Karlsruhe Institute of Technology

KASTEL

Modular Construction and Analysis of Multi-Round Straightline Extractability Compilers

Bachelor's Thesis of

Jannis Wunderlich

At the KIT Department of Informatics
KASTEL – Institute of Information Security and Dependability

First examiner:	Prof. Dr. Jörn Müller-Quade
Second examiner:	Prof. Dr. Thorsten Strufe
First advisor:	Dr. Michael Klooß

15. November 2025 – 15. April 2026

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Modular Construction and Analysis of Multi-Round Straightline Extractability Compilers (Bachelor's Thesis)

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Karlsruhe, 15. April 2026

. . . . . .

(Jannis Wunderlich)

Abstract

This thesis investigates the feasibility of a generic, modular compiler that treats an existing three-move straight-line extractability compiler as a black-box building block and extends it to multi-round public-coin protocols. We introduce a recursive compiler that compresses an interactive multi-round protocol layer by layer and analyze how its security and efficiency properties propagate through this recursion. The analysis demonstrates that while the framework successfully preserves completeness, zero-knowledge, and knowledge soundness, the modularity incurs a substantial cost. The repetition parameters required at each recursive level cause an exponential blowup in both runtime and communication complexity, restricting the compiler's efficiency to protocols with a strictly constant number of rounds. Furthermore, preserving the zero-knowledge property requires the input protocol to satisfy 2-Stage special Honest-Verifier Zero-Knowledge, a stronger form of special Honest-Verifier Zero-Knowledge introduced in this work. Ultimately, this thesis provides a clean theoretical framework for modular straight-line extractability, but exposes the asymptotic performance gap between our flexible black-box construction and specialized monolithic transformations.

Zusammenfassung

Diese Arbeit untersucht die Machbarkeit eines generischen, modularen Compilers, der einen existierenden 3-Zug Straight-Line-Extractability-Compiler als Black-Box-Baustein behandelt und auf mehrrundige Public-Coin-Protokolle erweitert. Wir stellen einen rekursiven Compiler vor, der ein mehrrundiges Protokoll Schicht für Schicht komprimiert, und analysieren systematisch, wie sich Sicherheits- und Effizienzgarantien durch die Rekursion verändern. Die Analyse zeigt, dass die modulare Konstruktion zwar Completeness, Zero-Knowledge und Knowledge-Soundness bewahrt, dies jedoch mit erheblichen Kosten verbunden ist. Die notwendigen Wiederholungsparameter auf jeder Rekursionsebene führen zu einer exponentiellen Erhöhung der Laufzeit und Kommunikationskomplexität, was die Effizienz des Compilers auf Protokolle mit einer konstanten Anzahl von Runden beschränkt. Darüber hinaus erfordert der Erhalt der Zero-Knowledge-Eigenschaft, dass das Eingabeprotokoll eine neue starke Bedingung erfüllt: 2-Stage special Honest-Verifier Zero-Knowledge. Zusammengefasst liefert diese Arbeit ein theoretisches Framework für modulare Straight-Line-Extrahierbare Compiler, verdeutlicht jedoch explizit den asymptotischen Leistungsunterschied zwischen flexiblen Black-Box-Konstruktionen wie unserer und spezialisierten monolithischen Transformationen.

Contents

List of Figures

List of Tables

1. Introduction

Zero-knowledge proof systems are a central tool in modern cryptography because they allow a prover to convince a verifier of the validity of a statement without revealing the corresponding secret. Among the simplest and most widely studied zero-knowledge protocols are Σ-protocols. In such a protocol, the prover first sends an initial message, the verifier responds with a random challenge, and the prover finally returns a response from which the verifier can check correctness. While this interactive structure is elegant and powerful, it also constitutes an important practical limitation: the prover and verifier must communicate. This restricts usability in settings where interaction is costly, or where the prover needs to prove the statement to multiple verifiers.

A fundamental step toward overcoming this limitation was the Fiat-Shamir transformation [FS86], which introduced the idea of replacing the verifier's random challenge by a hash query in the Random Oracle Model (ROM), thereby enabling non-interactive proofs. Since then, many works have taken inspiration from this idea, each offering different trade-offs in terms of efficiency, soundness, and zero-knowledge. Of particular interest for this thesis are compilers that achieve straight-line extractability, meaning that a knowledge extractor can recover a witness without rewinding the prover. This property is especially useful because rewinding can become problematic in concurrent or non-resettable environments. Classic straight-line extractable compilers in the ROM include the transformations of Pass [Pas03, Sec. 4.2] and Fischlin [Fis05, Sec. 3.1], as well as the later efficiency improvement from Kondi and Shelat [KS22, Sec. 5]. However, these constructions also have one important limitation namely, that they are only designed for three-move Σ-protocols and do not directly address more general multi-round public-coin protocols.

This restriction is significant because many proof systems such as bulletproofs stretch over multiple rounds. Attema, Fehr and Klooß [AFK23] introduced a compiler which relies on the Fiat-Shamir heuristic and is able to transform a $(2\mu + 1)$-move interactive protocol into a non-interactive one. However, since it relies on the Fiat-Shamir heuristic, their solution is not straight-line extractable. Recent work by Rotem and Tessaro [RT25] shows that straight-line extractability can, in fact, be extended to multi-round protocols. They presented two monolithic compilers which adapt the Fischlin and Pass compiler to the multi-round setting. While this approach is effective, the presented compilers

are specific and not generic, meaning that they cannot easily be adapted to other underlying sle-compilers. Therefore efficiency or security improvements from new compilers cannot be directly used in the multi-round setting since even when a better single-round construction is available, its use in the multi-round setting generally requires a new security analysis from scratch.

The goal of this thesis is to study whether a generic multi-round compiler is possible. More precisely, we investigate whether one can construct a modular straight-line extractability compiler that treats an existing compiler for Σ-protocols as a black-box building block and extends it to handle $(2\mu + 1)$-move public-coin protocols. The appeal of such a modular approach is that it makes existing and future single-round compilers interchangeable components in a more general framework. One could then derive properties of the resulting multi-round construction — such as completeness, zero-knowledge, and knowledge soundness — systematically from the corresponding properties of the underlying three-move compiler.

In this thesis we introduce a recursive compiler that compresses a multi-round protocol layer by layer by repeatedly applying a given straight-line extractable 3-move compiler to suitable intermediate Σ-protocols. We then analyze how the security and efficiency guarantees of the underlying compiler propagate through this recursion. In particular, the thesis studies completeness, efficiency, zero-knowledge and soundness and uses the compilers of Fischlin, Pass, and Kondi-Shelat as examples.

Our analysis shows that modularity comes at a substantial cost. Although the recursive construction is conceptually appealing, the repetition parameters required at each level lead to an exponential blowup in runtime and communication complexity. As a result, the compiler remains efficient only for protocols with a constant number of rounds. Moreover, the zero-knowledge analysis requires the input protocol to satisfy 2-Stage special Honest-Verifier Zero-Knowledge, a strong property which we introduce in this thesis and further limits the applicability of our compiler. The main contribution of this thesis is therefore twofold: on the one hand, it provides a generic framework for lifting straight-line extractable compilers from the three-move to the multi-round setting; on the other hand, it makes the asymptotic price of this modularity explicit and thereby clarifies the gap between a flexible black-box construction and the more specialized, asymptotically stronger transformations of Rotem and Tessaro.

Parallel to the line of work on sle-compilers in the ROM, the folklore technique known as *encryption to the sky* operates in the Common Reference String (CRS) model and pursues a similar goal, achieving straight-line extractability. Instead of modifying the proof structure, these CRS-based approaches include an encryption of the witness directly into the proof. The witness is encrypted using a public key for which no party knows the corresponding secret key. This makes the proof size at least as large as the

witness, but in many applications the witness is relatively small anyway. Since the resulting proof systems can be significantly more efficient overall, they are widely used in practice.

2. Definitions

We begin by introducing a few parameters used throughout the thesis. For a full list of parameters we refer to Table A.1.

λ	the security parameter
$q_{\mathrm{RO},\mathcal{A}}$	the maximum number of random-oracle queries an attacker has
$q_{\mathrm{RO},P}$	the maximum number of random-oracle queries a prover has
s	the maximum length of the input statement $\mathbb{x}$
σ	the length of the output of the $\mathcal{RO}$

Definition 1 (Σ-protocols)**.** *Let $\Pi^{\Sigma} = (P, V)$ be an interactive protocol for a relation $R \subseteq \mathcal{X} \times \mathcal{W}$. We call Π^{Σ} a Σ-protocol if it consists of exactly 3 messages in the following order: a_1 being the first message by the prover to the verifier, c_1 being a challenge sent by the verifier to the prover, and a_2 being the response by the prover to the verifier.*

Definition 2 (Public Coin, [AFK23, Def. 3])**.** *An interactive Protocol $\Pi = (P, V)$ is public-coin if all of V's random choices are made public. The message $c_i \leftarrow C_i$ of V in the $2i$'th move is called the i'th challenge and C_i is the challenge set.*

Definition 3 (The Random Oracle Model (ROM), [GRY25, Def. 3.1])**.** *For every $\sigma \in \mathbb{N}$, the Random Oracle $\mathcal{RO}$ is the uniform distribution over the set of all functions $\mathrm{f} : \{0,1\}^* \to \{0,1\}^{\sigma}$*

Definition 4 (Non-interactive argument (NARG), [CY24, Ch. 4.1])**.** *A non-interactive argument (NARG) in the ROM is a tuple (P, V) where P is an oracle algorithm known as the argument prover and V is a deterministic oracle algorithm known as argument verifier.*

Any party, especially P and V can query an existing random oracle $\mathcal{RO}$. The argument prover P receives as input an instance $\mathbb{x}$ and witness $\mathbb{w}$ for a relation $R \subseteq \mathcal{X} \times \mathcal{W}$ (such that $(\mathbb{x}, \mathbb{w}) \in R$), and outputs an argument string π. The argument verifier V receives as input the instance $\mathbb{x}$ and argument string π, and outputs a bit denoting whether to accept (the bit is 1) or reject (the bit is 0).

We say that NARG = (P, V) is a non-interactive argument in the ROM for a relation R if it satisfies two main properties: completeness (Definition 6) and soundness (Definition 17).

For a protocol to be considered a "zero-knowledge NARG", in addition to completeness and knowledge-soundness, we require it to satisfy the zero-knowledge property stated in Definition 10.

We begin by giving a definition for completeness in the interactive and non-interactive case. Loosely speaking the completeness error is the probability, that an honest prover cannot convince a verifier. Consequently, if the completeness error is too high, it is not practical to use the protocol.

Definition 5 (Completeness of an Interactive Protocol)**.** *Let $\Pi = (P, V)$ be a public coin interactive protocol for a relation $R \subseteq \mathcal{X} \times \mathcal{W}$. We say that (P, V) has completeness error $\epsilon(\lambda, s)$ if for every $(\mathbb{x}, \mathbb{w}) \in R$, it holds that:*

$$Pr\left[\langle P(\mathbb{w}), V\rangle(\mathbb{x}) = 1\right] \geq 1 - \epsilon(\lambda, s)$$

where $\langle P(\mathbb{w}), V\rangle(\mathbb{x})$ denotes the output of the verifier V after interacting with the prover P (who has witness $\mathbb{w}$) on common input $\mathbb{x}$.

Definition 6 (Completeness of a Non-Interactive Random Oracle Argument, [RT25, Def. 3])**.** *Let $\Pi_{\mathsf{ni}}(P^{\mathsf{ni}}, V^{\mathsf{ni}})$ be a non-interactive random oracle argument for a relation $R \subseteq \mathcal{X} \times \mathcal{W}$. We say that $(P^{\mathsf{ni}}, V^{\mathsf{ni}})$ has completeness error $\epsilon(\lambda, q_{\mathrm{RO},P}, s, \sigma)$ if:*

$$Pr\left[V^{\mathcal{RO}}(1^\lambda, \mathbb{x}, \pi) = 1 : \pi \leftarrow P^{\mathcal{RO}}(1^\lambda, \mathbb{x}, \mathbb{w})\right] \geq 1 - \epsilon(\lambda, q_{\mathrm{RO},P}, s, \sigma)$$

where the probability is taken over the randomness of P and the choice of $\mathcal{RO}$.

We now introduce the concept of Zero-Knowledge in its various forms. If a protocol is *Zero-Knowledge*, it guarantees that the verifier learns no information about the secret witness, $\mathbb{w}$, from the proof. For interactive proofs, we define a specific variant called *special Honest-Verifier Zero-Knowledge* (sHVZK), which assumes the verifier follows the protocol honestly. sHVZK guarantees the existence of an efficient simulator that, given a specific challenge and without access to the witness, can produce a proof transcript indistinguishable from a real protocol execution. The intuition is that if an attacker cannot distinguish between a proof constructed with the actual witness and a simulated proof constructed without it, no knowledge about the witness could have been revealed.

Definition 7 (special Honest-Verifier Zero-Knowledge, [Boo+16, Def. 9])**.** *Let $\Pi^\mu = (P, V)$ be a $(2\mu + 1)$-move public-coin interactive protocol for a relation $R \subseteq \mathcal{X} \times \mathcal{W}$. We say that Π^μ is* special Honest-Verifier Zero-Knowledge *with error $\epsilon_{\mathsf{zk}}(\lambda, \sigma)$ if there exists*

a probabilistic polynomial-time simulator Sim such that for every $(\mathbb{x}, \mathbb{w}) \in R$ *with* $|\mathbb{x}| \leq s$*, if* ρ *denotes the public-coin randomness of the verifier, then*

$$\Delta(\langle P(\mathbb{x}, \mathbb{w}), V(\mathbb{x}; \rho)\rangle,\ Sim(\mathbb{x}, \rho)) \leq \epsilon_{\mathsf{zk}}(\lambda, \sigma),$$

where $\Delta(\cdot, \cdot)$ *denotes the statistical distance between the distribution of the real transcript* $\langle P(\mathbb{x}, \mathbb{w}), V(\mathbb{x}; \rho)\rangle$ *and the distribution output by the simulator* $Sim(\mathbb{x}, \rho)$*.*

Definition 8 (2-Stage Protocol)**.** *Let* $\Pi^{\mu} = (P, V)$ *be an interactive,* $(2\mu + 1)$*-move public-coin protocol for a relation R. We call* Π^{μ} *a 2-Stage Protocol if the prover algorithm P can be decomposed into two sub-algorithms* P_1 *and* P_2 *such that for any statement-witness pair* $(\mathbb{x}, \mathbb{w}) \in R$*:*

1. ***Stage 1:*** *The prover generates the first message* a_1 *and, based on the verifier's first challenge* $c_1 \in C$*, computes an internal state* st_1*:*

$$(a_1, c_1, st_1) \leftarrow P_1(\mathbb{x}, \mathbb{w}, c_1)$$

2. ***Stage 2:*** *The remainder of the protocol can be executed without further access to the witness* $\mathbb{w}$*. For all subsequent rounds* $i \in \{2, \cdots, \mu + 1\}$*, the prover computes its messages* a_i *using only the public statement* $\mathbb{x}$*, the public parameters pp, the state* st_1*, and the history of challenges* $(c_1, \cdots, c_{i-1})$*:*

$$a_i \leftarrow P_2(\mathbb{x}, pp, st_1, c_1, \cdots, c_{i-1})$$

For the definition of 2-Stage sHVZK, we adapt our definition for sHVZK:

Definition 9 (2-Stage special Honest-Verifier Zero-Knowledge)**.** *Let* $\Pi = ((P_1, P_2), V)$ *be an interactive, 2-stage Protocol for a relation R. We say that it is 2-Stage special Honest-Verifier Zero-Knowledge (2SsHVZK) if there exists a probabilistic polynomial time simulator Sim, such that for all stateful interactive non-uniform adversaries* $\mathcal{A}$ *the difference between the following two probabilities is at most* $\epsilon_{2S}(s, \lambda)$*:*

$$\Pr[(\mathbb{x}, \mathbb{w}, \rho) \leftarrow \mathcal{A}^{\lambda}; (a_1, c_1, st_1) \leftarrow \langle P_1(\mathbb{x}, \mathbb{w}), V(\mathbb{x}, \rho)\rangle : (\mathbb{x}, \mathbb{w}) \in R \text{ and } \mathcal{A}(\mathbb{x}, a_1, c_1, st_1) = 1]$$
$$\Pr[(\mathbb{x}, \mathbb{w}, \rho) \leftarrow \mathcal{A}^{\lambda}; (a_1, c_1, st_1) \leftarrow Sim(\mathbb{x}, \rho) : (\mathbb{x}, \mathbb{w}) \in R \text{ and } \mathcal{A}(\mathbb{x}, a_1, c_1, st_1) = 1]$$

We now define Zero-Knowledge for non-interactive protocols in the ROM. Because the verifier does not actively participate in the creation of the proof, we no longer need to assume honest behavior.

Definition 10 (Zero-Knowledge in the ROM, [KR25, Def. B.2]). *Let $\Pi_{\mathsf{ni}} = (P, V)$ be a non-interactive proof system for a relation $R \subseteq \mathcal{X} \times \mathcal{W}$ in the Random Oracle Model. We say that Π_{ni} satisfies straight-line zero-knowledge with zero-knowledge error $\epsilon_{zk}(\lambda, q_{\mathrm{RO},P}, s, \sigma)$ if there exists a probabilistic polynomial-time (PPT) straight-line simulator Sim such that for all statements of maximum size s and adversaries $\mathcal{A}$ who can do at most $q_{\mathrm{RO},\mathcal{A}}$ queries to the $\mathcal{RO}$, the following holds:*

$$\left|\Pr[ExpZK^{\Pi_{\mathsf{ni}}}_{Real,\mathcal{A}}(\lambda) = 1] - \Pr[ExpZK^{\Pi_{\mathsf{ni}}}_{Sim,\mathcal{A}}(\lambda) = 1]\right| \leq \epsilon_{zk}(\lambda, q_{\mathrm{RO},P}, s, \sigma)$$

The two experiments are defined as follows:

- $\boldsymbol{ExpZK}^{\Pi_{\mathsf{ni}}}_{\boldsymbol{Real},\mathcal{A}}(\lambda)$*: The adversary $\mathcal{A}$ is given oracle access to a true random oracle $\mathcal{RO}$ and a proving oracle O_{Prove}. On input $(\mathbb{x}, \mathbb{w}) \in R$, the oracle O_{Prove} returns an honestly generated proof $\pi \leftarrow P^{\mathcal{RO}}(\mathbb{x}, \mathbb{w})$. $\mathcal{A}$ eventually outputs a bit $b \in \{0, 1\}$, which is the output of the experiment.*
- $\boldsymbol{ExpZK}^{\Pi_{\mathsf{ni}}}_{\boldsymbol{Sim},\mathcal{A}}(\lambda)$*: The adversary $\mathcal{A}$ is given oracle access to a simulated random oracle $\mathcal{RO}_{Sim}$ and a simulated proving oracle O_{Sim}, both controlled by the straight-line simulator Sim. On input $(\mathbb{x}, \mathbb{w}) \in R$, O_{Sim} ignores the witness $\mathbb{w}$ and returns a simulated proof $\pi \leftarrow Sim(\mathbb{x})$. The simulator Sim answers all direct random oracle queries made by $\mathcal{A}$ and may program the oracle to ensure the validity of π. $\mathcal{A}$ eventually outputs a bit $b \in \{0, 1\}$, which is the output of the experiment.*

Finally, we present the third property, *knowledge soundness*. Intuitively, a prover who can convince the verifier with non-negligible probability should know a valid witness $\mathbb{w}$.

To formalize this notion, we rely on protocols satisfying *special soundness*, which guarantees that a witness can be efficiently recovered from a suitable tree of accepting transcripts. A knowledge-sound protocol is then constructed so that any prover who succeeds with non-negligible probability induces such a transcript tree through its interaction with the random oracle with overwhelming probability. Importantly, this tree is not sent explicitly to the verifier, since doing so would generally destroy zero knowledge.

Instead, we define an efficient *extractor*. In the Random Oracle Model, this extractor can read every query made to the $\mathcal{RO}$. By observing a successful prover e.g. through his queries to the $\mathcal{RO}$, the extractor can — with high probability — extract a transcript tree from which it can compute the witness due to special soundness.

Knowledge soundness therefore requires that whenever a prover can produce an accepting proof with non-negligible probability, there exists an efficient extractor that can recover a witness from that prover's behaviour.

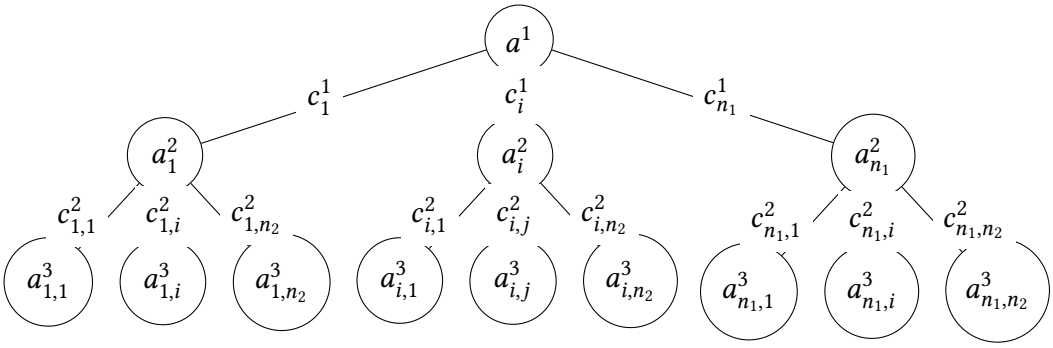


Figure 2.1.: A transcript tree for $\mu = 2$

To define special-soundness for multi-round protocols, we need to introduce the concept of transcript trees as first presented by [AFK23, Def. 6].

Definition 11 ($(n_1, \cdots, n_\mu)$-Transcript Tree)**.** *Let $n_1, \cdots, n_\mu, \mu \in \mathbb{N}$. Furthermore let $\Pi^\mu = (P, V)$ be a $(2\mu + 1)$-move public coin interactive protocol.*

*A $(n_1, \cdots, n_\mu)$-transcript tree contains $N = \Pi_{i=1}^{\mu} n_i$ **distinct**, valid transcripts for the same pair $(\mathbb{x}, \mathbb{w})$ in the following structure. The nodes represent P's messages, while the edges are the challenges. Starting from the root with depth 1, each node on depth j has exactly n_j outgoing edges and therefore children. One transcript therefore is a path from the root to a leaf.*

Definition 12 (Special-Soundness Tree, [AFK23][Def. 7])**.** *Let T be a $(n_1, \cdots, n_\mu)$-Transcript Tree according to Definition 11. We say T is a $(n_1, \cdots, n_\mu)$ Special-Soundness Tree if for every node, all outgoing challenge labels are pairwise distinct: $(\forall j \neq k : c_i^j \neq c_i^k)$*

Definition 13 ($(n_1, \cdots, n_\mu)$-Special-Soundness, [RT25][Def. 2])**.** *Let $\mu \geq 1$ and let (P, V) be an $(2\mu + 1)$-move public coin argument for a relation $R \subseteq \mathcal{X} \times \mathcal{W}$, and let $n_1, \cdots, n_\mu \in \mathbb{N}$. We say that (P, V) satisfies $(n_1, \cdots, n_\mu)$-special-soundness if there exists a deterministic polynomial-time algorithm $\mathcal{E}$ such that on input $\mathbb{x} \in X$ and a $(n_1, ..., n_\mu)$-Special-Soundness Tree T, $\mathcal{E}$ outputs a witness $\mathbb{w}$ such that $(\mathbb{x}, \mathbb{w}) \in R$.*

Definition 14 ($(n_1, \cdots, n_\mu)$-Strong-Special-Soundness Tree)**.** *Let T be a $(n_1, \cdots, n_\mu)$-transcript tree according to Definition 11. We say that T is an* $(n_1, \cdots, n_\mu)$-strong-special-soundness tree *if for every node of T, every pair of distinct outgoing edges corresponds to two transcript extensions such that either their challenge labels are distinct, or, if the challenge labels are equal, then the corresponding next prover messages are distinct.*

Definition 15 ($(n_1, \cdots, n_\mu)$-strong-Special-Soundness, [KS22][Def. 3.2]). *Let $\mu \geq 1$ and let (P, V) be an $(2\mu + 1)$-move public coin argument for a relation $R \subseteq \mathcal{X} \times \mathcal{W}$, and let $n_1, \cdots, n_\mu \in \mathbb{N}$. We say that (P, V) satisfies $(n_1, \cdots, n_\mu)$-strong-special-soundness if there exists a deterministic polynomial-time algorithm $\mathcal{E}$ such that on input $\mathbb{x} \in X$ and a $(n_1, \cdots, n_\mu)$ Strong-Special-Soundness Tree $\mathcal{E}$ outputs a witness $\mathbb{w}$ such that $(\mathbb{x}, \mathbb{w}) \in R$.*

Definition 16 (sle-Compiler). *Let $\Pi = (P, V)$ be a public-coin interactive protocol for a relation R. A Compiler C is a transformation in the Random Oracle Model which transforms any interactive input-protocol Π into a non-interactive protocol $[\Pi_{\mathsf{ni}} = (P^{\mathsf{ni}}, V^{\mathsf{ni}})] \leftarrow C(\Pi)$. We say that C is a straight-line extractable (sle)-Compiler if the resulting protocol Π_{ni} satisfies Straight-Line knowledge-soundness.*

Definition 17 (Straight-Line knowledge-soundness). *Let $\Pi_{\mathsf{ni}} = (P, V)$ be a non-interactive proof system for the corresponding verification relation R in the Random Oracle Model. Furthermore let $q_{\mathrm{RO},\mathcal{A}}$ be the maximum number of queries that an adversarial Prover can make to the random oracle and let s be the maximum size of the instance $\mathbb{x}$. We say that Π_{ni} is knowledge sound for extraction-Relation $\tilde{R}$ with knowledge-soundness error $\kappa(\lambda, q_{\mathrm{RO},\mathcal{A}}, s, \sigma)$ if:*

$$\exists\ PPT\ \mathcal{E} \quad \forall A : AdvKS_A(\lambda, q_{\mathrm{RO},\mathcal{A}}, s, \sigma) := \Pr[ExpKS_A(\lambda, q_{\mathrm{RO},\mathcal{A}}, s, \sigma) = 1] \leq \kappa(\lambda, q_{\mathrm{RO},\mathcal{A}}, s, \sigma)$$

The experiment $ExpKS_A^{(P,V)}(\lambda, q_{\mathrm{RO},\mathcal{A}}, s, \sigma)$ is defined as follows:

$$\begin{aligned}
&ExpKS_A^{(P,V)}(\lambda, q_{\mathrm{RO},\mathcal{A}}, s, \sigma) : \\
&\quad (\mathbb{x}, \pi) \leftarrow A^{\mathcal{RO}} \text{ (where } \mathcal{A} \text{ makes at most } q_{\mathrm{RO},\mathcal{A}} \text{ queries to the } \mathcal{RO}) \\
&\quad \mathbb{w} \leftarrow \mathcal{E}(\mathbb{x}, \pi, Q_{\mathcal{RO}}) \\
&\quad \text{return } (\mathbb{x}, \mathbb{w}) \notin \tilde{R} \wedge V^{\mathcal{RO}}(\mathbb{x}, \pi) = 1
\end{aligned}$$

Here, $Q_{\mathcal{RO}}$ denotes the set of random oracle queries made by $\mathcal{A}$ with $|Q_{\mathcal{RO}}| \leq q_{\mathrm{RO},\mathcal{A}}$.

