MPC-in-the-head paradigm and provide insights into possibilities for further optimization and constructions.

10 Conclusion

We initiated the formal analysis of zero-knowledge protocols following the MPC-in-the-head paradigm. Based on the observation that existing constructions are black-box in the MPC protocol they use while their security analysis is not, we proposed a new security notion for these MPC protocols. This modular security proof then enabled us to develop a machine-checked security proof of the ZKBoo protocol in EasyCrypt as an example of this protocol class.

References

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

[2] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Zero-knowledge from secure multiparty computation," in *39th ACM STOC*, D. S. Johnson and U. Feige, Eds. ACM Press, Jun. 2007, pp. 21–30.

[3] I. Giacomelli, J. Madsen, and C. Orlandi, "ZKBoo: Faster zero-knowledge for Boolean circuits," in *USENIX Security 2016*, T. Holz and S. Savage, Eds. USENIX Association, Aug. 2016, pp. 1069–1083.

[4] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO'86*, ser. LNCS, A. M. Odlyzko, Ed., vol. 263. Springer, Heidelberg, Aug. 1987, pp. 186–194.

[5] G. Zaverucha, M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, J. Katz, X. Wang, V. Kolesnikov, and D. Kales, "Picnic," National Institute of Standards and Technology, Tech. Rep., 2020, available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[6] C. de Saint Guilhem, L. De Meyer, E. Orsini, and N. P. Smart, "BBQ: Using AES in picnic signatures," in *SAC 2019*, ser. LNCS, K. G. Paterson and D. Stebila, Eds., vol. 11959. Springer, Heidelberg, Aug. 2019, pp. 669–692.

[7] C. Baum, C. Delpech de Saint Guilhem, D. Kales, E. Orsini, P. Scholl, and G. Zaverucha, "Banquet: Short and fast signatures from AES," *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 68, 2021. [Online]. Available: https://eprint.iacr.org/2021/068

[8] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha, "Post-quantum zero-knowledge and signatures from symmetric-key primitives," Cryptology ePrint Archive, Report 2017/279, 2017, http://eprint.iacr.org/2017/279.

[9] J. Katz, V. Kolesnikov, and X. Wang, "Improved non-interactive zero knowledge with applications to post-quantum signatures," in *ACM CCS 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM Press, Oct. 2018, pp. 525–537.

[10] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella Béguelin, "Computer-aided security proofs for the working cryptographer," in *CRYPTO 2011*, ser. LNCS, P. Rogaway, Ed., vol. 6841. Springer, Heidelberg, Aug. 2011, pp. 71–90.

[11] I. Damgaard, "On Σ-protocols," lecture notes, Aarhus University, 2011.

[12] D. Pointcheval and J. Stern, "Security proofs for signature schemes," in *EUROCRYPT'96*, ser. LNCS, U. M. Maurer, Ed., vol. 1070. Springer, Heidelberg, May 1996, pp. 387–398.

[13] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha, "Post-quantum zero-knowledge and signatures from symmetric-key primitives," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM Press, Oct. / Nov. 2017, pp. 1825–1842.

[14] Y. Ishai, E. Kushilevitz, S. Meldgaard, C. Orlandi, and A. Paskin-Cherniavsky, "On the power of correlated randomness in secure computation," in *TCC 2013*, ser. LNCS, A. Sahai, Ed., vol. 7785. Springer, Heidelberg, Mar. 2013, pp. 600–620.

[15] O. Seker, S. Berndt, L. Wilke, and T. Eisenbarth, "SNI-in-the-head: Protecting MPC-in-the-head protocols against side-channel analysis," in *ACM CCS 20*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM Press, Nov. 2020, pp. 1033–1049.

[16] M. Bellare and P. Rogaway, "The security of triple encryption and a framework for code-based game-playing proofs," in *EUROCRYPT 2006*, ser. LNCS, S. Vaudenay, Ed., vol. 4004. Springer, Heidelberg, May / Jun. 2006, pp. 409–426.

[17] D. Butler, D. Aspinall, and A. Gascón, "On the formalisation of Σ-protocols and commitment schemes," in *Principles of Security and Trust - 8th International Conference, POST 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, ser. Lecture Notes in Computer Science, F. Nielson and D. Sands, Eds., vol. 11426. Springer, 2019, pp. 175–196.

[18] R. Metere and C. Dong, "Automated cryptographic analysis of the pedersen commitment scheme," *CoRR*, vol. abs/1705.05897, 2017. [Online]. Available: http://arxiv.org/abs/1705.05897

[19] J. B. Almeida, M. Barbosa, E. Bangerter, G. Barthe, S. Krenn, and S. Zanella Béguelin, "Full proof cryptography: verifiable compilation of efficient zero-knowledge protocols," in *ACM CCS 2012*, T. Yu, G. Danezis, and V. D. Gligor, Eds. ACM Press, Oct. 2012, pp. 488–500.

[20] G. Barthe, B. Grégoire, and S. Z. Béguelin, "Formal certification of code-based cryptographic proofs," in *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, Z. Shao and B. C. Pierce, Eds. ACM, 2009, pp. 90–101.

[21] D. A. Basin, A. Lochbihler, and S. R. Sefidgar, "CryptHOL: Game-based proofs in higher-order logic," *Journal of Cryptology*, vol. 33, no. 2, pp. 494–566, Apr. 2020.

[22] D. Butler, D. Aspinall, and A. Gascón, "How to simulate it in isabelle: Towards formal proof for secure multi-party computation," in *Interactive Theorem Proving - 8th International Conference, ITP 2017, Brasília, Brazil, September 26-29, 2017, Proceedings*, ser. Lecture Notes in Computer Science, M. Ayala-Rincón and C. A. Muñoz, Eds., vol. 10499. Springer, 2017, pp. 114–130.

[23] H. Haagh, A. Karbyshev, S. Oechsner, B. Spitters, and P.-Y. Strub, "Computer-aided proofs for multiparty computation with active security," in *CSF 2018Computer Security Foundations Symposium*, S. Chong and S. Delaune, Eds. IEEE Computer Society Press, 2018, pp. 119–131.

[24] M. Backes and D. Unruh, "Computational soundness of symbolic zero-knowledge proofs against active attackers," in *CSF 2008Computer Security Foundations Symposium*, A. Sabelfeld, Ed. IEEE Computer Society Press, 2008, pp. 255–269.

[25] M. Backes, M. Maffei, and D. Unruh, "Zero-knowledge in the applied Pi-calculus and automated verification of the direct anonymous attestation protocol," in *2008 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2008, pp. 202–215.