Note that in the previous definition, $\mathcal{E}$ is a straight-line knowledge extractor because it does not get the prover as input and therefore cannot rewind the prover.

Definition 18 ($(n_1, \cdots, n_\mu)$ Straight-line transcript-tree Extractability). *Let $\Pi = (P, V)$ be a $(n_1, \cdots, n_\mu)$-special sound interactive argument system for a relation R and let Π_{ni} be a non-interactive argument system for the same relation R in the ROM. We say that Π_{ni} is transcript-tree extractable if the polynomial time extractor $\mathcal{E}$ in $ExpKS_A^{(P,V)}(\lambda)$, returns a valid $(n_1, \cdots, n_\mu)$ transcript-tree of accepting transcripts instead of a witness.*

3. Existing SLE-Compilers

In the following we present three of the most common 3-move straight-line extractability compilers from literature. All of them require the input protocol to be special-Honest-Verifier-Zero-Knowledge. After that, we will also introduce the existing multi-round compilers that satisfy straight-line extractability.

For the presentation of the Σ-sle-compilers, we slightly deviate from our standard notation. Instead of denoting a transcript by (a_1, c_1, a_2), we write it as (a, c, z) to improve readability when many indices are involved.

3.1. Pass

The first to introduce an approach making Σ-protocols non-interactive in the Random Oracle Model while guaranteeing straight-line extractability was Pass [Pas03, Sec. 4.2]. His transformation uses a cut-and-choose technique and can be applied to Σ-protocols with n-special-soundness. While his original paper only considered 2-special-soundness, the concept is adaptable to n-special-soundness, which we will prove and use in the following.

For n-special-sound protocols, Pass's compiler first constructs an intermediate interactive cut-and-choose protocol Π (see Figure 3.1). In this protocol, the prover generates the initial message a of the underlying Σ-protocol, uses $m \geq n$ given, distinct challenges $c_1, \ldots, c_m$, and computes the corresponding responses $z_1, \ldots, z_m$. Now the prover has m transcripts for the underlying Σ-protocol

$$\{(a, c^{(j)}, z^{(j)})\}_{j=0}^{m}$$

The prover then sends a, the m challenges $c_1, \ldots, c_m$, and commitments $com_1, \ldots, com_m$ to the m responses $z_1, \cdots, z_m$ to the verifier. It is crucial that the commitment scheme is straight-line extractable and position binding (see [RT25, Def. 9 and 10]). The verifier picks a random index $q \in \{1, \ldots, m\}$ and sends it to the prover. The prover decommits to z_q, and the verifier checks if (a, c_q, z_q) is a valid transcript for the underlying protocol.

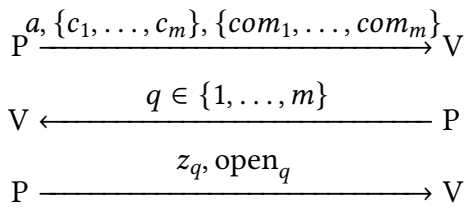


Figure 3.1.: The interactive cut-and-choose protocol Π for n-special-soundness.

A cheating prover without a witness can only construct responses for at most $n-1$ challenges, because if he could construct n, then, by n-special-soundness, he could compute the witness himself, which would contradict the fact that he does not know it. Therefore his chance to convince the verifier in a single execution is the probability that the public-coin verifier picks one of the faked transcripts. Since he fakes at most $n-1$ out of m transcripts, his chance to convince the verifier is $\frac{n-1}{m}$. We demand that $m \in O(n)$. In practice, setting $m = 2n$ constitutes a suitable parameter choice for the Pass compiler, however one has to make sure that the challenge space is large enough ($|C| \geq m$).

To make the protocol non-interactive, Pass's compiler applies the Fiat-Shamir transformation to the interactive protocol Π explained above. Specifically, the prover prepares t independent parallel repetitions. For each repetition $i \in \{1, \ldots, t\}$, the prover generates an initial message a_i, m distinct challenges $c_{i,1}, \ldots, c_{i,m}$, and straight-line extractable, position-binding commitments $com_{i,1}, \ldots, com_{i,m}$ to the corresponding responses $z_{i,1}, \ldots, z_{i,m}$.

The prover then derives the specific indices to be opened by querying the Random Oracle with the statement $\mathbb{x}$ and the full set of committed transcripts:

$$(q_1, \ldots, q_t) = H(\mathbb{x}, \{(a_i, \{(c_{i,j}, com_{i,j})\}_{j=1}^{m})\}_{i=1}^{t})$$

The resulting indices $q_i \in \{1, \ldots, m\}$ determine which response the prover must reveal for each repetition i. The final proof π consists of the initial messages, challenges, commitments, and the openings for the selected responses $\{z_{i,q_i}\}_{i=1}^{t}$.

The verifier accepts if the recomputed hash yields the same indices $(q_1, \ldots, q_t)$, the commitments are correctly opened, and each revealed transcript $(a_i, c_{i,q_i}, z_{i,q_i})$ is valid.

The security of the transformation relies on n-special-soundness: a valid witness can be extracted from any n distinct valid transcripts for the same a_i. Note that the prover must query the full set of committed transcripts to find the relevant challenges which he has to open. Because the compiler requires straight-line-extractable commitments in the

Random Oracle Model, the Extractor can extract each $z_1, \cdots, z_m$ for each commitment $com_1, \cdots, com_m$ with overwhelming probability.

In Section A.1 we prove that the n-special sound variant of the Pass-Transformation has Knowledge-Soundness-Error:

$$\kappa_{Pass} \leq q_{\mathrm{RO},\mathcal{A}} \cdot \left(\frac{n-1}{m}\right)^t + \mathrm{Adv}^{\mathrm{posbind}}_{\mathcal{VC},q_{\mathrm{RO},\mathcal{A}},A}(\lambda) + \mathrm{Adv}^{\mathrm{ext}}_{\mathcal{VC},q_{\mathrm{RO},\mathcal{A}},B}(\lambda)$$

with $\mathrm{Adv}^{\mathrm{posbind}}_{\mathcal{VC},q_{\mathrm{RO},\mathcal{A}},A}(\lambda)$ and $\mathrm{Adv}^{\mathrm{ext}}_{\mathcal{VC},q_{\mathrm{RO},\mathcal{A}},B}(\lambda)$ being the advantages of the commitment scheme.

3.2. Fischlin

Fischlin [Fis05, Sec. 3.1] introduced a communication efficient compiler that relies on a *proof-of-work* mechanism. In this transformation, the prover P first chooses $\vec{a} = (a_1, \cdots, a_t)$ which are the first messages for each repetition of the underlying protocol. Then for each repetition P computes the response z_i to randomly selected challenge c_i until he finds one such that $H(\mathbb{x}, \vec{a}, i, c_i, z_i) = 0^l$. The final proof is

$$\pi = \{(a_i, c_i, z_i)\}_{i=1}^t.$$

The verifier then checks that each transcript (a_i, c_i, z_i) is accepting for the underlying Σ-protocol and that $H(\mathbb{x}, \vec{a}, i, c_i, z_i) = 0^l$ for all i.

In Section A.2 we show that applying the Fischlin compiler to a n-special sound protocol results in a knowledge-soundness-error of:

$$\kappa \leq q_{\mathrm{RO},\mathcal{A}} \cdot \left(\frac{n-1}{2^l}\right)^t$$

where $q_{\mathrm{RO},\mathcal{A}}$ is the amount of Random Oracle Queries an attacker has.

We note that using randomized challenges is crucial, since if the prover always uses a predefined set of challenges in a certain order, there exists an attack on witness-indistinguishability which would break the zero-knowledge property for Sigma-protocols for languages with multiple witnesses, as proven in [KS22, pg. 10].

Another notable issue — this time regarding knowledge soundness — is that one cannot guarantee that an Attacker follows the prover-protocol strictly. An attacking prover can generally choose himself which challenge and transcript he wants to query, and therefore he could potentially check multiple valid transcripts of the form $(a, c, z^{(j)})$

with different $z^{(j)}$. From these transcripts, under the assumption of n-special-soundness, it is not possible to calculate the witness, since the challenges are the same in each transcript. Fischlin's initial solution to the problem was to require the initial protocol to have the property of unique-responses, meaning that it is difficult to find two (or more) transcripts $(a, c, z^{(j)})$ that only differ in $z^{(j)}$. Kondi-Shelat [KS22, pg. 10] argued that this strong property excludes many protocols (like OR compositions) and presented a property called strong-special soundness. It requires that a witness is extractable even from n transcripts which only differ in $(c^{(j)}, z^{(j)})$ meaning that they could have the same challenges. In this thesis we will use the second solution and require the input-protocols to be n-strong-special sound.

3.3. Kondi-Shelat

Kondi and Shelat [KS22, Sec. 5] improved the performance of Fischlin's proof-of-work transformation by using the birthday paradox. Instead of requiring a single transcript with $H(\mathrm{tr}_i) = 0^l$ in each repetition i, their method requires the prover to find multiple transcripts whose hash values collide. Specifically, for a target number r, the prover must find r distinct transcripts that hash to the exact same value, such that $H(\mathrm{tr}_i^{(1)}) = H(\mathrm{tr}_i^{(2)}) = \cdots = H(\mathrm{tr}_i^{(r)})$ for each repetition i.

This fundamentally changes the prover's task from a pre-image search to a multi-collision search, which improves efficiency and makes the transformation more practical than Fischlin's original construction. To construct the full proof, the prover must successfully find these r-way collisions across t parallel repetitions.

In Section A.3 we prove that applying the Kondi-Shelat compiler to a n-special-sound protocol results in a knowledge-soundness-error of:

$$\kappa_{KS} \leq q_{\mathrm{RO},\mathcal{A}} \cdot \left(\binom{n-1}{r} \frac{1}{2^{l(r-1)}} \right)^t$$

Furthermore, the same issue regarding unique responses applies for this transformation as it does for Fischlin's. Because the prover can choose the challenges of the queried transcripts himself, he might find r colliding transcripts by only changing the final response z but using the same challenge. Therefore, the underlying protocol must satisfy strong-special-soundness to ensure the extractor can successfully recover a witness from the colliding transcripts.

3.4. Multi-Round compilers by Rotem-Tessaro

Rotem and Tessaro introduced two straight-line extractability compilers for multi-round protocols in their recent paper [RT25]. We will quickly present them in the following.

The first compiler builds on Fischlin's transformation. As in Fischlin's protocol, the prover must find a transcript t_i for which $H(t_i) = 0^l$. In the multi-round version however, each transcript has the form $(a_1, c_1, \ldots, a_\mu, c_\mu, a_{\mu+1})$. To enforce that the prover creates a transcript tree required for extracting the witness with $(n_1, \ldots, n_\mu)$-special soundness, the compiler restricts the prover to at most k_i challenge choices in round i, where $k_i \geq n_i$.

If the prover's queries form a full $(n_1, \ldots, n_s)$-special-soundness tree, then a straight-line extractor can recover the witness from the Random Oracle queries. If, on the other hand, the prover does not build such a tree, a combinatorial lemma proved by Rotem and Tessaro shows that any tree without an $(n_1, \ldots, n_s)$-complete subtree can have only a limited number of leaves [RT25, Lem 3.6]. This bounds the number of transcripts on which the prover can attempt to satisfy $H(t_i) = 0^l$, and therefore bounds its overall success probability. With appropriate parameter choices, the probability of a prover convincing the verifier without his queries building a full transcript tree becomes negligible.

The second compiler is based on Pass's transformation and uses a given vector commitment scheme VC with certain security properties. In this construction, P generates and commits to $t \in \mathbb{N}$ $(n_1, \ldots, n_\mu)$-transcript-trees $T_i, \ldots, T_t$ using the vector commitment scheme. The prover then queries the Random Oracle on the commitment value $\mathsf{vcom} = \mathsf{VC.Commit}(1^\lambda, (T_1, \ldots T_t))$ together with the input $\mathbb{x}$. The output of this query determines a single path (transcript) for each tree T_i which the prover must open using VC.Open. The final proof π consists of the commitment vcom along with the openings and the corresponding correctness proofs for all elements appearing on the revealed paths. To verify, V checks the correctness of all openings using VC.Verify and then verifies that all the opened transcripts are accepting for the initial statement $\mathbb{x}$. For a prover who commits to at least one $(n_1, \ldots, n_\mu)$-special-soundness tree, the extractor can recover the witness. If however no $(n_1, \ldots, n_\mu)$-special-sound tree is among $T_1, \ldots, T_t$, Rotem and Tessaro prove that the probability of V accepting is negligible. This transformation works for a greater class of proofs than the first one; however, it requires heavier communication between the prover and the verifier.

4. Modular SLE-Compiler

In this chapter, we introduce our modular multi-round straight-line-extractability compiler C, which uses a Σ-sle-compiler S as a black-box building block to transform interactive multi-round protocols into non-interactive ones. We first present the construction of C and then analyze its main properties. In particular, we show that it is efficient and preserves completeness, zero-knowledge, and knowledge soundness.

We emphasize that the compiler C takes as input a Σ-sle-compiler S, which serves as the underlying compiler in our construction. We treat S abstractly via a prover and verifier interface and use this interface directly in the definition of C.

More precisely, for a given Σ-protocol Π^{Σ}, we denote by

$$S_{\Pi^{\Sigma}}.\mathrm{Prove}(\mathbb{x}, \mathbb{w})$$

the non-interactive proof generated by executing the prover of S with Π^{Σ} as it's underlying protocol and $(\mathbb{x}, \mathbb{w})$ as input.

Correspondingly, we denote by

$$S_{\Pi^{\Sigma}}.\mathrm{Verify}(\mathbb{x}, \pi)$$

the verifier algorithm of S with Π^{Σ} as the underlying interactive Σ-protocol, which on input a statement $\mathbb{x}$ and a proof π outputs whether π is a valid non-interactive proof.

In the following we use S as a global variable, so it is globally accessible via its interfaces, and we will not specifically denote it as an input to the functions.

Before presenting the compiler, we establish one piece of notation. Throughout this thesis, the symbol $\|$ denotes **tuple-wise concatenation** rather than bitstring concatenation. Therefore:

$$(11) \,\|\, (1) \neq (1) \,\|\, (11).$$

Now we present the prover and verifier algorithms for our modular multi-round sle-compiler C:

The non-interactive prover $P_{\mathsf{ni}}(\mathbb{x}, \mathbb{w})$

1. Set $h_0 = ()$

2. return $\pi_1 = \textbf{RecProof}(1, h_0, \mathbb{x}, \mathbb{w})$

RecProof($i, h_{i-1}, \mathbb{x}, \mathbb{w}$**)**

1. if $i = \mu + 1$: invoke P to calculate and return $a_{\mu+1}$

2. Else if $1 \leq i \leq \mu$:

(a) set $\mathbb{x}_i = (\mathbb{x}, h_{i-1})$

(b) return $\pi_i \leftarrow S_{\Pi_i^\Sigma}.\text{Prove}(\mathbb{x}_i, \mathbb{w})$

$\Pi_i^\Sigma(\mathbb{x}_i, \mathbb{w})$

1. Compute a_i with P, send it to V

2. V samples random challenge c_i

3. Get $\mathbb{x}$ and h_{i-1} from $\mathbb{x}_i$ and set $h_i = h_{i-1}||(a_i, c_i)$

4. $a_{i+1} = \textbf{RecProof}(i + 1, h_i, \mathbb{x}, \mathbb{w})$

The non-interactive verifier $V_{\mathsf{ni}}(\mathbb{x}, \pi_1)$

1. return **VerifyRecProof**$(1, \mathbb{x}, h_0 = (), \pi_1)$

VerifyRecProof(i, $\mathbb{x}, h_{i-1}, \pi_i$**)**

1. if $i = \mu + 1$: return $V(\mathbb{x}, h_\mu||\pi_{\mu+1}) = V(\mathbb{x}, h_\mu||a_{\mu+1})$

2. Else if $1 \leq i \leq \mu$:

(a) set $\mathbb{x}_i = (\mathbb{x}, h_{i-1})$

(b) run $S_{\Pi_i^\Sigma}.\text{Verify}(\mathbb{x}_i, \pi_i)$

(c) to validate π_{i+1} run **VerifyRecProof**$(i + 1, \mathbb{x}, h_{i-1}||(a_i, c_i), \pi_{i+1})$

Compiler version C^1 In the following proofs, we will use C's recursive structure to show different attributes of C. One frequently used notation is C^1, which means that we only compress the last 3 moves of a $(2\mu + 1)$-move interactive protocol (Π^μ), resulting in a $(2\mu - 1)$-move interactive protocol ($\Pi^{\mu-1}$). We define C^1 as follows:

The prover $P^{C^1}(\mathbb{x}, \mathbb{w})$

1. Use Π^μ to calculate the first $(\mu - 1)$-rounds and set $h_{\mu-1} = (a_1, c_1, a_2, \cdots, c_{\mu-1})$

2. set $\pi_\mu = \textbf{RecProof}(\mu, h_{\mu-1}, \mathbb{x}, \mathbb{w})$

3. return $(a_1, c_1, a_2, \cdots, c_{\mu-1}, \pi_\mu)$

RecProof and $\Pi_i^\Sigma(h_{i-1}, \mathbb{x}, \mathbb{w})$ are unchanged from the original version.

The verifier $V^{C^1}(h_{\mu-1}, \mathbb{x}, \pi_\mu)$

1. return $\textbf{VerifyRecProof}(\mu, \mathbb{x}, h_{\mu-1} = (a_1, c_1, a_2, \cdots, c_{\mu-1}), \pi_\mu)$

VerifyRecProof also stays the same

Notation Before starting our proofs, we clarify some notation which we will use throughout the rest of this thesis. Let Π^μ be the notation for an interactive Protocol with $(2\mu + 1)$-messages. Note that we will call a message a "move" and a "round" the interaction between the prover and verifier. A $(2\mu + 1)$-move protocol therefore only has μ rounds. This is different from common literature, however we believe that our notation is better, since it clearly distinguishes between the two, making it less confusing for the reader.

Let $C(\Pi^\mu)$ denote the output protocol after C has been fully applied to Π^μ, resulting in a non-interactive Protocol $(P_{\text{ni}}, V_{\text{ni}})$. Furthermore, let $C^1(\Pi^\mu) = \Pi^{\mu-1}$ be the output protocol after C^1 has been applied to Π^μ, where $h_{\mu-1}$ was obtained by running the first $(\mu - 1)$ rounds of the interactive Protocol Π^μ. The result is a $(2\mu - 1)$-move interactive protocol. C^1 can be understood as using S to make the Σ-protocol $(a_\mu, c_\mu, a_{\mu+1})$, with input $\mathbb{x}' = (\mathbb{x}||h_{\mu-1})$ non-interactive.

In the definition of our Compiler C, we use the sle-compiler S as a blackbox. However, S needs different inputs as for example the number of parallel repetitions t. This t influences factors like the runtime and the probability for a successful witness-extraction.

Since the $(2\mu + 1)$-move input protocol for C is $(n_1, \cdots, n_\mu)$ special-sound, to ensure a successful witness extraction, the extractor needs to create a $(n_1, \cdots, n_\mu)$-transcript tree. To do that, in each round i and from each π_i, the extractor has to extract n_i transcripts for $\Pi_i^\Sigma(h)$. The larger n_i is in a specific round i, the more parallel repetitions are required to extract at least n_i transcripts. Using the same amount of parallel repetitions in each round would either reduce the security of the protocol (if a small t is chosen) or unnecessarily increase the runtime (if a large t is chosen). To fix this issue, we use a different amount of parallel repetitions in each round depending on the branching factor n_i. We define t_i which gives the amount of parallel repetitions for S in round i, enabling us to tailor the amount of parallel repetitions to n_i.

Moreover, we use the notation Π_i^Σ to denote the intermediate Σ-protocol to which the sle-compiler S is applied in round i. For this protocol, the prover operates on the updated statement $\mathbb{x}_i = \mathbb{x} \parallel h_{i-1}$ and the original witness $\mathbb{w}$, resulting in the 3-message protocol transcript (a_i, c_i, π_{i+1}).

4.1. Completeness

Definition 19 (Iterative Completeness Error Function)**.** *Let Π^{Σ} be an interactive 3-move Σ-protocol with completeness error $\epsilon_{int}(\lambda, s)$ as defined in Definition 5. Let S be a sle-compiler parametrized by the number of parallel repetitions t, the amount of $\mathcal{RO}$ queries a prover has $q_{\mathrm{RO},P}$ and the length of the $\mathcal{RO}$ output σ, which transforms a 3-move protocol Π^{Σ} into a non-interactive protocol Π_{ni} in the Random Oracle Model.*

We define the iterative completeness error function $f_S : [0,1] \to [0,1]$ of the compiler S as the mapping from the interactive completeness error of the input protocol Π^{Σ} to the non-interactive completeness error $\epsilon_{\mathsf{ni}}(\lambda, q_{\mathrm{RO},P}, s, \sigma)$ of $S(\Pi^{\Sigma})$.

Formally:

$$\epsilon_{\mathsf{ni}}(\lambda, q_{\mathrm{RO},P}, s, \sigma) \leq f_S(\epsilon_{int}(\lambda, s))$$

Note: For simplicity in subsequent proofs, we drop the explicit instance and security parameters and denote the error mapping simply as $\epsilon_{\mathsf{ni}} = f_S(\epsilon_{int})$.

Theorem 1 (Completeness)**.** *Let $\Pi^{\mu} = (P^{\mu}, V^{\mu})$ be a $(2\mu+1)$-move interactive protocol with completeness error $\epsilon_{\Pi^{\mu}}$. Let S be a sle-compiler that transforms an interactive 3-move Σ-protocol into a non-interactive protocol with iterative completeness error function $f_S : [0,1] \to [0,1]$.*

If f_S is concave and monotonically increasing in $[0,1]$, then the completeness error of the fully compiled protocol $C(\Pi^{\mu})$ is at most:

$$\epsilon_{C(\Pi^{\mu})} \leq f_S^{\mu}(\epsilon_{\Pi^{\mu}})$$

where $f_S^{\mu}(x) = \underbrace{f_S(f_S(\ldots f_S}_{\mu}(x)\ldots))$. We simplify the notation by not including the round-specific parameters in f.

Before we prove Theorem 1, we show that for a 3-move Σ-protocol Π^1, $C(\Pi^1) = S(\Pi^1)$.

Lemma 2. *Let $\Pi^1 = (P^1, V^1)$ be a 3-move Σ-protocol. Let C be the transformation from Chapter 4, using the underlying sle-compiler S. Assume that for every statement $\mathbb{x}$ and every proof π, the algorithm $S_{\Pi^1}.\mathrm{Verify}(\mathbb{x}, \pi)$ rejects whenever an (opened) transcript within π_1 is not a valid transcript of Π^1.*

Then the protocols $C(\Pi^1)$ and $S(\Pi^1)$ are equivalent, meaning that:

$$P_{C(\Pi^1)}(\mathbb{x}, \mathbb{w}) = P_{S(\Pi^1)}(\mathbb{x}, \mathbb{w}) \quad \textit{and} \quad V_{C(\Pi^1)}(\mathbb{x}, \pi) = V_{S(\Pi^1)}(\mathbb{x}, \pi).$$

In particular, $C(\Pi^1)$ and $S(\Pi^1)$ define the same non-interactive proof system.

Proof. We show separately that the prover and verifier of $C(\Pi^1)$ are equivalent to those of $S(\Pi^1)$.

First consider the prover. By definition, $P_{C(\Pi^1)}$ starts by executing

$$\text{RecProof}(1, (), \mathbb{x}, \mathbb{w}),$$

which in turn invokes $\Pi_1^{\Sigma}((), \mathbb{x}, \mathbb{w})$. Since we consider 3-move protocols and therefore $\mu = 1$, there is no further recursive proof to generate, and the recursive call immediately returns the final Σ-protocol response a_2. Hence the transcript produced inside $\Pi_1^{\Sigma}((), \mathbb{x}, \mathbb{w})$ is exactly the ordinary transcript (a_1, c_1, a_2) generated by the interactive prover $P^1(\mathbb{x}, \mathbb{w})$. Since C uses the prove-interface of S, applying C to Π^1 yields exactly the same proof as applying S directly to Π^1. Thus

$$P_{C(\Pi^1)}(\mathbb{x}, \mathbb{w}) = P_{S(\Pi^1)}(\mathbb{x}, \mathbb{w}).$$

Now consider the verifier. Let π (consisting of transcripts of the form (a_1, c_1, π_2)) be a proof for $C(\Pi^1)$. Again, since $\mu = 1$, we have $\pi_2 = a_2$, so π is simply a transcript (a_1, c_1, a_2) of Π^1. By definition, $V_{C(\Pi^1)}$ accepts π if both of the following hold:

1. the transcript (a_1, c_1, a_2) is valid for Π^1 (check in **1.**)
2. the call to S_{Π^1}.Verify in step (b) accepts π.

And $V_{S(\Pi^1)}$ accepts if the the following holds:

1. S_{Π^1}.Verify accepts π.

Since we assume, that $V_{S(\Pi^1)}$ rejects if any opened transcript is not valid, the check in **1.** is redundant. [1] Therefore, both verifiers are equal and:

$$V_{C(\Pi^1)}(\mathbb{x}, \pi) = 1 \iff V_{S(\Pi^1)}(\mathbb{x}, \pi) = 1.$$

and

$$V_{C(\Pi^1)}(\mathbb{x}, \pi) = 0 \iff V_{S(\Pi^1)}(\mathbb{x}, \pi) = 0$$

Therefore

$$V_{C(\Pi^1)}(\mathbb{x}, \pi) = V_{S(\Pi^1)}(\mathbb{x}, \pi)$$

for all $\mathbb{x}$ and π.

We conclude that $C(\Pi^1)$ and $S(\Pi^1)$ are equivalent. □

[1] If S_{Π^1}.Verify accepts it implies that all opened transcripts are valid and if S_{Π^1}.Verify rejects, $V_{C(\Pi^1)}$ rejects independent from the result of (b)

With the help of Lemma 2 we can now prove Theorem 1.

Proof. We prove the Theorem by induction on μ, the number of recursive layers.

Base Case ($\mu = 1$): Let $\Pi^1 = (P^1, V^1)$ be a 3-move Σ-protocol. By the definition of the compiler and as proven in Lemma 2, applying C to a 3-move protocol is equivalent to applying S to it.

Therefore, $C(\Pi^1) = S(\Pi^1)$. By the definition of f_S, the completeness error is:

$$\epsilon_{C(\Pi^1)} = \epsilon_{S(\Pi^1)} \leq f_S(\epsilon_{\Pi^1}) = f_S^1(\epsilon_{\Pi^1})$$

The base case holds.

Induction Hypothesis (IH): Assume that for any $\mu \in \mathbb{N}$ and any $(2\mu + 1)$-move protocol Π^μ, $\epsilon_{C(\Pi^\mu)} \leq f_S^\mu(\epsilon_{\Pi^\mu})$ holds.

Induction Step ($\mu \longrightarrow \mu + 1$): We must show that for a $(2(\mu + 1) + 1)$-move protocol $\Pi^{\mu+1} = (P^{\mu+1}, V^{\mu+1})$, the error satisfies $\epsilon_{C(\Pi^{\mu+1})} \leq f_S^{\mu+1}(\epsilon_{\Pi^{\mu+1}})$.

Since the success of the protocol depends on the specific transcript history, we model the completeness error of $\Pi^{\mu+1}$ as a random variable depending on the partial transcript $h = (a_1, c_1, \ldots, a_\mu, c_\mu)$ which represents the first 2μ moves of the protocol.

We define the conditional completeness error $e(h)$ as the probability that the honest Verifier $V^{\mu+1}$ rejects the final, honestly generated transcript by $P^{\mu+1}$, conditioned on the fixed first 2μ messages h. Consequently, the completeness error of $\Pi^{\mu+1}$ is the expected value over all possible histories:

$$\epsilon_{\Pi^{\mu+1}} = \mathbb{E}[e(H)]$$

with H being the distribution of partial transcripts h generated by the interaction of the honest prover P and the honest verifier V.

Alternatively, we can view the execution of $\Pi^{\mu+1}$ as a two-phase process: first, the prover and verifier interactively generate the prefix history h; second, they execute the final 3-message exchange. We can treat this final exchange as a standalone Σ-protocol parameterized by the history, denoted as $\Pi_{\mu+1}^{\Sigma}(h)$. By definition, the completeness error of this final Σ-protocol is exactly the conditional error $e(h)$.

When one applies S to compress this Σ-protocol, it transforms the interactive exchange $\Pi_{\mu+1}^{\Sigma}(h)$ into a non-interactive proof. According to our definition of the iterative completeness error function, the completeness error for this compiled protocol—still conditioned on the specific history h—is mapped to $f_S(e(h))$.

In connection with the interactive first part, this single step of compilation effectively reduces the original $(2(\mu + 1) + 1)$-move protocol down to a $(2\mu + 1)$-move protocol, which we will denote as Π^{μ}. To find the overall completeness error of this newly reduced protocol Π^{μ}, we take the expected value of these transformed conditional errors over the distribution of all possible interactive histories H:

$$\epsilon_{\Pi^{\mu}} = \mathbb{E}[f_S(e(H))]$$

Note that splitting the protocol into two parts and then making the latter non-interactive is equal to applying C^1 to $\Pi^{\mu+1}$. Therefore, we know that

$$\epsilon_{C^1(\Pi^{\mu+1})} = \epsilon_{\Pi^{\mu}} = \mathbb{E}[f_S(e(H))]$$

Since f is concave per definition, we can apply Jensen's inequality to obtain the relation:

$$\epsilon_{\Pi^{\mu}} = \mathbb{E}[f_S(e(H))] \leq f_S(\mathbb{E}[e(H)]) = f_S(\epsilon_{\Pi^{\mu+1}})$$

By the recursive definition of the compiler, $C(\Pi^{\mu+1})$ is equivalent to first calculating $C^1(\Pi^{\mu+1})$ to obtain a reduced protocol Π^{μ}, and then fully compiling the resulting $(2\mu + 1)$-move protocol Π^{μ}.

Since Π^{μ} is a $(2\mu + 1)$-move protocol we can apply the Induction Hypothesis:

$$\epsilon_{C(\Pi^{\mu})} \leq f_S^{\mu}(\epsilon_{\Pi^{\mu}})$$

Substituting $\epsilon_{\Pi^{\mu}} \leq f_S(\epsilon_{\Pi^{\mu+1}})$ into the inequality:

$$\epsilon_{C(\Pi^{\mu+1})} = \epsilon_{C(C^1(\Pi^{\mu+1}))} \leq f_S^{\mu}(\epsilon_{\Pi^{\mu}}) \leq f_S^{\mu}(f_S(\epsilon_{\Pi^{\mu+1}})) = f_S^{\mu+1}(\epsilon_{\Pi^{\mu+1}})$$

The second $\leq$ can be done because f is monotonically increasing in [0,1].

□

4.1.1. Fischlin

In the following we will present the iterative completeness error functions f for the three 3-move sle-compilers presented in Chapter 3.

Theorem 3 (Iterative Completeness-Function for Fischlin). *The iterative Completeness function for Fischlin is*

$$f_{\epsilon_S(q_{\mathrm{RO},P}),t}(\epsilon_{\Pi^1}) = 1 - \left[(1-\epsilon_{\Pi^1})^t \cdot (1-\epsilon_S(q_{\mathrm{RO},P}))^t\right] = \epsilon_{\Pi_{\mathrm{ni}}}$$

with t being the amount of parallel repetitions for S and $\epsilon_S(q_{\mathrm{RO},P})$ being defined as the completeness error of $S(\Pi^1)$ if Π^1 is perfectly complete with $t = 1$ and $q_{\mathrm{RO},P}$ being the maximum amount of $\mathcal{RO}$ queries an honest prover can make.

Proof. We have to show that on input in $[0,1]$, the output of f is also in $[0,1]$, the monotonicity of f and finally that the output is actually the completeness error of the compiled protocol. To simplify the notation we only write ϵ_S instead of $\epsilon_S(q_{\mathrm{RO},P})$ even though ϵ_S depends on the number of queries an honest prover can make and we just write f without the specific indices.

1. Range: For $\epsilon_S \in [0,1]$ and $\epsilon_{\Pi^1} \in [0,1]$, it holds that $f_{\epsilon_S,t}(\epsilon_{\Pi^1}) \in [0,1]$.

   $\Rightarrow$ We show: $0 \leq f(\epsilon_\Pi) \leq 1$

   1. $0 \leq 1 - \left[(1-\epsilon_{\Pi^1})^t \cdot (1-\epsilon_S)^t\right]$ (if $t \geq 1$)

   $\iff (1-\epsilon_{\Pi^1})^t \cdot (1-\epsilon_S)^t \leq 1$

   $\iff (1-\epsilon_{\Pi^1})^t \cdot (1-\epsilon_S)^t \leq 1^t \cdot 1^t = 1$

   2. $1 \geq 1 - \left[(1-\epsilon_{\Pi^1})^t \cdot (1-\epsilon_S)^t\right]$

   $\iff \left[(1-\epsilon_{\Pi^1})^t \cdot (1-\epsilon_S)^t\right] \geq 0$

   $\iff \left[(1-\epsilon_{\Pi^1})^t \cdot (1-\epsilon_S)^t\right] \geq (1-1)^t \cdot (1-1)^t = 0$

   In the following we use the constant $C = (1-\epsilon_S)^t$, where $\epsilon_S \in [0,1]$ and $t \geq 1$. Note that since $0 \leq \epsilon_S \leq 1$, it follows that $0 \leq C \leq 1$.

   With that we can define the iterative completeness error function for Fischlin as:

   $$f(x) = 1 - C(1-x)^t \tag{4.1}$$

2. Monotonicity: To prove that $f(x)$ is monotonically increasing on the interval $[0,1]$, we show that the first derivative $f'(x) \geq 0$.

Differentiating $f(x)$ with respect to x with $t > 1$:

$$\begin{aligned} f'(x) &= \frac{d}{dx}\left(1 - C(1-x)^t\right) \\ &= -C \cdot \frac{d}{dx}(1-x)^t \\ &= -C \cdot t(1-x)^{t-1} \cdot (-1) \quad \text{(Chain Rule)} \\ &= C \cdot t \cdot (1-x)^{t-1} \end{aligned}$$

for $t = 1$:

$$\begin{aligned} f'(x) &= \frac{d}{dx}\left(1 - C(1-x)\right) \\ &= -C \cdot \frac{d}{dx}(1-x) \\ &= -C \cdot (-1) = C \geq 0 \end{aligned}$$

Since $C \geq 0$, $t \geq 1$ and $0 \leq (1 - \epsilon_{\Pi^1}) = (1-x) \leq 1$, $f'(x) \geq 0$.

3. Concavity: To prove that $f(x)$ is concave on the interval $[0, 1]$, we show that the second derivative $f''(x) \leq 0$.

   Differentiating $f'(x)$ with respect to x:

   for $t > 1$:

$$\begin{aligned} f''(x) &= \frac{d}{dx}\left(C \cdot t \cdot (1-x)^{t-1}\right) \\ &= C \cdot t \cdot \frac{d}{dx}(1-x)^{t-1} \\ &= C \cdot t \cdot (t-1)(1-x)^{t-2} \cdot (-1) \quad \text{(Chain Rule)} \\ &= -C \cdot t \cdot (t-1) \cdot (1-x)^{t-2} \end{aligned}$$

   for $t = 1$:

$$f''(x) = \frac{d}{dx} C = 0$$

   Since $C \geq 0$, $t \geq 1$, $(t-1) \geq 0$, and $0 \leq (1 - \epsilon_{\Pi^1}) = (1-x) \leq 1$, and the leading term is negative (-1), $f''(x) \leq 0$ and f is concave.

4. **Claim:** In the following we prove that the presented $f_{\epsilon_s,t}$ is in fact the correct iterative completeness error function for Fischlin's compiler. More specifically we prove: $f_{\epsilon_S,t}(\epsilon_{\Pi^1}) = \epsilon_{\Pi_{\mathsf{ni}}}$ with $\Pi_{\mathsf{ni}} = S(\Pi^1)$ and:

$$f_{\epsilon_S,t}(\epsilon_{\Pi^1}) = 1 - \left[(1-\epsilon_{\Pi^1})^t \cdot (1-\epsilon_S)^t\right]$$

We show that $f_{\epsilon_S,t}(\epsilon_{\Pi^1}) = \epsilon_{S(\Pi^1)}$. To do that we note that the probability that an honest prover can generate a valid transcript which the verifier accepts depends on the first message $\mathbf{a_1} = (a_1^1, a_1^2, \cdots, a_1^t)$ he commits to. Let $\mathbf{A}$ be a random variable representing the first message sent by an honest prover $P(\mathbb{x}, \mathbb{w})$. We define the conditional success probability $\rho(\mathbf{a_1})$ as the probability that the verifier accepts given the first message $\mathbf{a_1}$:

$$\rho(\mathbf{a_1}) = \Pr[V \text{ accepts } \mid \mathbf{A} = \mathbf{a_1}] \tag{4.2}$$

The success probability of the interactive Σ-protocol is the expected value of this conditional probability over the distribution of $\mathbf{A}$:

$$1 - \epsilon_{\Pi^1} = \mathbb{E}[\rho(\mathbf{A})] \tag{4.3}$$

Now we analyze how the completeness error changes after applying S to Π^1. For $V_{S(\Pi^1)}$ to accept a proof, the proof-of-work condition has to be satisfied (which happens with probability $(1-\epsilon_S)$), and the protocol Π^1 must succeed in each of the t repetitions. We define $Success_i$ as the event that the i'th repetition is accepted. Conditioned on the choice of a_1^i, the probability of success is:

$$\Pr[Success_i \mid a_1^i] = (1-\epsilon_S) \cdot \rho(a_1^i)$$

with $\rho(a_1^i) = \Pr[V \text{ accepts in repetition i } \mid \mathbf{A} = \mathbf{a_1}]$.

The total success probability of $S(\Pi^1) = C(\Pi^1)$ (see Lemma 2) is the probability that all t repetitions succeed. We calculate that with the expected value over all possible first messages a_1.

$$\begin{aligned} P_{success} &= \mathbb{E}_{\mathbf{A}}\left[\Pr[\text{All } t \text{ repetitions succeed} \mid \mathbf{A}]\right] \\ &= \mathbb{E}_{\mathbf{A}}\left[\prod_{i=1}^{t} \Pr[Success_i \mid a_1^i]\right] \\ &= \mathbb{E}_{\mathbf{A}}\left[\prod_{i=1}^{t} ((1-\epsilon_S)\cdot\rho(a_1^i))\right] = (1-\epsilon_S)^t \cdot \mathbb{E}_{\mathbf{A}}\left[\prod_{i=1}^{t}\rho(a_1^i)\right] \end{aligned}$$

Since each a_1^i is generated independently, all $\rho(a_1^i)$ are independent from each other. Therefore the expected value of the product is equal to the product of the expected values:

$$\begin{aligned} P_{success} &= (1-\epsilon_S)^t \cdot \prod_{i=1}^{t} \left[\mathbb{E}_{\mathbf{A}}\left[\rho(a_1^i)\right]\right] \\ &= (1-\epsilon_S)^t \cdot \prod_{i=1}^{t} (1-\epsilon_{\Pi^1}) = (1-\epsilon_S)^t \cdot (1-\epsilon_{\Pi^1})^t \end{aligned}$$

Since that gives us the probability for Success, we can calculate the completeness-error for $S(\Pi^1)$ with

$$1 - P_{success} = 1 - \left[(1-\epsilon_\Pi)^t \cdot (1-\epsilon_S)^t\right]$$

□

4.1.2. Pass

Theorem 4 (Completeness-function for Pass)**.** *The iterative Completeness function for Pass is*

$$f_t(\epsilon_{\Pi^1}) = 1 - \left[(1-\epsilon_{\Pi^1})^t\right] = \epsilon_{\Pi_{\mathsf{ni}}}$$

with t being the amount of parallel repetitions of S.

Proof. As the only difference between f and the function in Section 4.1.1 is the absence of the $(1-\epsilon_S)^t$ term, the proofs for range, monotonicity, and concavity remain valid. This holds because $0 \leq (1-\epsilon_{\Pi^1})^t \leq 1$ and $(1-\epsilon_S)^t = C = 1$.

In the following we show that the proposed function f_t is in fact the correct iterative completeness function for the Pass compiler:

We prove: $f_t(\epsilon_{\Pi^1}) = \epsilon_{\Pi_{\mathsf{ni}}}$ with $\Pi_{\mathsf{ni}} = C^1(\Pi^1)$ and:

$$f_t(\epsilon_{\Pi^1}) = 1 - \left[(1-\epsilon_{\Pi^1})^t\right]$$

To prove this, we again refer to Section 4.1.1, since the proof is very similar to what we did there. We only argue how $Success_i$ looks like, and the rest of the proof can be done in a similar way as in Section 4.1.1. Since, the Pass compiler is perfectly complete, the compilation of the protocol does not add any completeness error, therefore $1-\epsilon_S = 1$, and for $V_{S(\Pi^1)}$ to accept a valid proof, we just require P^1 to succeed in all

t_1 repetitions. The probability that the verifier accepts the i'th repetition therefore is: $\Pr[Success_i \mid a_1^i] = \rho(a_1^i)$.

The rest of the proof is similar to Section 4.1.1.

□

4.1.3. Kondi-Shelat

Theorem 5 (Completeness-function for Kondi-Shelat)**.** *The iterative Completeness function for Kondi-Shelat is*

$$f_{r,t}(\epsilon_{\Pi^1}) = 1 - [(1 - \epsilon_{\Pi^1})^{r \cdot t}] = \epsilon_{\Pi_{\mathsf{ni}}}$$

with t being the amount of parallel repetitions and r being the amount of transcripts with equal hash values the prover has to find.

Proof. Since the only difference of f compared to Section 4.1.2 is the different exponent ($r \cdot t$ instead of t) of $(1 - \epsilon_{\Pi^1})$, we we can use the proofs for the monotonicity, concavity and range from there.

Now we only need to prove the claim:

$f_{r,t}(\epsilon_{\Pi^1}) = \epsilon_{\Pi_{\mathsf{ni}}}$ with $\Pi_{\mathsf{ni}} = C^1(\Pi^1)$ and:

$$f_{r,t}(\epsilon_{\Pi^1}) = 1 - \left[(1 - \epsilon_{\Pi^1})^{r \cdot t}\right]$$

Similar to Section 4.1.2 we introduce and justify a new definition for $Pr[Success_i | a_1^i]$. For $V_{S(\Pi^1)}$ to accept the Kondi-Shelat style proof, in each of the t repetitions, the verifier has to successfully validate r transcripts. Since the Kondi-Shelat compiler is perfectly complete by construction, the prover will find r transcripts in each repetition where the hash-values match. Hence, $1 - \epsilon_S = 1$ and we therefore only need to focus on the completeness-error induced by the underlying protocol.

Hence: $\Pr[Success_i \mid a_1^i] = \rho(a_1^i)^r$. □

4.2. Efficiency

In this section, we establish upper bounds for the runtime and communication complexity of the modular compiler $C(\Pi^\mu)$, as introduced in Chapter 4. Later, in Section 4.5, we will leverage these bounds to analyze the parameter choices required to guarantee both polynomial runtime and polynomial communication efficiency.

4.2.1. Runtime

We assume the underlying protocols are public-coin. In such protocols, the interactive Verifier's role (during the calculation of the transcript) is limited to sampling uniformly random challenges c_i, an operation of negligible time compared to the Prover's tasks. Consequently, we assume that calculating a transcript non-interactively (by internally simulating the Verifier) is roughly equivalent in runtime to the Prover's execution in the interactive setting.

The following Theorem establishes efficiency bounds applicable to both the Prover and the Verifier. For the sake of generality, we use the notation T_{Π^μ} to represent the runtime of either the prover or the verifier within the protocol Π^μ. In the case of the prover T_{Π^μ} is defined as the runtime of the interactive prover to calculate a full transcript for Π^μ. In the verifiers case, T_{Π^μ} is the time it takes an interactive verifier to verify a transcript of Π^μ. The following proof applies independently and identically to both the Prover's and the Verifier's runtime logic.

Definition 20 (Iterative Runtime Function)**.** *Let Π^μ be an interactive $(2\mu+1)$-move protocol with Prover(resp. Verifier) runtime T_{Π^μ} and let S be a sle-compiler transforming a Σ-protocol into a non-interactive protocol.*

S has an Iterative Runtime Function $g_S : \mathbb{N} \to \mathbb{N}$ which takes T_{Π^μ} as input and maps it to the Prover(resp. Verifier) runtime $T_{\Pi_{\mu-1}}$ of the output protocol $C^1(\Pi^\mu)$.

Formally:

$$T_{\Pi_{\mu-1}} \leq g_S(T_{\Pi^\mu})$$

Note that $T_{\Pi_{\mu-1}}$ includes the time to interactively calculate the history h as well as the time for the calculation of the proof π_μ which replaces the last 3 moves. Furthermore we want to clarify, that even though they look quite similar there is a significant difference between the iterative completeness function defined in Section 4.1 and the iterative runtime function defined here. While the function for completeness is only defined for Σ-protocols, we define the runtime function for $(2\mu+1)$-move protocols of which the last 3 moves get compressed.

Theorem 6 (Runtime). *Let $\Pi^\mu = (P^\mu, V^\mu)$ be a $(2\mu+1)$-move interactive protocol. Let S be a sle-compiler that transforms an interactive 3-move protocol into a non-interactive one with iterative runtime function $\mathrm{g_S} : \mathbb{N} \to \mathbb{N}$.*

If $\mathrm{g_S}$ describes the runtime of the resulting protocol after one application of C^1 , then the runtime of the Prover (resp. Verifier) of the fully compiled protocol $C(\Pi^\mu)$ is determined by the recursive application of $\mathrm{g_S}$:

$$T_{C(\Pi^\mu)} \leq \mathrm{g_S}^\mu(T_{\Pi^\mu})$$

where $\mathrm{g_S}^\mu(x) = \underbrace{\mathrm{g_S}(\mathrm{g_S}(\ldots \mathrm{g_S}}_{\mu}(x)\ldots))$ denotes the function applied μ times.

Proof. We prove the theorem in a very similar manner to Theorem 1 by induction over μ, the number of recursive layers.

Base Case ($\mu = 1$): Let $\Pi^1 = (P^1, V^1)$ be a 3-move Σ-protocol. By the definition of the compiler, applying C to a 3-move protocol is equivalent to a single application of S (as proven in Lemma 2). Since $C(\Pi^1)$ and $S(\Pi^1)$ perform identical operations for a 3-move protocol, their runtimes equal. By the definition of $\mathrm{g_S}$, the Prover(resp. Verifier) runtime of the resulting protocol after a single application of S on Π^1 is $\mathrm{g_S}(T_{\Pi^1})$. Thus:

$$T_{C(\Pi^1)} = T_{S(\Pi^1)} \leq \mathrm{g_S}(T_{\Pi^1}) = \mathrm{g_S}^1(T_{\Pi^1})$$

The base case holds.

Induction Hypothesis (IH): Assume that for any $\mu \in \mathbb{N}$ and any $(2\mu+1)$-move protocol Π^μ, the Prover(resp. Verifier) runtime of the fully compiled, non-interactive protocol $C(\Pi^\mu)$ satisfies:

$$T_{C(\Pi^\mu)} \leq \mathrm{g_S}^\mu(T_{\Pi^\mu})$$

Induction Step ($\mu \to \mu+1$): We must show that for any $(2(\mu+1)+1)$-move protocol $\Pi^{\mu+1}$, the runtime satisfies $T_{C(\Pi^{\mu+1})} \leq \mathrm{g_S}^{\mu+1}(T_{\Pi^{\mu+1}})$.

By the recursive definition of the compiler C, fully compiling $\Pi^{\mu+1}$ is equivalent to first calculating $C^1(\Pi^{\mu+1})$ to obtain a reduced protocol Π^μ, and then fully compiling the resulting $(2\mu+1)$-move protocol Π^μ.

First, we look at the transformation of the $(2(\mu+1)+1)$-move protocol into the $(2\mu+1)$-move protocol.

Since the iterative runtime function of S is specifically defined to calculate the runtime of the resulting protocol $C^1(\Pi^{\mu+1}) = \Pi^\mu$ after applying S to the last 3 moves of $\Pi^{\mu+1}$, we can apply $\mathrm{g_S}$ to calculate the runtime of Π^μ.

$$T_{C^1(\Pi^{\mu+1})} \leq \mathrm{g_S}(T_{\Pi^{\mu+1}}) = T_{\Pi^\mu}$$

Now, we use C to compile Π^μ. Note that Π^μ is effectively a $(2\mu+1)$-move protocol (where the last message is a proof $\pi_{\mu+1}$). Therefore, we can apply the Induction Hypothesis:

$$T_{C(\Pi^\mu)} \leq \mathrm{g}_{\mathrm{S}}^{\mu}(T_{\Pi^\mu})$$

Substituting $T_{\Pi^\mu} \leq \mathrm{g}_{\mathrm{S}}(T_{\Pi^{\mu+1}})$ into the equation:

$$T_{C(\Pi^{\mu+1})} = T_{C(\Pi^\mu)} \leq \mathrm{g}_{\mathrm{S}}^{\mu}(T_{\Pi^\mu}) = \mathrm{g}_{\mathrm{S}}^{\mu}(\mathrm{g}_{\mathrm{S}}(T_{\Pi^{\mu+1}})) = \mathrm{g}_{\mathrm{S}}^{\mu+1}(T_{\Pi^{\mu+1}})$$

Thus, the efficiency theorem holds for all μ. □

4.2.2. Iterative Prover Runtime Functions

In the following we will present the iterative prover runtime-functions $\mathrm{g}^{\mathrm{P}}(T_{\Pi^\mu})$ for each 3-move sle-compiler presented in Chapter 3. As a reminder, here T_{Π^μ} stands for the time, a prover needs to calculate a full transcript for Π^μ.

4.2.2.1. Pass

Lemma 7. *For Pass's Compiler, a $(n_1, \cdots, n_\mu)$-special sound interactive Protocol Π^μ and $m_i \geq n_i$ for all $i \in \{1, \cdots, \mu\}$:*

$$\mathrm{g}_{\mathrm{Pass}}^{\mathrm{P}}(T_{\Pi^\mu}) = t_\mu \cdot m_\mu \cdot T_{\Pi^\mu} \geq T_{\Pi^{\mu-1}}$$

Proof. The Pass compiler uses a "cut-and-choose" technique which requires the generation of exactly m_μ distinct transcripts that diverge after a_μ in each of the t_μ repetitions. We use T_{Π^μ} as an upper bound on the calculation-time for each transcript, and since the prover needs to calculate $m_\mu \cdot t_\mu$ transcripts in total, $\mathrm{g}_{\mathrm{Pass}}^{\mathrm{P}}$ is a valid upper bound for the runtime of the resulting $(2\mu - 1)$ move protocol. □

4.2.2.2. Fischlin

Lemma 8. *For the Fischlin Compiler we argue that*

$$\mathrm{g}_{\mathrm{Fischlin}}^{\mathrm{P}}(T_{\Pi^\mu}) = q_{\mathrm{RO},P} \cdot T_{\Pi^\mu} \geq T_{\Pi^{\mu-1}}$$

where we define $q_{\mathrm{RO},P}$ as the maximum amount of random-oracle queries a prover can make to get a proof.

Proof. In Fischlin's transformation, the prover must find a valid transcript for the protocol Π_μ^Σ for t_μ parallel repetitions that satisfy a proof-of-work condition (a specific number of leading zeros in the hash).

Let $q_{\mathrm{RO},P}$ be defined as the maximum number of random oracle (hash) queries the prover is permitted to make across all t_μ repetitions combined to successfully generate the proof π_μ. The actual runtime of the prover involves calculating the initial messages $(a_1, c_1, a_2, \cdots, c_{\mu-1})$, computing the final Σ-transcripts $(a_\mu, c_\mu, a_{\mu+1})$ and checking these for the proof of work condition. Overall the prover can check up to $q_{\mathrm{RO},P}$ Σ-transcripts to find t_μ successful instances.

We upper-bound the time to calculate one final Σ-transcript by the time it takes to calculate the full transcript for Π^μ. Therefore each query to the random oracle takes T_{Π^μ} and since the prover can do up to $q_{\mathrm{RO},P}$ queries, $\mathrm{g}^{\mathrm{P}}_{\mathrm{Fischlin}}$ is valid upper bound for $T_{\Pi^{\mu+1}}$. □

4.2.2.3. Kondi-Shelat

Lemma 9. *For Kondi-Shelat's Compiler*

$$\mathrm{g}^{\mathrm{P}}_{\mathrm{Kondi-Shelat}}(T_{\Pi^\mu}) = q_{\mathrm{RO},P} \cdot T_{\Pi^\mu} \geq T_{\Pi^{\mu-1}}$$

Proof. In Kondi-Shelat's transformation, the prover must find valid transcripts for Π_μ^Σ for t_μ parallel repetitions that satisfy a collision condition (finding r_μ transcripts that share the same hash value).

Let $q_{\mathrm{RO},P}$ be defined as the global maximum number of random oracle (hash) queries the prover is permitted to make across all t_μ repetitions combined to successfully generate the proof π_μ. The actual runtime of the prover involves calculating the initial messages $(a_1, c_1, a_2, \cdots, c_{\mu-1})$, computing the Σ-transcripts $(a_\mu, c_\mu, a_{\mu+1})$ and checking these for the collision condition. Overall the prover can check $q_{\mathrm{RO},P}$ final responses to find the t_μ successful instances (where each instance requires r_μ colliding transcripts).

We upper-bound the time to calculate one Σ-transcript by the time it takes to calculate the full transcript for Π^μ. Therefore each query to the random oracle takes T_{Π^μ} and multiplying that by the amount of possible queries results in our above described iterative prover runtime function. □

4.2.3. Iterative Verifier Runtime Functions

In the following we present the iterative verifier runtime functions $g^{V}(T_{\Pi^{\mu}})$. As a reminder, in the following section, $T_{\Pi^{\mu}}$ is considered to be the time a verifier needs to verify a transcript of Π^{μ}. We use T_S to denote the time required to verify the conditions of the sle-compiler S, such as checking the hash value or opening a commitment. T_S is negligible compared to $T_{\Pi^{\mu}}$, however we will still include it in our analysis since it makes it more exact.

4.2.3.1. Pass

Lemma 10. *For Pass, the iterative verifier runtime function is:*

$$g^{V}_{\text{Pass}}(T_{\Pi^{\mu}}) = (T_{\Pi^{\mu}} + T_S) \cdot t_{\mu}$$

Proof. The verifier must check t_{μ} parallel repetitions of the protocol. For each repetition, the verifier validates the opened transcript as well as the claim that the right transcript was opened. Thus, the verification involves the base time $T_{\Pi^{\mu}}$ to check the validity of the opended transcript, plus the overhead T_S for verifying the compiler conditions, all repeated t_{μ} times. □

4.2.3.2. Fischlin

Lemma 11. *For the Fischlin Compiler, the iterative verifier runtime function is*

$$g^{V}_{\text{Fischlin}}(T_{\Pi^{\mu}}) = (T_{\Pi^{\mu}} + T_S) \cdot t_{\mu}$$

Proof. The verification process in Fischlin's transformation requires checking t_{μ} parallel repetitions. For each of the t_{μ} transcripts, the verifier must perform two checks: first, verifying the underlying proof transcript using the original verifier logic ($T_{\Pi^{\mu}}$), and second, checking the specific hash condition imposed by the compiler (T_S), such as ensuring the hash of the transcript falls below the target value. □

4.2.3.3. Kondi-Shelat

Lemma 12. *For Kondi-Shelats compiler we argue that*

$$g^{V}_{\text{Kondi-Shelat}}(T_{\Pi^\mu}) = (r_\mu \cdot T_{\Pi^\mu} + T_S) \cdot t_\mu$$

Proof. In the Kondi-Shelat construction, the verifier also has to check t_μ parallel repetitions. For each of the t_μ repetitions, the verifier verifies r_μ transcripts and checks if they share a common hash value. Therefore, the base verification work T_{Π^μ} is performed roughly r_μ times per repetition, in addition to the compiler-specific overhead T_S for checking hash collisions. □

4.2.4. Communication Complexity

We define the communication complexity of the interactive Protocol Π^μ as the sum of the communication complexity of all messages sent by the prover and the verifier. Note that since C transforms an interactive Protocol into a non-interactive one, when analysing the communication complexity for the fully compiled, non-interactive protocol, we only have to consider the one message sent from the prover to the verifier.

Definition 21 (Iterative Communication Complexity Function)**.** *Let Π^1 be a 3-move Σ-protocol with communication complexity W_{Π^1} and let S be a sle-compiler transforming a Σ-protocol into a non-interactive protocol.*

S has an Iterative Communication Complexity Function $g_{\text{com}} : \mathbb{N} \to \mathbb{N}$ which takes the communication complexity of the Σ-protocol W_{Π^1} as input and maps it to the communication complexity $W_{\Pi_{\text{ni}}}$ of the non-interactive protocol $S(\Pi^1) = \Pi_{\text{ni}}$.

Formally:

$$W_{\Pi_{\text{ni}}} = g_{\text{com}}(W_{\Pi^1})$$

Theorem 13 (Communication Complexity)**.** *Let $\Pi^\mu = (P^\mu, V^\mu)$ be a $(2\mu + 1)$-move interactive protocol with messages $(a_1, c_1, a_2 \ldots, c_\mu, a_{\mu+1})$. Let S be a sle-compiler with an iterative communication complexity function $g^{(i)}_{\text{com}}$ at round i (parameterized by round-specific variables like t_i).*

The total communication complexity $W_{C(\Pi^\mu)}$ of the fully compiled protocol is determined by recursively applying the iterative communication complexity function from the innermost recursion round (μ) up to the outermost round (1).

Specifically, if we define the communication size of $W_{\pi_{\mu+1}} = W_{a_{\mu+1}}$, then for each round $i \in \{1, \cdots, \mu\}$, the size of the compressed proof at that round is:

$$W_{\pi_i} = \mathrm{g}^{(\mathrm{i})}_{\mathrm{com}}(W_{a_i} + W_{c_i} + W_{\pi_{i+1}})$$

The total communication complexity is the size of the final, outermost proof:

$$W_{C(\Pi^\mu)} = W_{\pi_1}$$

Proof. We formalize the communication complexity by mirroring the recursive execution of the modular compiler C.

When applying C to a $(2\mu + 1)$-move protocol Π^μ, the compiler recursively reduces the protocol by compressing the last three moves at each step.

Base Case (The Innermost Round μ): Let Π^μ be a $(2\mu + 1)$-move interactive protocol. When applying C, the recursion starts by compressing the innermost layer. In the innermost layer of the recursion, the compiler isolates the final three messages of the interactive protocol: $(a_\mu, c_\mu, a_{\mu+1})$. This forms the 3-move Σ-protocol Π^Σ_μ conditioned on the prefix h (as described in Chapter 4). The total communication complexity of this Σ-protocol is the sum of its individual parts: $W_{a_\mu} + W_{c_\mu} + W_{a_{\mu+1}}$.

By the definition of the iterative communication complexity function, applying the sle-compiler S to these three moves yields a non-interactive proof π_μ of size:

$$W_{\pi_\mu} = \mathrm{g}^{(\mu)}_{\mathrm{com}}(W_{a_\mu} + W_{c_\mu} + W_{a_{\mu+1}})$$

To generalize the notation for the recursive sequence, we define $W_{\pi_{\mu+1}} = W_{a_{\mu+1}}$, allowing us to write this as $W_{\pi_\mu} = \mathrm{g}^{(\mu)}_{\mathrm{com}}(W_{a_\mu} + W_{c_\mu} + W_{\pi_{\mu+1}})$.

Recursive Step (Rounds $i = \mu - 1$ down to 1): Let Π^i be an interactive protocol where the last message is already a proof for a certain amount of compressed moves. At any intermediate round i, the previously compiled proof π_{i+1} acts as the final prover response for the newly reduced 3-move protocol $\Pi^\Sigma_i\ (a_i, c_i, \pi_{i+1})$. The total communication complexity of this intermediate 3-move protocol is the size of the initial message, the challenge, and the nested proof combined: $W_{a_i} + W_{c_i} + W_{\pi_{i+1}}$.

Applying S to compress this specific round yields a new proof π_i. Using the round-specific iterative function $\mathrm{g}^{(\mathrm{i})}_{\mathrm{com}}$, its size is exactly:

$$W_{\pi_i} = \mathrm{g}^{(\mathrm{i})}_{\mathrm{com}}(W_{a_i} + W_{c_i} + W_{\pi_{i+1}})$$

When the recursion reaches $i = 1$, after application of S, the protocol is fully compiled into a non-interactive one and the only message sent from the prover to the verifier is the final outermost proof π_1. Therefore, the total communication complexity $W_{C(\Pi^\mu)}$ is exactly W_{π_1}. □

4.2.4.1. Iterative Communication Function for Pass

Lemma 14 (Communication for Pass)**.** *The iterative communication complexity function for Pass applied to a n_1-special-sound protocol (with $m_1 \geq n_1$) is:*

$$\mathrm{g}_{\mathrm{com}}^{\mathrm{Pass}}(W_{\Pi^1}) \leq t_1 \cdot \left[\mathrm{Com}(|m_1 \cdot (W_{a_1} + W_{c_1} + W_{a_2})|) + (W_{a_1} + W_{c_1} + W_{a_2})\right]$$

With W_{Π^1} being the communication complexity of the original interactive protocol with the messages (a_1, c_1, a_2) which have communication complexity W_{a_1}, W_{c_1} and W_{a_2}. Furthermore, $m_1 \geq n_1$ is the number of distinct challenges the prover must answer in each repetition, t_1 is the amount of repetitions, and $\mathrm{Com}(|x|)$ *is a function which describes the size of the commitment with input size $|x|$ based on the commitment scheme used.*

Proof. As established in the runtime analysis, the Pass compiler relies on a "cut-and-choose" mechanism where the prover must provide a commitment to the answers for m_1 distinct challenges in each of the t_1 parallel repetitions. In each repetition, the prover must commit to a transcript size of size $m_1 \cdot (W_{a_1} + W_{c_1} + W_{a_2})$ which leads to a message of size $\mathrm{Com}(|m_1 \cdot (W_{a_1} + W_{c_1} + W_{a_2})|)$. Furthermore, the prover needs to send the opening of one transcript which leads to a communication complexity of $(W_{a_1} + W_{c_1} + W_{a_2})$. The prover has to send these things once for each repetition, so t_1 times, which concludes the proof. □

4.2.4.2. Iterative Communication Function for Fischlin

Lemma 15 (Communication for Fischlin)**.** *The iterative communication complexity function for Fischlin is:*

$$\mathrm{g}_{\mathrm{com}}^{\mathrm{Fischlin}}(W_{\Pi^1}) \leq t_1 \cdot (W_{a_1} + W_{c_1} + W_{a_2})$$

With W_{Π^1} being the communication complexity of the original interactive protocol with the messages (a_1, c_1, a_2) which have communication complexity W_{a_1}, W_{c_1} and W_{a_2}. Furthermore, t_1 is the amount of parallel repetitions.

Proof. In the Fischlin compiler, the prover must produce and send t_1 accepting transcripts of the underlying 3-move protocol that additionally satisfy the proof-of-work condition. Each such transcript consists of all three messages of the base protocol, namely (a_1, c_1, a_2).

Since the iterative communication function is defined for 3-move protocols, the communication cost of a single repetition is exactly

$$W_{a_1} + W_{c_1} + W_{a_2}.$$

Because the prover includes one complete transcript for each of the t_1 repetitions, the total communication complexity is obtained by multiplying this quantity by t_1. Hence, the overall communication complexity is

$$t_1 \cdot (W_{a_1} + W_{c_1} + W_{a_2}),$$

which proves the claim. □

4.2.4.3. Iterative Communication Function for Kondi-Shelat

Lemma 16 (Communication for Kondi-Shelat)**.** *The iterative communication complexity function for Kondi-Shelat is:*

$$g_{\text{com}}^{\text{Kondi-Shelat}}(W_{\Pi^1}) \leq r_1 \cdot t_1 \cdot (W_{a_1} + W_{c_1} + W_{a_2})$$

With W_{Π^1} being the communication complexity of the original interactive protocol with the messages (a_1, c_1, a_2) which have communication complexity W_{a_1}, W_{c_1} and W_{a_2}. Furthermore, r_1 is the amount of transcripts with the same hash value the prover has to find per repetition, and t_1 is the amount of repetitions.

Proof. In each repetition, the Kondi-Shelat compiler requires the prover to find a set of r_1 transcripts that all hash to the same value. In order for the verifier to check this condition, the prover must include all r_1 transcripts in the proof.

Each transmitted transcript is a full transcript of the underlying 3-move protocol and therefore has communication complexity

$$W_{a_1} + W_{c_1} + W_{a_2}$$

Thus, a single repetition contributes

$$r_1 \cdot (W_{a_1} + W_{c_1} + W_{a_2})$$

to the total communication complexity. Since this is done for each of the t_1 repetitions, the overall communication complexity is

$$r_1 \cdot t_1 \cdot (W_{a_1} + W_{c_1} + W_{a_2})$$

which proves the claim. □

4.3. Zero-Knowledge

In this chapter we will prove that $C(\Pi^\mu)$ is zero-knowledge according to Definition 10. To do that we will first prove Theorem 17, which will give us the tools to prove zero-knowledge in Theorem 18.

Before presenting this proof, it is crucial to understand why a standard special Honest-Verifier Zero-Knowledge (sHVZK) assumption is insufficient for our modular multi-round compiler. Our compilation process exposes not just a single transcript, but a comprehensive tree of related transcripts (with branching factor t_i). Within this tree, the verifier observes multiple accepting continuations that share the same initial prover messages but diverge on subsequent challenges. This exposure can reveal information that remains perfectly hidden in the single-transcript setting of ordinary sHVZK. For instance, if an initial prover message commits to a polynomial and subsequent challenges act as evaluation points, observing t_i accepting branches leak enough evaluations to interpolate the polynomial entirely if the degree of the polynomial is smaller than t_i. To prevent such leakage, we require the stronger notion of 2-Stage special Honest-Verifier Zero-Knowledge (2SsHVZK). This property guarantees the existence of a simulator that can generate the prover's initial message and a valid internal state after the first challenge without any knowledge of the witness such that the result is indistinguishable from a real execution. Because the remainder of the protocol relies solely on this simulated state rather than the witness, the above described attack is not possible anymore when using 2SsHVZK.

Building on this requirement, Theorem 17 states that a multi-round 2SsHVZK protocol guarantees sHVZK in the last recursive Σ-protocol Π_1^Σ during the execution of $C(\Pi^\mu)$. This is necessary because the sle-compilers (Chapter 3) require the input protocol for S to be sHVZK, to guarantee zero-knowledge for the non-interactive output protocol. We specifically restrict this proof to the "first" round Π_1^Σ because according to Definition 8, everything after the first round can be calculated honestly, making the existence of a simulator trivial.

Theorem 17 (2SsHVZK $\implies$ sHVZK for Π_1^Σ)**.** *Let $\Pi^\mu = ((P_1, P_2), V)$ be a $(2\mu + 1)$-move interactive public-coin protocol that satisfies 2-Stage special Honest-Verifier Zero-Knowledge (2SsHVZK) with 2SsHVZK error ϵ_{2S}. Let C be the recursive compiler defined in Chapter 4. We prove that when applying it to Π^μ, the first round intermediate 3-move Σ-protocol $\Pi_1^\Sigma = (\mathbb{x}, (a_1, c_1, \pi_2))$ upon which the sle-compiler S is applied, satisfies special Honest-Verifier Zero-Knowledge (sHVZK) and has sHVZK error of at most ϵ_{2S}.*

Proof. We split the proof in two parts. First we describe the simulator for Π_1^Σ and then in the second part, we analyse the protocols zero-knowledge error.

The Simulator for Π_1^Σ: To prove that the intermediate Σ-protocol Π_1^Σ is sHVZK, we must show the existence of a PPT simulator $Sim_{\Pi_1^\Sigma}$ that, given a statement $\mathbb{x}$ and a challenge c_i, can output a simulated transcript (a_1, c_1, π_2) that is computationally indistinguishable from a real transcript without requiring access to a witness $\mathbb{w}$. Note that due to the 2SsHVZK property of Π^μ, there exists a simulator Sim_{2S} that generates a simulated first message and internal state $(a_1, c_1, st_1) \leftarrow Sim_{2S}(\mathbb{x}, c_1)$ without knowledge of the witness $\mathbb{w}$, such that (a_1, c_1, st_1) is indistinguishable from the output of the honest first-stage prover $P_1(\mathbb{x}, \mathbb{w})$ who interacts with the honest Verifier.

We construct the simulator $Sim_{\Pi_1^\Sigma}(\mathbb{x}, c_1)$ as follows:

- $Sim_{\Pi_1^\Sigma}(\mathbb{x}, c_1)$ invokes $Sim_{2S}(\mathbb{x}, c_1)$ to obtain (a_1, c_1, st_1).
- Using the given challenge c_1 and the simulated state st_1, $Sim_{\Pi_1^\Sigma}$ calculates π_2 honestly by running **RecProof**$(2, (a_1, c_1), \mathbb{x})$. Note that the prover algorithm within Π_i^Σ is now P_2 which does not require the witness any more.

sHVZK Error Analyis: We prove that Π_1^Σ has sHVZK error at most ϵ_{2S} with a cryptographic reduction using the simulator $Sim_{\Pi_1^\Sigma}(\mathbb{x}, c_1)$. Suppose that there exists a PPT adversary $\mathcal{A}_{\mathsf{sHVZK}}$ that can distinguish transcripts for protocol Π_1^Σ: $tr_{real} = (a_1, c_1, \pi_2)_{real}$ and $tr_{sim} = (a_1, c_1, \pi_2)_{sim}$ with an advantage $\epsilon' > \epsilon_{2S}$.

Using $\mathcal{A}_{\mathsf{sHVZK}}$, we can construct a PPT reduction $\mathcal{A}_{2S}$ that breaks the 2SsHVZK property of the underlying protocol. $\mathcal{A}_{2S}$ operates as follows:

1. $\mathcal{A}_{2S}$ is given a state tuple (a_1, c_1, st_1), which is either generated by the interaction of an honest first-stage prover $P_1(\mathbb{x}, \mathbb{w})$ with the verifier, or by the simulator $Sim_{2S}(\mathbb{x}, c_1)$.
2. $\mathcal{A}_{2S}$ computes the remainder of the protocol (π_2) by running **RecProof** in combination with the honest second-stage prover algorithm P_2
3. $\mathcal{A}_{2S}$ feeds the resulting transcript (a_1, c_1, π_2) to the adversary $\mathcal{A}_{\mathsf{sHVZK}}$ and outputs whatever $\mathcal{A}_{\mathsf{sHVZK}}$ outputs.

Because the algorithm P_2 and **RecProof** are both PPT functions, if $\mathcal{A}_{\mathsf{sHVZK}}$ could distinguish between $(a_1, c_1, \pi_2)_{real}$ and $(a_1, c_1, \pi_2)_{sim}$ with an advantage $\epsilon' > \epsilon_{2S}$, there would be a PPT Attacker $\mathcal{A}_{2S}$ who could decide between $(a_1, c_1, st_1)_{real}$ and $(a_1, c_1, st_1)_{sim}$ with probability ϵ', which would break the assumption that for all PPT attackers ϵ_{2S} is the maximum advantage. □

Having established that Π_1^Σ satisfies sHVZK, we now turn to the zero-knowledge analysis of the compiled protocol $C(\Pi^\mu)$.

Definition 22 (Zero-Knowledge Error Function of a sle-Compiler)**.** *Let S be a sle-compiler and let Π^{Σ} be a sHVZK 3-move Σ-protocol with sHVZK error ϵ. We define $\mathrm{f}_S(param, \epsilon) \rightarrow [0,1]$ as the zero-knowledge error function for compiler S which given the sHVZK error ϵ of the input protocol Π^{Σ} and the relevant parameters outputs the non-interactive zero-knowledge error ϵ_{ni}(Definition 10) of the compiled non-interactive output protocol $\Pi_{\mathsf{ni}} = S(\Pi^{\Sigma})$.*

Theorem 18 (Zero-Knowledge)**.** *Let $\Pi^{\mu} = (P, V)$ be a $(2\mu+1)$-move interactive protocol that satisfies 2-Stage special Honest-Verifier Zero-Knowledge (2SsHVZK) with 2SsHVZK error ϵ_{2S}. Let C be the modular compiler introduced in Chapter 4, and let $\mathcal{S}$ (with zero-knowledge error function f_S) denote the underlying sle-compiler utilized for the compression of the first round Σ-protocol Π_1^{Σ}.*

We show that the compiled non-interactive protocol $\Pi_{\mathsf{ni}} = C(\Pi^{\mu})$ satisfies Zero-Knowledge in the Random Oracle Model with zero-knowledge error $\epsilon_{\mathsf{zk}} \leq \mathrm{f}_S(param, \epsilon_{2S})$.

Proof. To prove that the compiled non-interactive protocol $\Pi_{\mathsf{ni}} = C(\Pi^{\mu})$ satisfies Zero-Knowledge in the Random Oracle Model, we must construct a PPT simulator $\mathcal{S}_{\mathsf{ni}}$ which controls the two simulated oracles (RO_{prove}, RO_q) which an attacker $\mathcal{A}$ has at his disposal.

First we explain how the simulator answers to **direct oracle queries** made by the prover:

The simulator maintains one global table T. On input a query q, it proceeds as follows:

- if $q \in \mathrm{dom}(T)$, return $T[q]$;
- otherwise sample $y \leftarrow \{0,1\}^{\sigma}$ uniformly, set $T[q] := y$, and return y.

Whenever the simulator needs to program the random oracle in order to generate a simulated proof, it writes the programmed value into the same table T. Therefore all answers given to $\mathcal{A}$ and all values used while simulating proofs are consistent with one shared random oracle view.

Now we present the second part of the simulator, which is the **Proof Oracle**. On input $(\mathbb{x}, \mathbb{w})$, the Simulator $\mathcal{S}_{\mathsf{ni}}$, ignores the witness and must return a valid proof π_{sim}.

Recall from Chapter 4 that the final non-interactive proof π_{real} is generated by invoking the underlying sle-compiler S on the top-level intermediate Σ-protocol:

$$\pi_{real} \leftarrow S[\Pi_1^{\Sigma}((), \mathbb{x}, \mathbb{w})]$$

where a transcript for Π_1^{Σ} has the form (a_1, c_1, π_2).

We construct our proof oracle $RO_{prove}(\mathbb{x})$, utilizing the properties established in Theorem 17:

1. **Simulate the Base Protocol:** To calculate a transcript for Π_1^Σ, $RO_{prove}(\mathbb{x})$ uses $Sim_{\Pi_1^\Sigma}$ the sHVZK simulator for Π_1^Σ. It returns a simulated transcript (a_1, c_1, π_2) which —as shown in Theorem 17— can be distinguished from a real transcript with at most ϵ_{2S}.
2. **Apply the sle-Compiler Simulation:** By Theorem 17 Π_1^Σ is sHVZK with sHVZK-error ϵ_{2S}. Furthermore, S with function $\mathrm{f}_S(\mathrm{param}, \epsilon_{2S})$ has a Zero-Knowledge simulator, denoted Sim_S. Our proof oracle $RO_{prove}(\mathbb{x})$ invokes Sim_S which uses the transcripts generated by $Sim_{\Pi_1^\Sigma}$ to output the final non-interactive proof π_{sim}.

These two Oracles define the simulated experiment

$$\mathsf{Exp}^{\Pi_{\mathsf{ni}}}_{ZK,\mathrm{Sim},\mathcal{A}}(\lambda)$$

as described in Definition 10.

Since Π_1^Σ is a 3-move Σ-protocol with sHVZK error ϵ_{2S}, we can use the Zero-Knowledge Error function f_S to calculate the resulting non-interactive zero-knowledge error of the fully compiled protocol $S(\Pi_1^\Sigma)$. Therefore for every adversary $\mathcal{A}$:

$$\left|\Pr\left[\mathsf{Exp}^{\Pi_{\mathsf{ni}}}_{ZK,\mathrm{Real},\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\Pi_{\mathsf{ni}}}_{ZK,\mathrm{Sim},\mathcal{A}}(\lambda) = 1\right]\right| \leq f_S(\mathrm{param}, \epsilon_{2S}).$$

Thus $\Pi_{\mathsf{ni}} = C(\Pi^\mu)$ satisfies non-interactive zero-knowledge in the Random Oracle Model with zero-knowledge error at most $\mathrm{f}_S(\mathrm{param}, \epsilon_{2S})$.

□

In the following we will introduce the zero-knowledge error function $\mathrm{f}_S(\mathrm{param}, \epsilon_{2S})$ for the three sle-compilers introduced in Chapter 3:

For these compilers, the relevant parameters are σ which is the output bits of the Random Oracle, t_1 which is the amount of parallel repetitions, $q_{\mathrm{RO},\mathcal{A}}$ which is the amount of queries an Attacker can make to the Random Oracle and ϵ_{2S} which is the special Honest-Verifier Zero-Knowledge error of the interactive input protocol Π_1^Σ.

- **Fischlin**: The only way that an attacker can distinguish between a real and simulated transcript when using the Fischlin compiler is, if the attacker (who can do at most $q_{\mathrm{RO},\mathcal{A}}$ queries to the ROM), queries exactly the transcript which the simulator needs to program. The probability of that is $\frac{q_{\mathrm{RO},\mathcal{A}}}{2^\sigma}$. Since the Fischlin transformation has t_1 repetitions and therefore needs to program the Random Oracle at t_1 different values, an attacker's advantage is $\frac{t_1 \cdot q_{\mathrm{RO},\mathcal{A}}}{2^\sigma}$. Now if the input

protocol isn't perfectly sHVZK, the probability that in one of the t_1 repetition, the output of the simulator for Π_1^Σ can be distinguished from a real transcript needs to be considered. By a union bound, this adds $t_1 \cdot \epsilon_{2S}$ and the overall error function is:

$$\mathrm{f}_{\text{Fischlin}}(t_1, \sigma, q_{\text{RO},\mathcal{A}}, \epsilon_{2S}) \leq t_1 \cdot \epsilon_{2S} + \frac{t_1 \cdot q_{\text{RO},\mathcal{A}}}{2^\sigma}$$

We want to note, that if we had used the "original" Fischlin compiler as proposed in [Fis05, Sec. 3], there exists an attack on witness indistinguishability. However we do not need to consider that, since we introduced the randomized version in Section 3.2 which Kondi-Shelat presented in their paper [KS22, pg. 10].

- **Pass:** When using the Pass compiler, the proof relies on a cut-and-choose technique where the prover commits to m transcripts for each of the t_1 repetitions. To simulate this, the simulator generates 1 valid simulated transcript (using $Sim_{\Pi_1^\Sigma}$) and $m-1$ invalid/dummy transcripts per repetition. It commits to all of them and programs the Random Oracle to output the indexes of the valid transcripts. Since he only queries the random oracle once

$$(q_1, \ldots, q_{t_1}) = \mathcal{RO}\left(\mathbb{x}, \left(a_i, (c_{i,j}, com_{i,j})_{j=1}^m\right)_{i=1}^t\right)$$

to get all the challenges, the simulator only needs to program the $\mathcal{RO}$ at this exact spot. The zero-knowledge property breaks if the adversary queries the Random Oracle on the exact commitments before the simulator has a chance to program the oracle. Assuming the commitment scheme provides at least σ bits of entropy, the probability of an adversary with $q_{\text{RO},\mathcal{A}}$ queries guessing the commitment sequence is bounded by $\frac{q_{\text{RO},\mathcal{A}}}{2^\sigma}$. Additionally, because the simulator must open one valid simulated transcript for each of the t_1 repetitions, the underlying 2SsHVZK error scales by t_1. The overall zero-knowledge error function is:

$$\mathrm{f}_{\text{Pass}}(t_1, \sigma, q_{\text{RO},\mathcal{A}}, \epsilon_{2S}) \leq t_1 \cdot \epsilon_{2S} + \frac{q_{\text{RO},\mathcal{A}}}{2^\sigma}$$

- **Kondi-Shelat:** The Kondi-Shelat compiler leverages the birthday paradox, requiring the prover to find r distinct transcripts that hash to the exact same value in each of the t_1 repetitions. To simulate this in the Random Oracle Model, the simulator must program hash collisions for r distinct valid transcripts per repetition. Consequently, the simulator invokes the underlying 2SsHVZK simulator $Sim_{\Pi_1^\Sigma}$ a total of $t_1 \cdot r$ times, accumulating a base simulation error of $t \cdot r \cdot \epsilon_{2S}$. Similar to Fischlin, the simulation fails if the adversary queries the Random Oracle on any of these programmed transcripts before the simulator programs them. Since there

are $t_1 \cdot r$ programmed queries, the probability that an adversary with $q_{\mathrm{RO},\mathcal{A}}$ queries hits any of them is $\frac{t_1 \cdot r \cdot q_{\mathrm{RO},\mathcal{A}}}{2^\sigma}$. Thus, the overall zero-knowledge error function is:

$$\mathrm{f}_{\text{Kondi-Shelat}}(t_1, \sigma, q_{\mathrm{RO},\mathcal{A}}, r, \epsilon_{2S}) \leq t_1 \cdot r \cdot \epsilon_{2S} + \frac{t_1 \cdot r \cdot q_{\mathrm{RO},\mathcal{A}}}{2^\sigma}$$

Since all of these functions are negligible if the parameters are chosen correctly, zero-knowledge is preserved.

4.4. Soundness

In the following we prove that $C(\Pi^\mu)$ is straight-line knowledge-sound, meaning that there exists an Extractor which can recompute a witness from a valid proof with overwhelming probability. We split this Extractor into two crucial parts. The first part generates a valid transcript-tree and the second part extracts a witness from this tree. Since we demand (strong)-special-soundness from the input protocol, the second part is guaranteed if the first part succeeds; therefore we mainly focus on generating the transcript-tree. At the end of this section however, we will note some things about the second extractor part, especially the need for strong-special-soundness depending on which Σ-sle-compiler was used in generating the proof π_1.

Our unique modular approach of building the compiler C forces us to make the extractor for $C(\Pi^\mu)$ modular as well, using the extractors of the underlying Σ-sle compilers. We require the Σ-compilers to be tt-sle Definition 18, which means that they can extract a valid transcript-tree for a relation rather than a witness. We define the extraction relation

$$\tilde{R}_i(h) = \{(\mathbb{x}_i, tt) | tt \text{ is a valid } n_i \text{ transcript-tree for } \Pi_i^\Sigma(h)\}$$

and $\Pi_i^\Sigma(h)$ is the Σ protocol (a_i, c_i, π_{i+1}) with $\mathbb{x}_i = \mathbb{x}||h_{i-1}$.

4.4.1. The Tree Extractor

In the following we present the extractor $\mathcal{E}_C$ which given a statement $\mathbb{x}$ and a valid proof π_1 for the protocol $\Pi_{\mathsf{ni}} = C(\Pi^\mu)$, extracts a valid $(n_1, \cdots, n_\mu)$-transcript-tree. From this transcript tree, a witness $\mathbb{w}$ for the original extraction relation $\tilde{R}$ can be calculated.

Definition 23 (Recursive Transcript-Tree Extractor $\mathcal{E}_C$)**.** *Let Π^μ be a $(2\mu + 1)$-move interactive protocol with $(n_1, \cdots, n_\mu)$-(strong)-special soundness for extraction relation $\tilde{R}$. Let S be a sle-Compiler with corresponding Extractor $\mathcal{E}_S$. $\mathcal{E}_S$ outputs the a valid n_i transcript-tree for $\tilde{R}_i(h)$ if successful and outputs $\perp$ if not. We define the recursive extractor $\mathcal{E}_C(\mathbb{x}, \pi_1, Q_{\mathcal{RO}})$ for the compiled protocol $\Pi_{\mathsf{ni}} = C(\Pi^\mu)$ in Figure 4.1.*

In Figure 4.1, we use that the extractor has access to a specific Random Oracle $\mathcal{RO}_{h_{i-1}}$ for each prefix h_{i-1}. In the following we argue why this is possible using the principle of Domain Separation. In the definition of the prover algorithm Π_i^Σ (see Chapter 4), the statement for the new Σ-protocol we create is updated to $\mathbb{x}_i = \mathbb{x}||h_{i-1}$. During the calculation of a proof π_i, we demand that all prover queries to the Random Oracle $\mathcal{RO}$ start with the statement $\mathbb{x}_i$. Then we can model each $\mathcal{RO}$ query q made during the

The Extractor $\mathcal{E}_C(\mathbb{x}, \pi_1, Q_{\mathcal{RO}})$

1. Initialize $h_0 = ()$.
2. Return **RecExtract**$(1, h_0, \pi_1)$.

RecExtract(i, h_{i-1}, π_i)

- **Base Case** $(i = \mu + 1)$:
  1. The input $\pi_{\mu+1}$ corresponds to the final protocol message $a_{\mu+1}$.
  2. Return the leaf node $a_{\mu+1} = \pi_{\mu+1}$.
- **Recursive Step** $(1 \leq i \leq \mu)$:
  1. Define the statement for the current round as $\mathbb{x}_i = (\mathbb{x}||h_{i-1})$.
  2. Invoke the underlying extractor $\mathcal{E}_S$ on the non-interactive proof π_i with access to the list of Random Oracle queries to the Random Oracle $\mathcal{RO}_{h_{i-1}}$:

$$T_i \leftarrow \mathcal{E}_S(\mathbb{x}_i, \pi_i, Q_{\mathcal{RO}_{h_{i-1}}})$$

  3. If $\mathcal{E}_{S[n_i]}$ outputs $\perp$, abort everything and let $\mathcal{E}_C$ return $\perp$
  4. if $\mathcal{E}_{S[n_i]}$ is successful it outputs a witness for $\tilde{R}_i(h_{i-1})$ which per definition is a valid n_i-transcript-tree:

$$T_i = \left\{(a_i, c_i^{(j)}, \pi_{i+1}^{(j)})\right\}_{j=1}^{n_i}$$

  5. For each $j \in \{1, \cdots, n_i\}$:
     a) Update the history: $h_i^{(j)} = h_{i-1}||(a_i, c_i^{(j)})$.
     b) Recursively extract the subtree:

$$\tau_j \leftarrow \textbf{RecExtract}(i + 1, h_i^{(j)}, \pi_{i+1}^{(j)})$$

  6. Return the tree node composed of root a_i, edges $c_i^{(1)}, \cdots, c_i^{(n_i)}$, and the corresponding subtrees $\tau_1, \cdots, \tau_{n_i}$.

Figure 4.1.: The recursive Extractor $\mathcal{E}_C$

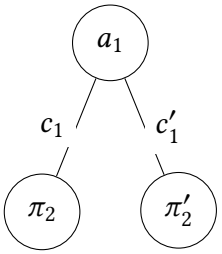
calculation of π_i as $\mathcal{RO}(\mathbb{x}_i||q')$. Note that since $\mathbb{x}_i$ contains h_{i-1}, the prefix $\mathbb{x}_i$ has a different length for proofs on a different level and is unique for the calculation of two different π_i on the same level.

Therefore the Random Oracle queries made to generate π_i can be understood as queries made to a new Random Oracle $\mathcal{RO}_{h_{i-1}}(q) := \mathcal{RO}(\mathbb{x}_i||q)$. Since $\mathcal{RO}$ behaves as a random function, and the prefixes $\mathbb{x}_i$ are distinct for every node in the transcript-tree, the functions $\mathcal{RO}_{h_{i-1}}$ for each h_{i-1} behave as independent Random Oracles. Any attack on a particular derived oracle $\mathcal{RO}_{h_{i-1}}$ immediately yields an attack on the Random Oracle $\mathcal{RO}$, since $\mathcal{RO}_{h_{i-1}}$ is just $\mathcal{RO}$ restricted to the domain prefixed by $\mathbb{x}_i$.

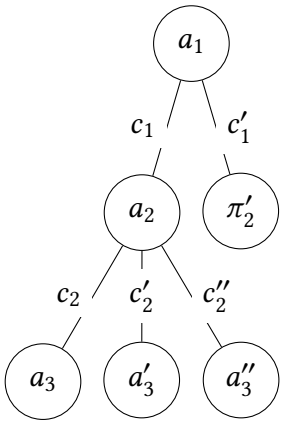
In the following sections Section 4.4.1.1 and Section 4.4.1.2 we analyse $\mathcal{E}_C$ defined in Figure 4.1 and prove, that it actually extracts a $(n_1, \cdots, n_\mu)$-transcript-tree in polynomial time. For the analysis we assume that $\mathcal{E}_S$ (the extractor for the sle-Compiler) extracts a transcript-tree for a corresponding proof with probability 1 and never outputs $\perp$. We will consider the real probability for a successful extraction in Theorem 21 where we analyse the knowledge-soundness error for $C(\Pi^\mu)$.

To avoid confusion and help with understanding the recursive nature of $\mathcal{E}_C$, we introduce a simplified example-execution of $\mathcal{E}_C$ for a $(2, 3)$-special sound protocol in Figure 4.2:

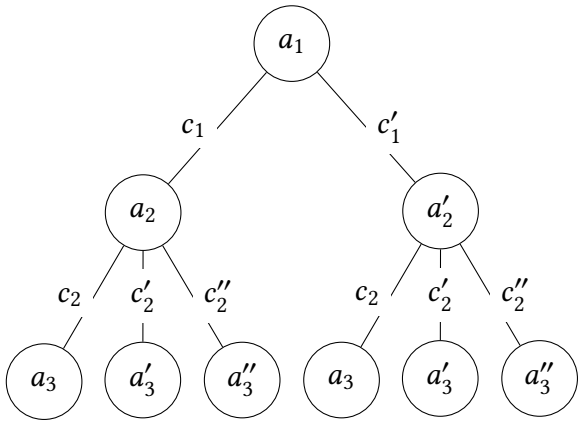
(a) Initial state: A single valid non-interactive proof π_1



(b) $\mathcal{E}_{S,1}$ extracts a $n_1 = 2$-transcript tree



(c) Application of $\mathcal{E}_{S,2}$ on π_2 extracts the left $n_2 = 3$ subtree



(d) Application of $\mathcal{E}_{S,2}$ on π_2' extracts the right $n_2 = 3$ subtree and completes the full (2,3)-transcript tree

Figure 4.2.: Step-by-step recursive extraction process for a protocol with $(2, 3)$-special soundness.

Finally before starting the proofs, we want to clarify some notation used in the following. Since we use $\mathcal{E}_C$ to extract transcript-trees for Protocols of different length, we write $\mathcal{E}_{C(\Pi^\mu)}$ to reference the extractor which returns a $(n_1, \cdots, n_\mu)$ transcript-tree of accepting transcripts for Π^μ from a valid proof for $C(\Pi^\mu)$. Furthermore in each level i, n_i different transcripts have to be extracted and n_i can be different on each level. This is why we use the notation $\mathcal{E}_{S[n_i]}$ for the Extractor which on input π_i returns n_i valid transcripts for $\Pi_i^\Sigma(h)$.

4.4.1.1. Tree Extraction

We prove that the extractor defined in Figure 4.1 returns a $(n_1, \cdots, n_\mu)$ transcript-tree if π_1 is valid and $\mathcal{E}_{S[n_i]}$ never outputs $\perp$.

Theorem 19. *Let Π^μ be a $(2\mu+1)$-move interactive protocol with $(n_1, \cdots, n_\mu)$-(strong)-special soundness for relation $\tilde{R}$. Let S be a tt-sle-Compiler for a n_Σ-special sound Σ-Protocol Π^Σ with an Extractor $\mathcal{E}_{S[n_\Sigma]}$.*

Given a valid proof π_1 for $C(\Pi^\mu)$, $\mathcal{E}_{C(\Pi^\mu)}$ as defined in Figure 4.1, returns a $(n_1, \cdots, n_\mu)$-transcript-tree of valid transcripts for Π^μ.

Proof. We prove the theorem by a complete induction over the length of the underlying protocol Π^μ.

Base Case$(\mu = 1)$: For $\mu = 1$, Π^μ is just a standard n_1-special sound Σ-protocol and applying $\mathcal{E}_{C(\Pi^1)}$ to a valid proof π_1 for $C(\Pi^1)$ per definition just applies $\mathcal{E}_{S[n_1]}$ on π_1. Since S is a tt-sle compiler, $\mathcal{E}_{S[n_1]}$ per definition extracts n_1 valid transcripts which can be arranged as a n_1-transcript-tree. Therefore the base case holds.

Induction Hypothesis: For any $\mu \in \mathbb{N}$ and a $(2\mu+1)$-move, $(n_1, \cdots, n_\mu)$-special sound protocol Π^μ, $\mathcal{E}_{C(\Pi^\mu)}$ extracts a valid $(n_1, \cdots, n_\mu)$ transcript-tree for $\tilde{R}$ from a valid proof $\pi_1 = C(P^\mu(\mathbb{x}, \mathbb{w}))$.

Induction Step$(\mu \to \mu+1)$: Consider a $(2(\mu+1)+1)$-move interactive protocol $\Pi^{\mu+1}$ with $(n_1, \cdots, n_{\mu+1})$-special soundness. Let π_1 be a valid proof for the compiled protocol $C(\Pi^{\mu+1})$.

The first call to the recursive extractor, **RecExtract**$(1, h_0, \pi_1)$, invokes the underlying single-round extractor $\mathcal{E}_{S[n_1]}$ on π_1. Since S is a tt-sle compiler for $\tilde{R}_1$, by definition $\mathcal{E}_{S[n_1]}$ extracts a set of n_1 valid transcripts for the first round:

$$T_1 = \left\{(a_1, c_1^{(j)}, \pi_2^{(j)})\right\}_{j=1}^{n_1}$$

Here, each tuple consists of the first message a_1, a challenge $c_1^{(j)}$, and a response which, per definition, is a valid proof $\pi_2^{(j)}$ for the remaining protocol steps.

For each $j \in \{1, \cdots, n_1\}$, the extractor updates the history to $h_1^{(j)} = h_0 || (a_1, c_1^{(j)})$. The remaining protocol, conditioned on this history, is effectively a $(2\mu+1)$-move protocol $\Pi^\mu(h_1^{(j)})$ with $(n_2, \cdots, n_{\mu+1})$-special soundness.

By the Induction Hypothesis, calling $\mathcal{E}_C$ on $\pi_2^{(j)}$ successfully extracts a valid $(n_2, \cdots, n_{\mu+1})$-transcript-tree, denoted as τ_j.

Finally, the extractor constructs a new tree with root a_1 and edges $c_1^{(1)}, \cdots, c_1^{(n_1)}$ pointing to the subtrees $\tau_1, \cdots, \tau_{n_1}$. Since τ_j are valid $(n_2, \cdots, n_{\mu+1})$-trees and the corresponding prefixes $(a_1, c_1^{(j)})$ are embedded into the calculation of $\pi_2^{(j)}$, the full transcripts $(a_1, c_1^{(j)}, \cdots, c_{\mu+1}, a_{\mu+2})$ are valid. Therefore, the resulting composition is a valid $(n_1, n_2, \cdots, n_{\mu+1})$-transcript-tree for $\Pi^{\mu+1}$. □

4.4.1.2. Efficiency of the Extractor

Theorem 20. *Let Π^μ be a $(2\mu+1)$-move interactive protocol with $(n_1, \cdots, n_\mu)$-(strong)-special soundness. Let S be a tt-sle-Compiler with an Extractor $\mathcal{E}_{S[n_\Sigma]}$ with extraction runtime $t_{\mathcal{E}_{S[n_\Sigma]}}$. Given a valid proof π_1 for $C(\Pi^\mu)$, $\mathcal{E}_C$ as defined in Figure 4.1 runs in time*

$$t_{\mathcal{E}_{C(\Pi^\mu)}} \leq \sum_{i=1}^{\mu} \left(\left[\prod_{j=1}^{i-1} n_j \right] \cdot t_{\mathcal{E}_{S[n_i]}} \right)$$

with $\prod_{j=1}^{0} n_i = 1$

Proof. We prove the efficiency of the extractor by a complete induction over the length of the protocol Π^μ.

Before we start, note that if the $\mathcal{E}_{C(\Pi^\mu)}$ aborts and returns $\perp$, its runtime is strictly smaller than if it succeeds. Therefore in the following we only consider a successful extraction.

Base Case $(\mu = 1)$: For $\mu = 1$, Π^μ is a 3-round Σ-protocol, so π_1 is the result from applying the compiler S to Π^1.

The extractor $\mathcal{E}_C$ runs the extractor $\mathcal{E}_{S[n_1]}$ once to extract the n_1 different transcripts, which takes exactly $t_{\mathcal{E}_{S[n_1]}}$. Therefore the overall runtime is $t_{\mathcal{E}_{C(\Pi^1)}} = t_{\mathcal{E}_{S[n_1]}}$.

Induction Hypothesis: For $\mu = n$ with $n \in \mathbb{N}$, the claim holds that $\mathcal{E}_{C(\Pi^\mu)}$ can extract a valid $(n_1, \cdots, n_\mu)$ transcript-tree for an underlying $(2\mu+1)$-move protocol from a proof π_1 in time:

$$t_{\mathcal{E}_{C(\Pi^\mu)}} \leq \sum_{i=1}^{\mu} \left(\left[\prod_{j=1}^{i-1} n_j \right] \cdot t_{\mathcal{E}_{S[n_i]}} \right)$$

Induction Step $(\mu \to \mu+1)$: Given $(2(\mu+1)+1)$-move $(n_1, \cdots, n_{\mu+1})$-special-sound protocol $\Pi^{\mu+1}$ and a valid proof π_1 for $C(\Pi^{\mu+1})$, we analyse the time it takes the extractor $\mathcal{E}_{C(\Pi^{\mu+1})}$ to extract a valid $(n_1, \cdots, n_{\mu+1})$-transcript-tree.

As used in the prior proofs, we split the application of C on $\Pi^{\mu+1}$ into two steps, first applying C^1 to $\Pi^{\mu+1}$ to get Π^μ and then applying C to the resulting protocol. Since $C^1(\Pi^{\mu+1})$ is a $(2\mu+1)$-move interactive protocol the Induction Hypothesis, states that extracting a $(n_1, \cdots, n_\mu)$ transcript-tree with valid transcripts for Π^μ takes

$$t_{\mathcal{E}_{C(\Pi^\mu)}} \leq \sum_{i=1}^{\mu} \left(\left[\prod_{j=1}^{i-1} n_j \right] \cdot t_{\mathcal{E}_{S[n_i]}} \right)$$

The extracted transcripts are valid for $C^1(\Pi^{\mu+1}) = \Pi^{\mu}$, and therefore have the form $(a_1, c_1, a_2, \cdots, c_\mu, \pi_{\mu+1})$. To extract a $(n_1, \cdots, n_{\mu+1})$ transcript-tree with valid transcripts for $\Pi^{\mu+1}$, we need to extract a valid $n_{\mu+1}$ transcript-tree from each $\pi_{\mu+1}$ (each leaf of the $(n_1, \cdots, n_\mu)$-transcript-tree). The amount of leaves can be calculated by

$$\prod_{j=1}^{\mu} n_j$$

Each extraction on level $\mu + 1$ takes $t_{\mathcal{E}_{S[n_{\mu+1}]}}$, so overall:

$$\begin{aligned} t_{\mathcal{E}_{C(\Pi^{\mu+1})}} &\leq \sum_{i=1}^{\mu} \left(\left[\prod_{j=1}^{i-1} n_j \right] \cdot t_{\mathcal{E}_{S[n_i]}} \right) + \left(\prod_{j=1}^{\mu} n_j \right) \cdot t_{\mathcal{E}_{S[n_{\mu+1}]}} \\ &= \sum_{i=1}^{\mu+1} \left(\left[\prod_{j=1}^{i-1} n_j \right] \cdot t_{\mathcal{E}_{S[n_i]}} \right) \end{aligned}$$

□

4.4.2. Knowledge-Soundness

From Section 4.4.1.1 and Section 4.4.1.2 we know, that if each extraction of $\mathcal{E}_{S[n_i]}$ is successful, $\mathcal{E}_C$ returns a valid $(n_1, \cdots, n_\mu)$-transcript-tree in polynomial time. In the following we look at the knowledge-soundness error of $C(\Pi^\mu)$. Since we require the input protocols to be $(n_1, \cdots, n_\mu)$-(strong)-special-sound, the extraction of a witness is guaranteed as long as the tree-extractor can successfully output a valid transcript-tree. Therefore the knowledge-soundness-error is the probability that $\mathcal{E}_C$ can not extract a valid transcript-tree from a valid proof π_1, in other words— the probability that at least one $\mathcal{E}_{S[n_i]}$ outputs $\perp$.

Theorem 21. *Let Π^μ be a $(2\mu + 1)$-move interactive protocol with $(n_1, \cdots, n_\mu)$-special soundness for extraction relation $\tilde{R}$. Let $\sigma, q_{\mathrm{RO},\mathcal{A}}, q^{(i)}_{\mathrm{RO},\mathcal{A}}, s$ be the length of the Random Oracle's output; the amount of oracle queries an Adversary can make overall; the amount of oracle queries an Adversary can make to create a proof π_i in round i; the maximum length of the statement $\mathbb{x}$.*

Let S be a tt-sle-Compiler (and let Π^Σ be a n_Σ-special-sound Σ-protocol) where $S(\Pi^\Sigma)$ has Straight-line knowledge-soundness Error (Definition 17) $\kappa_{\mathcal{E}_{S[n_\Sigma]}}(\sigma, q^{(i)}_{\mathrm{RO},\mathcal{A}}, s)$.

The knowledge-soundness error for $C(\Pi^\mu)$ is:

$$\kappa_{\mathcal{E}_{C(\Pi^\mu)}}(\sigma, q_{\mathrm{RO},\mathcal{A}}, s) = \sum_{i=1}^{\mu} \left(\left[\prod_{j=1}^{i-1} n_j \right] \cdot \kappa_{\mathcal{E}_{S[n_i]}}(\sigma, q^{(i)}_{\mathrm{RO},\mathcal{A}}, s) \right)$$

where we define the empty product $\prod_{j=1}^{0} n_j = 1$.

We briefly clarify the difference between the parameters $q_{\mathrm{RO},\mathcal{A}}$ and $q^{(i)}_{\mathrm{RO},\mathcal{A}}$ used in Theorem 21. The parameter $q_{\mathrm{RO},\mathcal{A}}$ denotes the total number of random-oracle queries that an adversary can make while producing the full proof for $C(\Pi^\mu)$. In contrast, $q^{(i)}_{\mathrm{RO},\mathcal{A}}$ denotes an upper bound on the number of random-oracle queries made to any single derived oracle $\mathcal{RO}_{h_{i-1}}$ while producing one valid subproof π_i.

By domain separation, the queries to $\mathcal{RO}_{h_{i-1}}$ are exactly the queries relevant to the extractor

$$\mathcal{E}_{S[n_i]}(\mathbb{x} \| h_{i-1}, \pi_i, Q_{\mathcal{RO}_{h_{i-1}}}).$$

Thus, $q^{(i)}_{\mathrm{RO},\mathcal{A}}$ should be understood as a *local* bound for a single subproof on level i, whereas $q_{\mathrm{RO},\mathcal{A}}$ is the *global* query bound of the adversary.

We state the theorem using the parameters $q^{(i)}_{\mathrm{RO},\mathcal{A}}$ in order to keep the bound modular and to reflect the recursive structure of the extractor. However, in the Random Oracle Model there is no mechanism that enforces a separate budget for each derived oracle $\mathcal{RO}_{h_{i-1}}$. Hence, for an unconditional proof, one may simply instantiate

$$q^{(i)}_{\mathrm{RO},\mathcal{A}} = q_{\mathrm{RO},\mathcal{A}} \qquad \text{for all } i \in \{1, \cdots, \mu\}.$$

Proof. We prove the theorem using a sequence of hybrid experiments to bound the overall knowledge-soundness error. According to Definition 17, a non-interactive argument system is knowledge-sound if the probability that an adversary outputs a valid proof ($V_{\mathsf{ni}}(\mathbb{x}, \pi_1) = 1$) but the extractor fails to output a valid witness is smaller than a certain function. Since we expect that the extraction of a witness from a valid $(n_1, \cdots, n_\mu)$-transcript-tree is guaranteed (see Section 4.4), we will bound the probability that $\mathcal{E}_C$ fails to extract a complete, valid tree given an accepting π_1 to calculate the knowledge-soundness error.

We define a sequence of hybrid experiments $\mathcal{H}_0, \cdots, \mathcal{H}_\mu$ to model the step-by-step top-down extraction process. Our goal is to bound the Probability, that the entire extraction $\mathcal{E}_C(\mathbb{x}, \pi_1)$ fails, but the verifier accepts, π_1.

- **Hybrid $\mathcal{H}_0$:** The experiment simply runs the verifier on the adversary's output. It outputs 0 if $V_{\mathsf{ni}}(\mathbb{x}, \pi_1) = 1$, and 1 otherwise.

- **Hybrid $\mathcal{H}_1$:** The experiment runs $V_{\mathsf{ni}}(\mathbb{x}, \pi_1)$. If it accepts, it invokes $\mathcal{E}_{S[n_1]}$ on π_1. The experiment outputs 0 if the verifier accepts *and* $\mathcal{E}_{S[n_1]}$ successfully extracts a witness for $\tilde{R}_1$ which is a valid n_1-transcript-tree T_1 consisting of transcripts $(a_1, c_1^{(j)}, \pi_2^{(j)})$. If the extraction fails, or the Verifier does not accept, it outputs 1.

  We analyse the probability, that the *Extraction* fails in the first step so the overall probability that the experiment $\mathcal{H}_1$ outputs 1 minus the probability, that the Verifier does not accept the proof: $\Pr[\mathcal{H}_1 = 1] - \Pr[\mathcal{H}_0 = 1]$

  $$\begin{aligned}
  &\Pr[\mathcal{H}_1 = 1] - \Pr[\mathcal{H}_0 = 1] \\
  &= \Pr[V(\mathbb{x}, \pi_1) = 0 \vee \mathcal{E}_{S[n_1]} \text{ fails}] - \Pr[V(\mathbb{x}, \pi_1) = 0] \\
  &= \Pr[V(\mathbb{x}, \pi_1) = 0] + \Pr[\mathcal{E}_{S[n_1]} \text{ fails} \wedge V(\mathbb{x}, \pi_1) = 1] - \Pr[V(\mathbb{x}, \pi_1) = 0] \\
  &= \Pr[\mathcal{E}_{S[n_1]} \text{ fails} \wedge V(\mathbb{x}, \pi_1) = 1] \\
  &\leq \kappa_{\mathcal{E}_{S[n_1]}}(\sigma, q^{(1)}_{\mathsf{RO},\mathcal{A}}, s)
  \end{aligned}$$

  with using the rule: $\Pr[A \vee B] = \Pr[A] + \Pr[B \wedge \neg A]$

- **Hybrid $\mathcal{H}_i$ (for $1 < i < \mu$):** The experiment proceeds as $\mathcal{H}_{i-1}$ and if $\mathcal{H}_{i-1} = 1$, $\mathcal{H}_i$ also returns 1. Otherwise, $\mathcal{H}_i$ uses $\mathcal{E}_{S[n_i]}$ on all leaves (π_i) in the transcript-tree extracted by $\mathcal{H}_{i-1}$. If $\mathcal{H}_{i-1}$ outputs 0, it means that it has successfully extracted a valid $(n_1, \cdots, n_{i-1})$-transcript-tree with $N_i = \prod_{j=1}^{i-1} n_j$ valid proofs π_i as its leaves.

  For $\mathcal{H}_i$, the experiment additionally runs $\mathcal{E}_{S[n_i]}$ on all N_i leaves (π_i). It outputs 0 if all N_i extractions succeed, and 1 otherwise.

- **Hybrid $\mathcal{H}_\mu$ (for $i = \mu$):** Following the rule for **Hybrid** $\mathcal{H}_i$, for the experiment $\mathcal{H}_\mu$, we run $V_{\mathsf{ni}}(\mathbb{x}, \pi_1)$ and $\mathcal{E}_C(\mathbb{x}, \pi_1)$ and return $\overline{V_{\mathsf{ni}}(\mathbb{x}, \pi_1) = 1 \wedge \mathcal{E}_C(\mathbb{x}, \pi_1) \neq \perp}$

We now bound the probability, that an extraction error occurs in the extraction from $\mathcal{H}_{i-1}$ to $\mathcal{H}_i$. We do that in a similar manner to the proof for **Hybrid** $\mathcal{H}_1$ by bounding the following: $\Pr[\mathcal{H}_i = 1] - \Pr[\mathcal{H}_{i-1} = 1]$. We use the notation F_j which is the event that at least one of the N_j extractions of $\mathcal{E}_{S[n_j]}(\mathbb{x}, \pi_j)$ fails.

$$\begin{aligned}
&\Pr[\mathcal{H}_i = 1] - \Pr[\mathcal{H}_{i-1} = 1] \\
&= \Pr[V(\mathbb{x}, \pi_1) = 0 \vee F_1 \vee \cdots \vee F_{i-1} \vee F_i] \\
&- \Pr[V(\mathbb{x}, \pi_1) = 0 \vee F_1 \vee \cdots \vee F_{i-1}]
\end{aligned}$$

Let $A = [V(\mathbb{x}, \pi_1) = 0 \vee F_1 \vee \cdots \vee F_{i-1}]$ and let $B = [F_i]$

$$\begin{aligned}
&= \Pr[A \vee B] - \Pr[A] \\
&= \Pr[A] + \Pr[B \wedge \neg A] - \Pr[A] \\
&= \Pr[B \wedge \neg A] \leq \Pr[B] = \Pr[F_i]
\end{aligned}$$

We are left to bound the probability of F_i meaning that at least one extraction of a leaf of the $(n_1, \cdots, n_{i-1})$-transcript-tree fails. We do that by applying a union bound over the N_i extractions at this specific level:

$$\Pr[F_i] \leq \left(\prod_{j=1}^{i-1} n_j \right) \cdot \kappa_{\mathcal{E}_{S[n_i]}}(\sigma, q^{(i)}_{\mathrm{RO},\mathcal{A}}, s)$$

The total failure event F is defined as the adversary producing a valid proof (which is $\Pr[\mathcal{H}_0 = 0]$), but the full μ-level extraction failing (which means $\mathcal{H}_\mu$ outputs 1). Therefore, the overall knowledge-soundness error is exactly $\Pr[\mathcal{H}_\mu = 1] - \Pr[\mathcal{H}_0 = 1]$.

Using a telescoping sum across all our hybrid experiments, we bound this total error:

$$\kappa_{\mathcal{E}_{C(\Pi^\mu)}}(\sigma, q_{\mathrm{RO},\mathcal{A}}, s) = \Pr[\mathcal{H}_\mu = 1] - \Pr[\mathcal{H}_0 = 1] = \sum_{i=1}^{\mu} (\Pr[\mathcal{H}_i = 1] - \Pr[\mathcal{H}_{i-1} = 1])$$

Substituting our level-by-level bounds, we obtain:

$$\kappa_{\mathcal{E}_{C(\Pi^\mu)}}(\sigma, q_{\mathrm{RO},\mathcal{A}}, s) \leq \sum_{i=1}^{\mu} \left(N_i \cdot \kappa_{\mathcal{E}_{S[n_i]}}(\sigma, q^{(i)}_{\mathrm{RO},\mathcal{A}}, s) \right)$$

Finally, substituting $N_i = \prod_{j=1}^{i-1} n_j$ yields the final knowledge-soundness error bound:

$$\kappa_{\mathcal{E}_{C(\Pi^\mu)}}(\sigma, q_{\mathrm{RO},\mathcal{A}}, s) \leq \sum_{i=1}^{\mu} \left(\left[\prod_{j=1}^{i-1} n_j \right] \cdot \kappa_{\mathcal{E}_{S[n_i]}}(\sigma, q^{(i)}_{\mathrm{RO},\mathcal{A}}, s) \right)$$

□

4.4.3. Witness Extractor

As established in Section 4.4.1, the tree extractor $\mathcal{E}_C$ successfully constructs a valid $(n_1, \cdots, n_\mu)$-transcript-tree with overwhelming probability. The final step is to extract a witness $\mathbb{w}$ for the extraction relation $\tilde{R}$ from that tree.

The feasibility of this extraction depends heavily on the structural properties of the extracted transcript-tree which are dictated by the underlying sle-Compiler used in the calculation of the proof to compress the intermediate protocols Π_i^Σ.

- **Pass Compiler:** The Pass compiler (Section 3.1) strictly enforces that the prover utilizes distinct challenges. Consequently, all extracted transcripts within the resulting tree are guaranteed to have distinct challenge edges meaning that they form $(n_1, \cdots, n_\mu)$-Special-Sound Trees (Definition 12). For these trees, standard $(n_1, \cdots, n_\mu)$-special soundness is sufficient to compute a valid witness for $\tilde{R}$.

- **Fischlin and Kondi-Shelat Compilers:** Conversely, the Fischlin and Kondi-Shelat compilers do not restrict the prover to distinct challenges. This introduces the possibility that the extractor $\mathcal{E}_{S[n_i]}$ outputs n_i valid transcripts that share overlapping challenges. To successfully extract a witness from a transcript-tree containing non-distinct challenges, the underlying interactive protocol must satisfy the stricter requirement of $(n_1, \cdots, n_\mu)$-**strong**-special soundness.

4.5. Analysis

In this section, we synthesize the results from all preceding chapters and analyse how large certain parameters can be, and, more importantly, which trade-offs arise. Furthermore, we establish the final bounds on completeness, efficiency, zero-knowledge, and knowledge-soundness for $C(\Pi^\mu)$.

First, it is necessary to establish the exact scope of this analysis. Overall, we want $C(\Pi^\mu)(\mathbb{x}, \mathbb{w})$ to be secure and efficient for a certain security parameter λ, and statement, witness pair $(\mathbb{x}, \mathbb{w})$ meaning that the knowledge-soundness error of the protocol is at most $2^{-\lambda}$. In the following chapter we will first analyse how large the knowledge-soundness error $\kappa_i = 2^{-\lambda_i}$ of the intermediate Σ-sle-compiled proofs have to be in order to guarantee a knowledge-soundness error of $2^{-\lambda}$ for $C(\Pi^\mu)(\mathbb{x}, \mathbb{w})$. Then we will analyse the runtime, communication efficiency, completeness and knowledge-soundness of $C(\Pi^\mu)$ with S used as the underlying compiler for each Σ-sle-compiler presented in Chapter 3.

Note that we do not consider zero-knowledge in this section, since as proven in Section 4.3, the zero-knowledge error depends only on the first round compilation meaning that $C(\Pi^\mu)$ is zero-knowledge per requirement of S.

4.5.1. Relation between λ and λ_i

Since $\kappa = 2^{-\lambda}$ mainly depends on the knowledge-soundness errors $\kappa_i = 2^{-\lambda_i}$ of the compiled intermediate Σ-protocols $S(\Pi_i^\Sigma)$, we first need to understand how large these can be to ensure $\kappa = 2^{-\lambda}$ for $C(\Pi^\mu)$. The choice of λ_i will also influence other factors like the efficiency of our compiler as explained later on.

From Theorem 21 we know that:

$$\kappa\varepsilon_{C(\Pi^\mu)}(\sigma, q_{\mathrm{RO},\mathcal{A}}, s) = \sum_{i=1}^{\mu} \left(N_i \cdot \kappa\varepsilon_{S[n_i]}(\sigma, q^{(i)}_{\mathrm{RO},\mathcal{A}}, s) \right)$$

where $N_i = \prod_{j=1}^{i} n_j$ is the number of nodes at level i.

To achieve our global target of $\kappa\varepsilon_{C(\Pi^\mu)} \leq 2^{-\lambda}$, we can distribute the acceptable error evenly across all μ rounds. Although this bound is not tight, it is sufficient for establishing the required upper bound. We require each term in the sum to be bounded by $\frac{2^{-\lambda}}{\mu}$. Therefore we know:

$$
\begin{aligned}
N_i \cdot 2^{-\lambda_i} &\leq \frac{2^{-\lambda}}{\mu} \\
2^{-\lambda_i} &\leq \frac{2^{-\lambda}}{N_i \cdot \mu} && \text{(divide both sides by } N_i > 0\text{)} \\
2^{\lambda_i} &\geq 2^{\lambda} \cdot N_i \cdot \mu && \text{(take reciprocals and flip inequality)} \\
\lambda_i &\geq \log_2(2^{\lambda} \cdot N_i \cdot \mu) && \text{(apply } \log_2\text{)} \\
\lambda_i &\geq \lambda + \log_2(N_i) + \log_2(\mu) && \text{(logarithm product rule)}
\end{aligned}
$$

In our analysis we require $n_i \in O(1)$, so therefore $N_i \in O(c^{\mu})$ with c being a constant and $\log_2(N_i) \in O(\mu \cdot \log(c)) = O(\mu)$. Using these findings, we can finalize the relation between λ and λ_i:

$$\lambda_i \geq \lambda + O(\mu) + O(\log(\mu)) = \lambda + O(\mu)$$

We conclude that the security parameter of the intermediate Σ-protocol depends on the overall security parameter as well as the amount of rounds of the protocol.

Knowledge Soundness of the Σ-Protocols Now that we know that each individual compiled Σ-protocol needs a knowledge-soundness error of $\kappa_i \leq 2^{-(\lambda+O(\mu))}$, we can continue by analysing the parameter-choices for S to reach that knowledge-soundness error. In particular, t_i the amount of parallel repetitions are crucial since the Knowledge-soundness error bounds of the Σ-sle-compilers from Chapter 3, show that for a n-special sound protocol, the knowledge-soundness error κ_i and the amount of parallel repetitions t_i depend heavily on each other. To achieve a low knowledge-error κ_i, one needs to chose a rather large t_i, which however will increase the runtime of the compiler S.

In the following analysis we present the efficiency and security bounds for $C(\Pi^{\mu})$ when using one of the Σ-sle-compilers from Chapter 3 as the underlying compiler for C. To do that — for Fischlin's, Pass's and Kondi-Shelat's compiler— we first analyse the Σ-sle-compilers themselves to understand how one needs to choose t_i in order to reach the knowledge-soundness error $\kappa_i = 2^{-\lambda_i}$. Then we use these findings and apply them to the analysis of our multi-round compiler C.

4.5.2. Fischlin

For Fischlin's Σ-compiler, the knowledge-soundness error is $\kappa_{\text{Fischlin}} \leq q^{(i)}_{\text{RO},\mathcal{A}} \cdot \left(\frac{n-1}{2^l}\right)^{t_i} \leq \kappa_{\text{Target}}$ with l being the amount of leading zeros of the hash value. Solving it for t_i yields:

$$
\begin{aligned}
& q^{(i)}_{\text{RO},\mathcal{A}} \cdot \left(\frac{n-1}{2^l}\right)^{t_i} \leq \kappa_{\text{Target}} \\
\iff & \log_2(q^{(i)}_{\text{RO},\mathcal{A}}) + \log_2\left[\left(\frac{n-1}{2^l}\right)^{t_i}\right] \leq \log_2(\kappa_{\text{Target}}) \\
\iff & t_i \cdot \log_2\left(\frac{n-1}{2^l}\right) \leq \log_2(\kappa_{\text{Target}}) - \log_2(q^{(i)}_{\text{RO},\mathcal{A}}) \\
\iff & t_i \cdot (\log_2(n-1) - l) \leq \log_2(\kappa_{\text{Target}}) - \log_2(q^{(i)}_{\text{RO},\mathcal{A}}) \\
\iff & t_i \geq \frac{\log_2(\kappa_{\text{Target}}) - \log_2(q^{(i)}_{\text{RO},\mathcal{A}})}{\log_2(n-1) - l} = \frac{\log_2(q^{(i)}_{\text{RO},\mathcal{A}}) - \log_2(\kappa_{\text{Target}})}{l - \log_2(n-1)}
\end{aligned}
$$

Note that the inequality sign flips in the final step because the term $(\log_2(n-1) - l)$ is strictly negative. This is the case since for the proof-of-work to be difficult, we require $2^l > n-1$, which implies $l > \log_2(n-1)$. If we want λ_i bits of security, then $\kappa_{\text{Target}} = 2^{-\lambda_i}$ and therefore:

$$
t_i \geq \frac{\log_2(q^{(i)}_{\text{RO},\mathcal{A}}) - (-\lambda_i)}{l - \log_2(n-1)} = \frac{\log_2(q^{(i)}_{\text{RO},\mathcal{A}}) + \lambda_i}{l - \log_2(n-1)} \tag{4.4}
$$

- **Branching Factor n:** In most protocols n does not depend on the security parameter λ_i, so $n \in O(1)$. As n grows, the effective bits of security gained per repetition decreases forcing t_i to increase.
- **Random Oracle Queries $q^{(i)}_{\text{RO},\mathcal{A}}$:** Since we model $\mathcal{A}$ as a PPT algorithm, his query-budget $q^{(i)}_{\text{RO},\mathcal{A}}$ is polynomial in λ_i. Therefore $\log_2(q^{(i)}_{\text{RO},\mathcal{A}}) \in O(\log(\lambda_i))$, which means that we can ignore it in the asymptotic analysis.
- **Proof of Work Difficulty l:** We use $l \in O(\log(\lambda_i))$ which is analogue to what Fischlin required in his analysis [Fis05, Construction 1].
- **Parallel Repetitions t_i:** From the previous items, where we established $n \in O(1)$ and $l \in O(\log(\lambda_i))$, we can conclude $t_i \in O(\frac{\lambda_i}{\log(\lambda_i)})$, which can be upper bounded as $O(\lambda_i) = O(\lambda + \mu)$.

In the following we consider $n \in \mathcal{O}(1)$ and $l \in \mathcal{O}(\log(\lambda_i))$ which is analogous to what Fischlin required in his analysis [Fis05, Construction 1].

4.5.2.1. Efficiency

Now, we will use the findings from above to apply them to our modular, multi-round compiler C and analyse the efficiency of $C(\Pi^\mu)$.

Prover runtime: The prover's runtime is the most restrictive bottleneck in the Fischlin compilation of a multi-round protocol. The runtime is mostly influenced by $q^{(i)}_{\mathrm{RO},P}$ where $q^{(i)}_{\mathrm{RO},P}$ is the maximum amount of random oracle queries a prover has to produce a single proof on level i. Using the prover runtime function established in Section 4.2.2, the final runtime of the non-interactive prover is calculated recursively like this:

$$
\begin{aligned}
T_{\mu-1} &= q^{\mu}_{\mathrm{RO},P} \cdot T_{\mu} \\
T_{\mu-2} &= q^{\mu-1}_{\mathrm{RO},P} \cdot T_{\mu-1} = q^{\mu-1}_{\mathrm{RO},P} \cdot (q^{\mu}_{\mathrm{RO},P} \cdot T_{\mu}) \\
T_{\mu-3} &= q^{\mu-2}_{\mathrm{RO},P} \cdot T_{\mu-2} = \ldots\ldots
\end{aligned}
$$

leading to a runtime of P_{ni} of:

$$
T_{P_{\mathsf{ni}}} \approx \left(\prod_{i=1}^{\mu} q^{(i)}_{\mathrm{RO},P} \right) \cdot T_{\Pi^\mu} \tag{4.5}
$$

To ensure that the protocol runs correctly, we require $q^{(i)}_{\mathrm{RO},P} \in \mathcal{O}(\lambda_i^c)$, so for it to be polynomial.

Given that the query bound $q^{(i)}_{\mathrm{RO},P}$ at each level is polynomial, we must constrain the round complexity μ to ensure that the cumulative product $\prod_{i=1}^{\mu} q^{(i)}_{\mathrm{RO},P}$ remains polynomially bounded. By representing the polynomial upper bound of $q^{(i)}_{\mathrm{RO},P}$ as $a \cdot \lambda_i^b$ (where a and b are constants), we can evaluate the full product across all rounds:

$$
\prod_{i=1}^{\mu} \left(a \cdot \lambda_i^b \right) \in \mathcal{O}\left[\left(a \cdot \lambda_i^b \right)^{\mu} \right]
$$

To prevent this term from growing superpolynomially, μ must be restricted to a constant d. Substituting $\mu = d$ yields:

$$
\left(a \cdot \lambda_i^b \right)^{d} = a^d \cdot \lambda_i^{b \cdot d}
$$

Because b and d are both constants, this confirms that the overall runtime remains polynomial when $\mu \in O(1)$. Conversely, if the round complexity were to scale logarithmically (or higher), the non-interactive prover's runtime $T_{P_{\text{ni}}}$ would suffer from a superpolynomial blowup and would no longer be efficient.

Since we bound $\mu \in O(1)$, we can also make a new statement about λ_i:

$$\lambda_i \geq \lambda + O(\mu) = \lambda + O(1) \in O(\lambda)$$

Verifier runtime: The verifier's runtime is consistent across all compiler variants:

$$T_{V_{\text{ni}}} \approx T_{\Pi^\mu} \cdot \prod_{i=1}^{\mu} t_i + \sum_{i=1}^{\mu} \left[\left(\prod_{j=1}^{i} t_j \right) \cdot T_S \right] \tag{4.6}$$

The verifier is efficient relative to the prover because it does not perform the exhaustive search. However, the runtime still scales linearly with the total number of leaves in the recursion tree ($\prod_{i=1}^{\mu} t_i$). Since we need a constant μ, this is acceptable, because $t_i \in O(\lambda_i)$.

Communication Efficiency When using the Fischlin compiler as the underlying sle-compiler, we get the following communication complexity for $C(\Pi^\mu)$. This general formula is derived from the formula for 3-move Σ protocols established in Section 4.2.4. To help the reader understand how this general function is built, we have made an example in Section A.4.

$$W_{C(\Pi^\mu)} = W_{\pi_1} = \left[\sum_{i=1}^{\mu} \left(\prod_{j=1}^{i} t_j \right) (W_{a_i} + W_{c_i}) \right] + \left(\prod_{j=1}^{\mu} t_j \right) W_{a_{\mu+1}}$$

We know that $W_{a_i/c_i} \in O(\lambda)$ and to achieve polynomial communication efficiency, $\prod_{j=1}^{\mu} t_j$ needs to be strictly polynomial. If we use $\mu \in O(1)$, as already established before, this is the case.

4.5.2.2. Completeness

Before we start with the analysis of the completeness error, we prove the following helping lemma which we need later on:

Lemma 22. *Let* $\mathsf{negl}(\lambda)$ *be a negligible function and let* $p(\lambda)$ *be a polynomial. Then the function*

$$f(\lambda) := 1 - \big(1 - \mathsf{negl}(\lambda)\big)^{p(\lambda)}$$

is negligible.

Proof. Let $\mathsf{negl}(\lambda)$. Then

$$f(\lambda) = 1 - (1 - \mathsf{negl}(\lambda))^{p(\lambda)}.$$

Using Bernoulli's inequality, we obtain

$$(1 - \mathsf{negl}(\lambda))^{p(\lambda)} \geq 1 - p(\lambda) \cdot \mathsf{negl}(\lambda),$$

and therefore

$$f(\lambda) = 1 - (1 - \mathsf{negl}(\lambda))^{p(\lambda)} \leq p(\lambda) \cdot \mathsf{negl}(\lambda)$$

Since $\mathsf{negl}(\lambda)$ is negligible and the product of a negligible function with a polynomial is again negligible, it follows that $f(\lambda)$ is negligible. □

The completeness error $\epsilon_{C(\Pi^\mu)}$ of $C(\Pi^\mu)$ with Π^μ being a $(2\mu + 1)$-move protocol with completeness error ϵ_{Π^μ} and C using Fischlin's compiler as the underlying sle-compiler is:

$$\epsilon_{C(\Pi^\mu)}(\epsilon_S^{\text{Fischlin}}, (t_1, \cdots, t_\mu)) = 1 - \left[(1 - \epsilon_{\Pi^\mu})^{\prod_{i=1}^{\mu} t_i} \cdot (1 - \epsilon_S^{\text{Fischlin}})^{\sum_{i=1}^{\mu} \prod_{j=1}^{i} t_j}\right]$$

For $\epsilon_{C(\Pi^\mu)}$ to stay negligible, we need the exponents of $(1 - \epsilon_S^{\text{Fischlin}})$ and $(1 - \epsilon_{\Pi^\mu})$ to be polynomial as proven in Lemma 22. Since the analysis above requires $\mu \in O(1)$ and $t \in O(\lambda)$ already, using these bounds results in the exponents being polynomial and the completeness-error being negligible.

The preceding analysis illustrates one of the inherent trade-offs in parameter selection. Optimizing for prover efficiency by restricting the query budget $q_{\text{RO},P}^{(i)}$ inherently increases the completeness error. This tension arises because the single-round completeness error ϵ_S is heavily dependent on the number of random oracle queries the prover makes, as established in Section 4.1.1.

4.5.2.3. Knowledge Soundness

For the knowledge-soundness, the same analysis holds for all three compilers, so we will introduce it here and use it later on in the same way.

Since we already chose the knowledge-soundness error κ_i of the intermediate Σ-protocols based on the knowledge-soundness error of the fully compiled protocol $C(\Pi^\mu)$, the security of the protocol is guaranteed and we only need to ensure that the extractor runs in polynomial time.

Runtime of the Extractor As proven in Section 4.4.1.2, the runtime of the Extractor is

$$t_{\mathcal{E}_{C(\Pi^\mu)}} = \sum_{i=1}^{\mu} \left(\left[\prod_{j=1}^{i-1} n_j \right] \cdot t_{\mathcal{E}_{S[n_i]}} \right)$$

We now analyse how the parameters μ, n_i, and $t_{\mathcal{E}[n_i]}$ can influence the runtime. First of all it is important to clarify that we require $t_{\mathcal{E}_{C(\Pi^\mu)}}$ to be be polynomial. Since we demand that $t_{\mathcal{E}[n_i]}$ runs in polynomial time, that is not a factor in making the overall runtime superpolynomial. We therefore focus on μ and n_j.

Since we are working in the O-notation, we only need to take the biggest factor of the sum into account:

$$\prod_{j=1}^{\mu-1} n_j \in O(n^\mu)$$

As established earlier, we require $n_j \in O(1)$ and $\mu \in O(1)$ resulting in $t_{\mathcal{E}(\Pi^\mu)}$ staying polynomially bounded.

Finally we conclude the Analysis for the Fischlin Compiler. Using $\mu, n_i \in O(1)$, $l \in O(\log(\lambda))$ and $t_i \in O(\lambda)$, enables us to use the compiler C efficiently and securely. While the communication complexity is very good, the completeness-error and prover-runtime are rather large.

4.5.3. Pass

From Section A.1 we know the knowledge-soundness error for the Pass compiler. If we set κ_{target} as the target soundness error in each recursive proof, we get:

$$q^{(i)}_{\text{RO},\mathcal{A}} \cdot \left(\frac{n-1}{m} \right)^{t_i} + \text{Adv}^{\text{posbind}}_{\mathcal{VC},q^{(i)}_{\text{RO},\mathcal{A}},A}(\lambda) + \text{Adv}^{\text{ext}}_{\mathcal{VC},q^{(i)}_{\text{RO},\mathcal{A}},B}(\lambda) \leq \kappa_{\text{target}}$$

To isolate t_i, we must first subtract the negligible advantage terms from our target soundness error:

$$q^{(i)}_{\mathrm{RO},\mathcal{A}} \cdot \left(\frac{n-1}{m}\right)^{t_i} \leq \kappa_{\mathrm{target}} - \mathrm{Adv}^{\mathrm{posbind}}_{\mathcal{VC},q^{(i)}_{\mathrm{RO},\mathcal{A}},A}(\lambda) - \mathrm{Adv}^{\mathrm{ext}}_{\mathcal{VC},q^{(i)}_{\mathrm{RO},\mathcal{A}},B}(\lambda)$$

In the following we use $\tilde{u} = \kappa_{\mathrm{target}} - \mathrm{Adv}^{\mathrm{posbind}}_{\mathcal{VC},q^{(i)}_{\mathrm{RO},\mathcal{A}},A}(\lambda) - \mathrm{Adv}^{\mathrm{ext}}_{\mathcal{VC},q^{(i)}_{\mathrm{RO},\mathcal{A}},B}(\lambda)$

$$\left(\frac{n-1}{m}\right)^{t_i} \leq \frac{\tilde{u}}{q^{(i)}_{\mathrm{RO},\mathcal{A}}}$$

$$t_i \cdot \log_2\left(\frac{n-1}{m}\right) \leq \log_2(\tilde{u}) - \log_2(q^{(i)}_{\mathrm{RO},\mathcal{A}})$$

$$t_i \cdot (\log_2(n-1) - \log_2(m)) \leq \log_2(\tilde{u}) - \log_2(q^{(i)}_{\mathrm{RO},\mathcal{A}})$$

Since $n - 1 < m$, the term $(\log_2(n-1) - \log_2(m))$ is strictly negative. Dividing both sides by this term flips the inequality sign:

$$t_i \geq \frac{\log_2(\tilde{u}) - \log_2(q^{(i)}_{\mathrm{RO},\mathcal{A}})}{\log_2(n-1) - \log_2(m)}$$

$$t_i \geq \frac{\log_2(q^{(i)}_{\mathrm{RO},\mathcal{A}}) - \log_2(\tilde{u})}{\log_2(m) - \log_2(n-1)}$$

In the following, we ignore the attacker's advantages against the commitment scheme. This is justified when the commitment scheme is instantiated with parameters strong enough that its security loss is negligible compared to the other error terms in the analysis (e.g. $\mathrm{Adv}^{\mathrm{posbind}}_{\mathcal{VC},q^{(i)}_{\mathrm{RO},\mathcal{A}},A}(\lambda), \mathrm{Adv}^{\mathrm{ext}}_{\mathcal{VC},q^{(i)}_{\mathrm{RO},\mathcal{A}},B}(\lambda) \leq (\kappa_{\mathrm{target}})^2$) or if the commitment scheme is perfectly position-binding and perfectly straight-line extractable.

Therefore with $\kappa_{\mathrm{target}} = 2^{-\lambda_i}$ we get:

$$t_i \geq \frac{\log_2(q^{(i)}_{\mathrm{RO},\mathcal{A}}) + \lambda_i}{\log_2(m) - \log_2(n-1)}$$

If we bound $n \in O(1)$ and $m \in O(n)$, then $\log_2(m) - \log_2(n-1) \in O(1)$. And since $q^{(i)}_{\mathrm{RO},\mathcal{A}}$ is polynomial we conclude that $t_i \in O(\lambda_i)$.

4.5.3.1. Efficiency

Now we look at the efficiency of $C(\Pi^\mu)$ when using the Pass compiler as the underlying sle-compiler.

Prover Runtime:

$$T_{P_{\text{ni}}} \approx \left(\prod_{i=1}^{\mu} t_i \cdot m_i \right) \cdot T_{\Pi^\mu} \tag{4.7}$$

Since T_{Π^μ} is polynomial per requirement, we need to ensure that $\prod_{i=1}^{\mu} t_i \cdot m_i$ stays polynomial as well. Since we know that $m, n \in O(1)$, and $t_i \in O(\lambda_i)$ we need to choose $\mu \in O(1)$ for the prover runtime to be polynomial.

Verifier Runtime: Since the Verifier Runtime is the same between all 3 Compilers and is not an important factor in selecting the parameters, we will not go into detail here and refer to our analysis of the Fischlin Compiler.

Communication Efficiency The communication Efficiency when using the Pass compiler is:

$$W_{C(\Pi^\mu)} = \sum_{i=1}^{\mu} \left(\prod_{j=1}^{i} t_j \right) (W_{a_i} + W_{c_i}) + \left(\prod_{i=1}^{\mu} t_i \right) W_{a_{\mu+1}} + \sum_{i=1}^{\mu} \left(\prod_{j=1}^{i} t_j \right) W_{Com_i}$$

Where W_{Com_i} is the communication size of the commitment on level i. We define $Com(x)$ as a function that, given an input x, returns the size of a commitment to that input. Therefore:

$$W_{Com_i} = Com(m_i \cdot (W_{a_i} + W_{c_i} + W_{\pi_{i+1}}))$$

With the above chosen parameters ($\mu \in O(1), t \in O(\lambda)$), the communication complexity of $C(\Pi^\mu)$ is polynomial. However, compared to the communication complexity of the compiled protocol with Fischlin or Kondi-Shelat as the underlying compiler, the communication complexity is significantly higher due to the commitments which have to be sent.

4.5.3.2. Completeness

When using the Pass compiler or the Kondi-Shelat compiler, the completeness-error is better compared to Fischlin's compiler:

$$\epsilon_{C(\Pi^\mu)}(t_1, \cdots, t_\mu) = 1 - \left[(1 - \epsilon_{\Pi^\mu})^{\prod_{i=1}^{\mu} t_i} \right]$$

For the completeness error to remain negligible, the exponent $\prod_{i=1}^{\mu} t_i$ must be bounded by a polynomial in the security parameter λ as proven in Lemma 22. Because ϵ_{Π^μ} is negligible, raising $(1 - \epsilon_{\Pi^\mu})$ to a polynomial power keeps the result extremely close to 1, ensuring the overall error stays negligible. As we analysed above, the repetition parameter $t_i \in O(\lambda_i)$ and therefore to keep t^μ polynomially bounded, the round complexity μ must be strictly constant.

4.5.3.3. Knowledge Soundness

The knowledge-soundness error $\kappa_{C(\Pi^\mu)}$ is negligible if we use the parameters which we introduced in the previous paragraphs. Furthermore, the extractor works in polynomial time. We will not go into detail why, because the analysis is equal to what we did above.

Using the Pass compiler as the underlying Σ-sle-compiler results in roughly the same asymptotic bounds as when using the Fischlin compiler. However, since $t_i \cdot m_i \ll q^{(i)}_{\mathrm{RO},P}$ the runtime of the prover is much better when using Pass as the underlying compiler compared to Fischlin or Kondi-Shelat.

4.5.4. Kondi-Shelat

As established in Section A.3, the knowledge-soundness error for the Kondi-Shelat compiler is bounded by:

$$q^{(i)}_{\mathrm{RO},\mathcal{A}} \cdot \left(\binom{n-1}{r} \frac{1}{2^{l(r-1)}} \right)^{t_i} \leq \kappa_{Target}$$

If we require λ_i bits of security we set $\kappa_{Target} = 2^{-\lambda_i}$:

$$\begin{aligned}
& q^{(i)}_{\mathrm{RO},\mathcal{A}} \cdot \left(\binom{n-1}{r} \frac{1}{2^{l(r-1)}} \right)^{t_i} \leq 2^{-\lambda_i} \\
\iff & \log_2(q^{(i)}_{\mathrm{RO},\mathcal{A}}) + \log_2 \left[\left(\binom{n-1}{r} 2^{-l(r-1)} \right)^{t_i} \right] \leq -\lambda_i \\
\iff & t_i \cdot \left(\log_2 \binom{n-1}{r} - l(r-1) \right) \leq -\lambda_i - \log_2(q^{(i)}_{\mathrm{RO},\mathcal{A}}) \\
\iff & t_i \geq \frac{-\lambda_i - \log_2(q^{(i)}_{\mathrm{RO},\mathcal{A}})}{\log_2 \binom{n-1}{r} - l(r-1)} = \frac{\lambda_i + \log_2(q^{(i)}_{\mathrm{RO},\mathcal{A}})}{l(r-1) - \log_2 \binom{n-1}{r}}
\end{aligned}$$

In this equation, n (the branching factor) and r (the collision target) are protocol constants and therefore $O(1)$, and since the adversary's query budget $q_{\mathrm{RO},\mathcal{A}}$ is polynomial, $\log_2(q^{(i)}_{\mathrm{RO},\mathcal{A}})$ is asymptotically logarithmic. The defining variable is l, which represents the bit-length of the hash output where the prover must find collisions.

To ensure that the honest prover runs in polynomial time, the difficulty of finding these collisions must be bounded by a polynomial in λ_i. This strictly requires that the hash length l scales logarithmically, meaning $l \in O(\log \lambda_i)$. Substituting this back into our equation for t_i, we find:

$$t_i \in O\left(\frac{\lambda_i + \log(\lambda_i)}{\log \lambda_i}\right) = O(\lambda_i)$$

Assuming the round complexity μ is constant (which we will prove is necessary below), $\lambda_i \approx \lambda$. Therefore, the number of parallel repetitions t_i scales almost linearly with the overall security parameter λ.

4.5.4.1. Completeness

The completeness error for the Kondi-Shelat compiler is:

$$\epsilon_{C(\Pi^\mu)}((t_1, \ldots, t_\mu), (r_1, \ldots, r_\mu)) = 1 - \left[(1 - \epsilon_{\Pi^\mu})^{\prod_{i=1}^{\mu} r_i \cdot t_i}\right]$$

Because the base interactive protocol's completeness error ϵ_{Π^μ} is negligible, the exponent $\prod_{i=1}^{\mu} r_i \cdot t_i$ must be bounded by a polynomial in the security parameter λ as proven in Lemma 22.

Since r_i is a constant and $t_i \in O(\lambda)$, the exponent grows as $O(\lambda^\mu)$. To keep this term polynomially bounded, the round complexity μ must be strictly constant meaning: $\mu \in O(1)$.

4.5.4.2. Efficiency

Prover Runtime: As for the Fischlin compiler, the prover's runtime is the most restrictive bottleneck in the execution of $C(\Pi^\mu)$. The runtime is heavily influenced by the maximum number of random oracle queries allowed per proof $q^{(i)}_{\mathrm{RO},P}$:

$$T_{P_{\mathrm{ni}}} \approx \left(\prod_{i=1}^{\mu} q^{(i)}_{\mathrm{RO},P}\right) \cdot T_{\Pi^\mu}$$

The argument and analysis of the formula stays the same as for the Fischlin compiler.

Again, to ensure the overall runtime stays polynomial, we are strictly forced to bound the number of rounds to a constant, $\mu \in O(1)$. If μ were logarithmic or linear, the prover's runtime would become superpolynomial.

Verifier Runtime: Since the Verifier Runtime is the same between all 3 Compilers and is not an important factor in selecting the parameters, we will not go into detail here and refer to our analysis of the Fischlin Compiler.

Communication Complexity When using the Kondi-Shelat compiler as the underlying sle-compiler, we get the following communication complexity:

$$W_{C(\Pi^\mu)} = W_{\pi_1} = \sum_{i=1}^{\mu} \left(\prod_{j=1}^{i} r_j t_j \right) (W_{a_i} + W_{c_i}) + \left(\prod_{j=1}^{\mu} r_j t_j \right) W_{a_{\mu+1}}$$

Since $W_{a_i/c_i} \in O(\lambda)$, the multiplicative term $\prod_{j=1}^{\mu} r_j t_j$ needs to stay polynomial. Because r_j is a constant and $t_j \in O(\lambda)$, if we consider $\mu \in O(1)$, as already established before, the communication efficiency stays polynomial.

4.5.4.3. Knowledge Soundness

The runtime of the tree extractor $\mathcal{E}_{C(\Pi^\mu)}$ depends on the total number of extracted transcripts:

$$t\mathcal{E}_{C(\Pi^\mu)} = \sum_{i=1}^{\mu} \left(\left[\prod_{j=1}^{i-1} n_j \right] \cdot t\mathcal{E}_{\lfloor n_i \rfloor} \right)$$

Because the underlying protocol's branching factor n_i is constant, and we have established that $\mu \in O(1)$ is required for efficiency, the product $\prod_{j=1}^{\mu-1} n_j$ remains a constant. Therefore, the extractor is guaranteed to run in strict polynomial time.

4.5.5. A Possibility for Optimization: Hybrid Compilation

Throughout our analysis, a strict trade-off between the available Σ-protocol compilers becomes evident. Applying the Fischlin compiler across all μ rounds yields a good bound on communication complexity, but results in an exponential blowup in the prover's runtime, scaling with $O(q_{\mathrm{RO},P}^{(i)})$. Conversely, the Pass compiler offers a vastly superior

prover runtime because $t_i \cdot m_i \ll q_{\mathrm{RO},P}^{(i)}$, but it suffers from severe communication overhead due to the nested transmission of commitment trees.

Because our compiler C is strictly modular and processes the interactive protocol layer-by-layer, we are not forced to use the same underlying compiler S for every round. We can significantly optimize the transformation by utilizing a hybrid compilation strategy.

By applying the Pass compiler to the outermost rounds (e.g., rounds 1 and 2) and switching to the Fischlin compiler for the innermost rounds, we can leverage the strengths of both compilers while effectively neutralizing their worst bottlenecks.

Example: A 4-Round Hybrid Compilation Consider a protocol with $\mu = 4$ rounds.

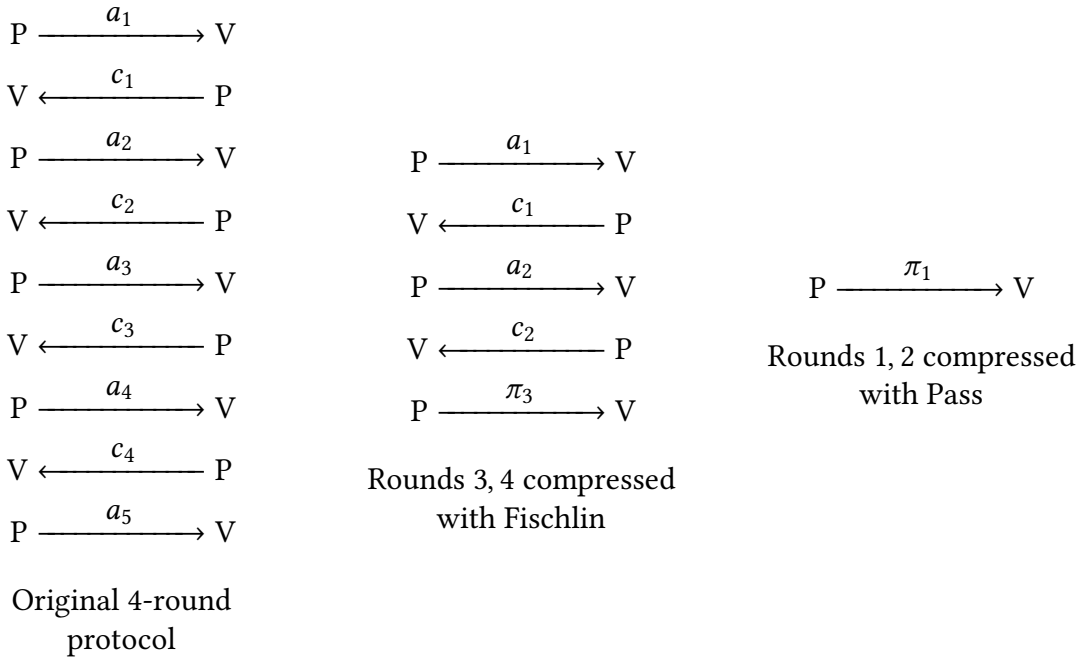


Figure 4.3.: Hybrid compilation strategy for a 4-round protocol

- **Pure Fischlin:** The prover's runtime scales with the product of the random oracle queries at each level: $\prod_{i=1}^{4} q_{\mathrm{RO},P}^{(i)} = (q_{\mathrm{RO},P}^{(i)})^4$. This makes the protocol computationally infeasible.
- **Pure Pass:** The prover's runtime is manageable, but the communication complexity requires sending commitments of commitments nested 4 levels deep, resulting in a massive final proof size for π_1.

- **The Hybrid Approach:** We apply the Pass compiler to compress the top, outermost rounds ($i = 1$ and $i = 2$) and the Fischlin compiler to compress the bottom, innermost rounds ($i = 3$ and $i = 4$).

Impact on Runtime: By using the Pass compiler for the first two rounds, the prover avoids the exhaustive proof-of-work search at the top of the recursion tree. In round 1, the prover now only needs to compute the underlying proof π_2 exactly $t_1 \cdot m_1$ times. For round 2, the prover computes π_3 exactly $t_1 \cdot m_1 \cdot t_2 \cdot m_2$ times. This reduces the exponential blowup severely because the exhaustive Fischlin search is confined strictly to the lower rounds ($i = 3, 4$). The hybrid prover runtime is therefore vastly superior to the cost of only using Fischlin's compiler.

Impact on Communication Complexity: While the top-level Pass compilations generate cut-and-choose trees, the branches of these trees are populated by the proofs from the lower rounds (π_3, π_4). Because we switch to the Fischlin compiler for these inner rounds, we ensure that these proofs remain compact (consisting only of the transcripts that satisfied the hash condition, completely devoid of heavy commitment trees). Since these inner proofs are replicated across all the opened branches of the outer Pass compiler, keeping their individual size minimal is crucial. The hybrid approach prevents the final proof π_1 from exploding in size, preserving an efficient communication complexity while maintaining a good prover runtime.

4.5.6. Comparison with the Compilers of Rotem and Tessaro

The main difference between the two compilers Rotem and Tessaro [RT25] introduced and our modular construction lies in the *achievable round complexity*. Rotem and Tessaro develop two monolithic compilers for multi-round protocols and obtain a clear trade-off between proof size and the number of rounds that can still be handled efficiently. Their first compiler, which can be viewed as the multi-round analogue of Fischlin's transformation, produces the more communication-efficient proofs, but it only remains efficient when the underlying protocol has round complexity at most [RT25, pg. 98]

$$O\left(\frac{\log \lambda}{\log \log \lambda}\right)$$

Their second compiler, which generalizes Pass's transformation to the multi-round setting, supports a larger class of protocols and remains efficient for protocols with [RT25, pg. 99]

$$O(\log \lambda)$$

rounds. This improved round scalability comes at the cost of a larger proof size, since the proof must additionally contain openings of a committed transcript tree.

In contrast, our modular construction developed in this thesis is currently significantly more restricted with respect to round complexity, only allowing a constant number of rounds. From the perspective of achievable round complexity, the asymptotic ordering is therefore clear: Rotem and Tessaro's second compiler is the strongest, as it supports $O(\log \lambda)$ rounds; their first compiler is slightly more restrictive, but still supports the super-constant regime $\Theta(\log \lambda / \log \log \lambda)$; and our modular construction of this thesis remains limited to the constant-round setting.

This gap in round complexity also explains the broader performance difference between the approaches. Rotem and Tessaro avoid the recursive blowup of repetition parameters by analyzing the multi-round protocol structure directly and compiling it in a monolithic way. As a result, they obtain substantially better asymptotic efficiency and security bounds for genuinely multi-round protocols. Our modular construction, by contrast, offers a conceptually clean layer-by-layer reduction using an underlying three-move sle compiler, but this modularity is purchased at the price of worse asymptotic bounds. Accordingly, the modular approach should primarily be viewed as a structurally elegant framework, whereas the constructions of Rotem and Tessaro are asymptotically stronger—especially with respect to the central parameter of achievable round complexity.

Furthermore Rotem and Tessaros compilers work with sHVZK protocols and do not require the input protocol to be 2SsHVZK as our modular compiler does.

This makes the compilers presented in [RT25] more usable than the modular compiler C presented in this thesis.

5. Example Protocol

In this chapter we present *Compressed Σ-protocols*, which were introduced in [AC20b]. We will start by introducing the concept and roughly explain how they work, and then we will prove that our compiler from Chapter 4 can be applied to Compressed Σ-protocols. Since [AC20a] contains the same information as [AC20b] but explains the different steps in more detail, we refer to [AC20a] even though it is not a published paper.

5.1. Compressed Σ-protocols

The final compressed Σ-protocol Π_c is obtained by composing the three protocols Π_0, Π_1, and Π_2. We briefly introduce each of them and explain how they interact to form Π_c. To keep the presentation accessible, we simplify some aspects and refer the reader to the original paper for the full mathematical details behind the protocol. Furthermore, we want to note, that the notation in the following might be confusing because while throughout the thesis we used $\mathbb{x}$ as the statement for a protocol, and $\mathbb{w}$ as the witness, in the following this will not be the case. This is because we want to use notation which is as close to the original notation in [AC20a] as possible, so that the readers who read the original paper will not get confused. Therefore in the following, we will explicitly state what the statement and witness for each protocol is.

5.1.1. The Basic Pivot (Π_0)

The first protocol, denoted as Π_0 [AC20a, Protocol 2], is a basic linear 3-move Σ-protocol used to prove that the prover knows a secret vector and secret scalar that perfectly open a public vector commitment P, meaning that $g^x h^y = P$. The statement for this protocol is (P, L, y) and the secret (x, y). Π_0 works as follows:

- Move 1: The prover generates a random vector r and scalar ρ, creates a commitment A, and sends A along with the linear form evaluation $t = L(r)$ to the verifier.

- Move 2: The verifier sends a uniformly random challenge c to the prover.
- Move 3: The prover computes and sends the masked responses $z = cx + r$ and $\phi = c\gamma + \rho$. The verifier then checks the validity of the commitment opening ($g^z h^\phi = AP^c$) and ensures the linear form evaluates correctly ($L(z) = cy + t$).

This protocol is perfectly complete, special honest-verifier zero-knowledge and unconditionally special sound. However, the communication cost is linear because the entire vector z has to be sent.

5.1.2. Reduction Protocol to Relation R_2 (Π_1)

Π_1 [AC20a, Protocol 3] serves as a 2-move argument of knowledge that reduces the relation proved in the final step of Π_0 (R_1) into a new relation (R_2).

The core idea is to fuse the two checks which have to be done by the verifier in Π_0 into one Pedersen vector commitment, for which one check suffices. This is achieved by introducing an additional generator k to the public parameters. The statement for this protocol is $(\hat{P}, \hat{L}, \hat{y})$ and the witness is $\hat{z}$.

- On input $\hat{P} = AP^c$, $\hat{L}(z, \phi) = L(z)$, $\hat{y} = cy + t$, $\hat{z} = (z, \phi)$, the verifier provides a random challenge c
- Then the prover sends the witness $\hat{z}$ and the verifier can check the newly combined relation $\hat{g}^{\hat{z}} k^{c\hat{L}(\hat{z})} = \hat{P} k^{c\hat{y}}$.

Note that this protocol does not satisfy zero-knowledge since the prover essentially sends the witness to the verifier.

5.1.3. Compressed Proof of Knowledge (Π_2)

Π_2 [AC20a, Protocol 4] applies a bulletproof-like compression mechanism to create a proof of knowledge for the linear relation R_2 with logarithmic rather than linear communication complexity.

It achieves that logarithmic communication complexity because the prover splits the witness into a left and a right half and sends commitments (A and B) to the cross-terms. The protocol continues halving the dimension of the witness vector in each iteration until the dimension is reduced to 2. One halving-Iteration looks as follows:

- The prover sends A (the commitment to the right half) and B (the commitment to the left half)

- The verifier responds with a challenge c.
- Both parties then compute a new set of generators $\mathbf{g}'$ and a combined commitment Q'.
- If the dimension is 2, the prover sends the combined response z'; otherwise, the protocol recurses on the halved dimension and $\mathbf{g}'$ and Q'.

5.1.4. The Compressed Pivot ($\Pi_c = \Pi_2 \circ \Pi_1 \circ \Pi_0$)

Π_c [AC20a, Protocol 5] is the direct composition of the previous three building blocks: $\Pi_c := \Pi_2 \circ \Pi_1 \circ \Pi_0$. The general idea is to execute Π_0, but instead of sending the last message which would be very large (linear communication complexity) we transform the Relation into one which can be used by bulletproofs in Π_1. Then in Π_2, the witness is shrunk down to achieve logarithmic communication complexity.

In simplified terms, the protocol works as follows:

- The execution begins with the prover sending the initial commitment elements from Protocol Π_0.
- The verifier then issues two random challenges (c_0, c_1) (one "for" Π_0 and the other one "for" Π_1), effectively executing the challenge phase of Π_0 and the reduction of Π_1 simultaneously.
- Finally, the prover and verifier engage in the recursive halving process defined in Π_2 until the witness is fully compressed.

5.2. Application of C on compressed Σ-protocols

To prove the applicability of C on Compressed Σ-protocols, we need to prove that they are 2sSHVZK, as well as $(n_1, \cdots, n_\mu)$-(strong)-special sound for an extraction Relation $\tilde{R}$.

Since we have to formally prove different properties of the protocol in the following, we use specific, mathematical details which are not explained above. So if the reader wants to understand the following section, we strongly advise to read Attema and Cramer's original paper, with all details. The notation will be equal to that in the paper in the following.

5.2.1. 2 Stage special Honest Verifier Zero Knowledge

In [AC20a, Theorem 3], the authors already prove that the Π_c is sHVZK. However, 2 Stage special Honest Verifier Zero Knowledge is a stronger definition, which we will prove in the following. We prove 2 Stage sHVZK by first proving that Π_c is a 2-stage protocol and then showing that it is actually 2SsHVZK.

Theorem 23 (Π_c is a 2-stage Protocol)**.** *Let Π_c be the Protocol explained in Section 5.1.4 and defined in [AC20a, Protocol 5]. We show that Π_c is a 2 Stage Protocol*

Proof. To prove that Π_c is a 2 Stage Protocol, we construct a prover algorithm P_1 which gets the statement and witness and another prover algorithm $P_2(x, pp, st_1, (c_0, c_1))$, which only gets the statement, the public parameters and the state of P_1, and needs to finish the protocol without the witness.

First Stage P_1:

1. The prover samples random blinding factors $r \leftarrow_R \mathbb{Z}_q^n$ and $\rho \leftarrow_R \mathbb{Z}_q$.
2. The prover computes $t = L(r)$ and the initial commitment $\hat{A} = g^r h^\rho$, and sends $\hat{A}, t$ to the verifier.
3. The verifier responds with the initial challenge $c = (c_0, c_1) \leftarrow_R \mathbb{Z}_q$.
4. Using the original witness (x, γ), the prover computes the intermediate masked responses $z = c_0 x + r$ and $\phi = c_0 \gamma + \rho$.
5. The prover constructs the combined intermediate witness $\hat{z} = (z, \phi)$.
6. The prover sets its state $st_1 = \hat{z}$ and discards the original witness (x, γ).

Second Stage P_2 (Witness-Independent): We now construct the second-stage prover $P_2((P, L, y), pp, st_1, (c_0, c_1))$.

1. P_2 uses the statement (P, L, y), the public parameters (g, h, k), the challenges c_0, c_1, and the initial message $\hat{A}, t$ to compute the targets for the recursive relation: $Q := \hat{A} P^{c_0} k^{c_1(c_0 y + t)}$ and $\tilde{L}(\hat{z}) := c_1 L(z)$.
2. P_2 retrieves the intermediate witness $\hat{z}$ from st_1.
3. To compute the next messages in the protocol (the cross-term commitments A and B), P_2 only needs to split $\hat{z}$ into its left and right halves, $\hat{z}_L$ and $\hat{z}_R$, and compute $A = \hat{g}_R^{\hat{z}_L} k^{\tilde{L}_R(\hat{z}_L)}$ and $B = \hat{g}_L^{\hat{z}_R} k^{\tilde{L}_L(\hat{z}_R)}$.
4. Upon receiving the next challenge c from the verifier, P_2 computes the folded vector $z' = \hat{z}_L + c\hat{z}_R$.

5. P_2 recursively repeats this halving process (Π_2) using the newly folded z' as the state for the next round until the dimension is reduced to 2, at which point it sends the final z'.

During the entire execution of Stage 2 (which encompasses the Π_1 reduction and Π_2 recursive compression), the prover P_2 relies strictly on the folded intermediate witness $\hat{z}$ (stored in st_1) and the verifier's challenges. The original witness (x, γ) is never referenced or required to compute any of the cross-terms or folded responses. Therefore, Π_c satisfies the strict definition of a 2-Stage Protocol. □

Theorem 24 (Π_c is 2SsHVZK)**.** *Let Π_c be the Protocol explained in Section 5.1.4 and defined in [AC20a, Protocol 5]. We show that Π_c is 2 Stage special Honest-Verifier Zero-Knowledge*

Proof. To prove that the compressed pivot Π_c is 2 Stage Special Honest-Verifier Zero-Knowledge, we need to show that the real and simulated transcripts (a_1, c, st_1) are computationally indistinguishable. We do that assuming that Π_c is not 2sSHVZK and then create a reduction which would break the sHVZK property of Π_0.

Let us explicitly map the variables of the first stage of Π_c to the underlying protocol Π_0. In [AC20a, Protocol 5], the first stage corresponds precisely to the execution of Π_0. The only meaningful difference is, that the verifier already sends the challenge c_1 for Π_1 which however, does not influence the 2SsHVZK property of Π_c, since it is only used in the second stage prover algorithm P_2. The initial message is $a_1 = (\hat{A}, t)$, the challenge is $c = (c_0, c_1)$, and the resulting intermediate state passed to the second stage is $st_1 = \hat{z} = (z, \phi)$.

Assume, for the sake of contradiction, that there exists an efficient polynomial-time adversary $\mathcal{A}$ that can distinguish between a real first-stage tuple (a_1, c, st_1) and a simulated first-stage tuple (a_1', c', st_1') with a non-negligible advantage ϵ.

We construct an adversary $\mathcal{B}$ against the sHVZK property of the basic pivot Π_0 as follows:

1. $\mathcal{B}$ receives a public input (P, L, y) and a randomly chosen challenge c_0.
2. $\mathcal{B}$ interacts with its challenger to receive a Π_0 transcript $(\hat{A}, t, c_0, z, \phi)$. The challenger generates this transcript either by running the real honest prover algorithm or by running the SHVZK simulator for Π_0.
3. $\mathcal{B}$ parses this transcript into the 2-stage tuple format required by $\mathcal{A}$: $a_1 = (\hat{A}, t)$, $c = (c_0, \{\})$, and $st_1 = (z, \phi)$.

4. $\mathcal{B}$ invokes the distinguisher $\mathcal{A}$ who chooses c_1 randomly and decides the tuple $(a_1, c = (c_0, c_1), st_1)$.
5. $\mathcal{A}$ outputs a bit b' indicating whether it believes the tuple is real ($b' = 0$) or simulated ($b' = 1$). $\mathcal{B}$ outputs the exact same bit b'.

Because the construction of Π_c dictates that its simulator simply runs the simulator for Π_0 to generate the first stage, the distribution of the tuples provided to $\mathcal{A}$ by $\mathcal{B}$ perfectly matches the environment $\mathcal{A}$ expects.

Therefore, if $\mathcal{A}$ can distinguish the first stage of Π_c with advantage ϵ, $\mathcal{B}$ can distinguish the transcript of Π_0 with the exact same advantage ϵ. However, per definition Π_0 is perfectly complete and special honest-verifier zero-knowledge. Consequently, no such efficient adversary $\mathcal{B}$ can exist.

This contradicts our initial assumption. Thus, the real and simulated transcripts of the first stage are indistinguishable, concluding the proof that Π_c is 2-Stage special Honest-Verifier Zero-Knowledge. □

5.2.2. Strong-special-soundness

In the following section we prove that the compressed pivot protocol Π_c is computationally strong special sound. Since Π_1 and the collision-cases of the proof may yield a non-trivial discrete logarithm relation instead of a witness for the original relation, we work with an extended extraction relation.

Relations used in the following: For the basic pivot relation R, we define

$$\widetilde{R} := R \ \vee \ \mathrm{DLOG}_{g,h},$$

where $\mathrm{DLOG}_{g,h}$ denotes the relation consisting of all non-zero pairs $(\alpha, \beta) \in \mathbb{Z}_q^2 \setminus \{(0,0)\}$ such that

$$g^\alpha h^\beta = 1.$$

For the relation

$$R_1 = \left\{((\hat{P}, \hat{L}, \hat{y}); \hat{z}) : \hat{P} = \hat{g}^{\hat{z}} \ \wedge \ \hat{y} = \hat{L}(\hat{z})\right\},$$

we define

$$\widetilde{R_1} := R_1 \ \vee \ \mathrm{DLOG}_{\hat{g},k},$$

where $\mathrm{DLOG}_{\hat{g},k}$ denotes the relation consisting of all non-zero tuples

$$(\alpha_1, \ldots, \alpha_m, \beta) \in \mathbb{Z}_q^{m+1} \setminus \{0\}$$

such that

$$\prod_{i=1}^{m} \hat{g}_i^{\alpha_i} \cdot k^{\beta} = 1.$$

Furthermore, for

$$R_2 = \left\{ ((Q, \widetilde{L}); \hat{z}) : Q = \hat{g}^{\hat{z}} k^{\widetilde{L}(\hat{z})} \right\},$$

we define:

$$\widetilde{R}_2 := R_2 \ \vee \ \mathsf{DLOG}_{\hat{g},k}.$$

Thus, throughout this section, "extracting" means outputting a witness for the currently relevant original relation or a non-trivial discrete logarithm relation among the public generators.

Frequent DLOG Break In the following, we will frequently run into the same formula, for which we argue, that it breaks the DLOG assumption. We prove this property here to leave it out in the following, simplifying the subsequent proofs.

Theorem 25. *Let* $(Q, \widetilde{L}; \hat{z}_1), (Q, \widetilde{L}; \hat{z}_2) \in R_2$ *with* $\hat{z}_1 \neq \hat{z}_2$. *Then*

$$\hat{g}^{\hat{z}_1 - \hat{z}_2} k^{\widetilde{L}(\hat{z}_1 - \hat{z}_2)} = 1.$$

In particular, $z_1 - z_2$ *is non-zero and therefore yields a witness for* $\mathsf{DLOG}_{\hat{g},k}$.

Proof. Since both witnesses open the same relation instance, we have

$$Q = \hat{g}^{\hat{z}_1} k^{\widetilde{L}(\hat{z}_1)} \qquad \text{and} \qquad Q = \hat{g}^{\hat{z}_2} k^{\widetilde{L}(\hat{z}_2)}.$$

Dividing the two equalities gives

$$\hat{g}^{\hat{z}_1 - \hat{z}_2} k^{\widetilde{L}(\hat{z}_1) - \widetilde{L}(\hat{z}_2)} = 1.$$

Since $\widetilde{L}$ is linear, this is equal to

$$\hat{g}^{\hat{z}_1 - \hat{z}_2} k^{\widetilde{L}(\hat{z}_1 - \hat{z}_2)} = 1.$$

Because $\hat{z}_1 \neq \hat{z}_2$, $z_1 - z_2$ is non-zero. □

We now prove strong-special-soundness for the three building blocks Π_0, Π_1 and Π_2, which enables us to prove strong-special-soundness for Π_c later on.

2-strong special soundness for Π_0.

Theorem 26. *Protocol Π_0 is 2-strong special sound for the extraction relation $\widetilde{R}_0$.*

Proof. Recall that an accepting transcript of Π_0 has the form

$$T = ((t, A), c, (z, \phi)).$$

We show that Π_0 is 2-strong special sound for $\widetilde{R}_0$.

Let $T_1 = ((t, A), c_1, (z_1, \phi_1))$ and $T_2 = ((t, A), c_2, (z_2, \phi_2))$ be two accepting transcripts forming a 2-strong-special-soundness tree.

If $c_1 \neq c_2$, then the ordinary special-soundness extractor for Π_0 already outputs a witness for R as explained in [AC20a].

To prove strong-special soundness we however need to also consider the case that $c_1 = c_2$ and $(z_1, \phi_1) \neq (z_2, \phi_2)$.

Since both transcripts are accepting, they satisfy the verification equation

$$g^{z_1} h^{\phi_1} = AP^c \qquad \text{and} \qquad g^{z_2} h^{\phi_2} = AP^c.$$

Dividing these two equalities yields

$$g^{z_1 - z_2} h^{\phi_1 - \phi_2} = 1.$$

Because $(z_1, \phi_1) \neq (z_2, \phi_2)$, the pair

$$(z_1 - z_2, \phi_1 - \phi_2) \neq (0, 0)$$

is a witness for $\mathrm{DLOG}_{g,h}$. Hence the extractor outputs a witness for $\widetilde{R}_0$.

□

2-strong special soundness for Π_1.

Theorem 27. *Protocol Π_1 is 2-strong special sound for the extraction relation $\widetilde{R}_1$.*

Proof. An accepting transcript of Π_1 has the form

$$T = (\emptyset, c, \hat{z}).$$

We show that Π_1 is 2-strong special sound for $\widetilde{R}_1$.

Let

$$T_1 = (\emptyset, c_1, \hat{z}_1) \qquad \text{and} \qquad T_2 = (\emptyset, c_2, \hat{z}_2)$$

be two accepting transcripts forming a 2-strong-special-soundness tree.

If $c_1 \neq c_2$, then the special-soundness argument from [AC20a] applies.

Assume now that $c_1 = c_2 =: c$. Since the tree is strong-special-sound, we must have

$$\hat{z}_1 \neq \hat{z}_2.$$

Both transcripts satisfy the verification equation

$$\hat{g}^{\hat{z}_i} k^{c\hat{L}(\hat{z}_i)} = \hat{P}\, k^{c\hat{y}} \qquad \text{for } i \in \{1, 2\}.$$

Dividing the two equalities gives

$$\hat{g}^{\hat{z}_1 - \hat{z}_2} k^{c(\hat{L}(\hat{z}_1) - \hat{L}(\hat{z}_2))} = 1.$$

Using linearity of $\hat{L}$, this can be rewritten as

$$\hat{g}^{\hat{z}_1 - \hat{z}_2} k^{c\hat{L}(\hat{z}_1 - \hat{z}_2)} = 1.$$

Since $\hat{z}_1 \neq \hat{z}_2$, the coefficient vector

$$(\hat{z}_1 - \hat{z}_2,\ c\hat{L}(\hat{z}_1 - \hat{z}_2))$$

is non-zero, and therefore constitutes a witness for $\mathsf{DLOG}_{\hat{g},k}$. Thus the extractor outputs a witness for $\widetilde{R}_1$.

□

$(3, \ldots, 3)$-strong special soundness for Π_2.

Theorem 28. *Protocol Π_2 is $(n_1, \ldots, n_\mu)$-strong special sound with $n_i = 3$ for the extraction relation $\widetilde{R}_2$.*

Proof. We prove the statement by induction on the recursion depth μ of Π_2.

Recall that Π_2 is a recursive proof system for the relation

$$R_2 = \{((Q, \widetilde{L}); \hat{z}) : Q = \hat{g}^{\hat{z}} k^{\widetilde{L}(\hat{z})}\}.$$

For any even dimension m and vector $g \in \mathbb{G}^m$ we define $g_L = (g_1, \cdots, g_{\frac{m}{2}})$ as its left half and $g_R = (g_{\frac{m}{2}+1}, \cdots, g_m)$ as its right half. The corresponding split of a linear form was introduced in [AC20a, Sec. 4.2], and we refer the reader there for details.

At each recursive step the prover sends two commitments (A, B), receives a challenge $c \in \mathbb{Z}_q$, and the protocol continues on the folded instance.

$$Q' := AQ^c B^{c^2}, \qquad \widetilde{L}' := c\widetilde{L}_L + \widetilde{L}_R, \qquad \hat{g}' := \hat{g}_L^c * \hat{g}_R,$$

with folded witness

$$\hat{z}' := \hat{z}_L + c\hat{z}_R.$$

Base case $\mu = 1$. In this case Π_2 is a 3-move protocol. Let

$$T_i = ((A, B), c_i, \hat{z}_i), \qquad i \in \{1, 2, 3\},$$

be the three accepting transcripts of a 3-strong-special-soundness tree.

Case 1: The challenges are pairwise distinct:

Then the standard special-soundness extractor of [AC20a] applies and outputs a witness for $R_2 \subseteq \tilde{R}_2$.

Case 2: Two or more challenges collide:

$$c_1 = c_2 =: c.$$

Because the transcript tree is strong-special-sound, the next prover messages must be different. In the base case, the next prover message is the final response:

$$\hat{z}_1 \neq \hat{z}_2.$$

Since both transcripts are accepting for the same first message (A, B) and the same challenge c, they define two different witnesses for the same folded instance

$$(Q', \widetilde{L}').$$

By Theorem 25, this yields a witness for $\mathsf{DLOG}_{\hat{g}',k}$, and therefore for $\mathsf{DLOG}_{\hat{g},k}$ as well. Hence the extractor outputs a witness for $\widetilde{R}_2$.

Induction Hypothesis: Assume that for recursion depth μ protocol Π_2 is $(n_1, \ldots, n_\mu)$-strong special sound with $n_i = 3$ for $\tilde{R}_2$.

Induction step $(\mu \to \mu + 1)$

Let T be a $(n_1, \ldots, n_\mu)$-strong special soundness tree for Π_2 with $n_i = 3$. We use R'_2 as the extraction relation for an $(n_2, \ldots, n_\mu)$-strong special soundness tree. At the root, the prover message is (A, B) and there are three outgoing branches.

Case 1: The three root challenges are pairwise distinct. Then, exactly as in the standard special-soundness proof, we obtain three accepting subtrees for three pairwise distinct challenges c_1, c_2, c_3, from which we can extract valid witnesses. If the extraction of the subtrees yield a DLOG break, that constitutes the witness for $\tilde{R}_2$. If however they yield a witness for R'_2, by applying the usual Vandermonde argument to the three extracted folded witnesses gives a witness for the parent relation instance $(Q, \widetilde{L})$. Hence the extractor outputs a witness for R_2.

Case 2: Two root challenges collide. Without loss of generality assume

$$c_1 = c_2 =: c.$$

By the definition of a strong-special-soundness tree, the next prover messages on these two branches must differ. Denote them by

$$(A'_1, B'_1) \neq (A'_2, B'_2).$$

Because the root challenge is the same on both branches, both children correspond to the same folded instance

$$Q' := AQ^c B^{c^2}, \qquad \widetilde{L}' := c\widetilde{L}_L + \widetilde{L}_R, \qquad \hat{g}' := \hat{g}_L^c * \hat{g}_R.$$

Now consider the two rooted subtrees below (A'_1, B'_1) and (A'_2, B'_2). Each of them is again a $(3, \ldots, 3)$-strong-special-soundness tree for the same subinstance $(Q', \widetilde{L}')$. By the induction hypothesis, applying the extractor to each subtree returns either:

1. a witness for R'_2 for the same instance $(Q', \widetilde{L}')$, or
2. a non-trivial witness for $\mathsf{DLOG}_{\hat{g}',k}$.

If either subtree already yields a witness for $\mathsf{DLOG}_{\hat{g}',k}$, then we are done.

So assume both subtrees yield witnesses

$$w_1, w_2$$

for the same relation instance $(Q', \widetilde{L}')$.

We claim that $w_1 \neq w_2$. Indeed, by Theorem 23, once the intermediate witness stored after the first stage is fixed, all later prover messages in Π_2 are deterministic functions of that witness and the public transcript. Therefore, if $w_1 = w_2$, then the first prover messages in the two recursive calls would also coincide, which would imply

$$(A'_1, B'_1) = (A'_2, B'_2),$$

contradicting the strong-special-soundness condition at the root. Hence

$$w_1 \neq w_2.$$

Since w_1 and w_2 are two distinct witnesses for the same instance $(Q', \widetilde{L}')$, Theorem 25 yields a witness for $\mathrm{DLOG}_{\hat{g}',k}$, and therefore for $\widetilde{R}_2$.

This completes the induction and proves the theorem. □

Strong special soundness of the compressed pivot. We can now combine the previous results.

Theorem 29. *Let*

$$\Pi_c := \Pi_2 \circ \Pi_1 \circ \Pi_0.$$

Then Π_c *is computationally* $(2, 2, 3, \ldots, 3)$*-strong special sound for the extraction relation*

$$\widetilde{R} := R \ \vee \ \mathrm{DLOG},$$

where DLOG *denotes the existence of a non-trivial discrete logarithm relation among the public generators of the protocol.*

Proof. The protocol Π_c is the sequential composition of the three building blocks Π_0, Π_1 and Π_2.

By Theorem 26, from a 2-strong-special-soundness tree for the Π_0 part we can extract either a witness for R or a witness for $\mathrm{DLOG}_{g,h}$.

By Theorem 27, from a 2-strong-special-soundness tree for the Π_1 part we can extract either a witness for R_1 or a witness for $\mathrm{DLOG}_{\hat{g},k}$.

By Theorem 28, from a $(3, \ldots, 3)$-strong-special-soundness tree for the Π_2 part we can extract either a witness for R_2 or a witness for $\mathrm{DLOG}_{\hat{g},k}$.

Therefore, from a $(2, 2, 3, \ldots, 3)$-strong-special-soundness tree for Π_c we can extract either a witness for the original relation R or a non-trivial discrete logarithm relation among the public generators. This is exactly the extraction relation $\widetilde{R}$. □

5.3. Example

The goal of this chapter is to demonstrate how the compiler C (Chapter 4) can be applied to compressed Σ-protocols. In the previous sections, we already showed that compressed Σ-protocols satisfy all required properties. We now turn from the theoretical analysis to a concrete instantiation and examine how the parameters must be chosen in practice. The goal of this example is to make the parameter choices explicit and to illustrate what the resulting efficiency looks like in practice. We use the Fischlin transformation as the underlying 3-move protocol compiler.

From Chapter 5, we know that compressed Σ-protocols satisfy $(2, 2, 3, \ldots, 3)$-strong-special-soundness. Hence, the corresponding special-soundness parameters are

$$n_1 = n_2 = 2 \qquad \text{and} \qquad n_i = 3 \text{ for all } i \geq 3.$$

We set the target security level to $\lambda = 128$, meaning that the knowledge-soundness error of the resulting, fully compiled, non-interactive proof should be at most 2^{-128}. As discussed in Section 4.5.1, the security parameter for each individual 3-move compilation can be calculated with $\lambda_i \geq \lambda + O(\mu)$. Since throughout this example we assume $\mu \in O(1)$, we may choose

$$\lambda_i = \lambda = 128$$

for every recursion level i.

To ensure that $\mu \in O(1)$, one may either restrict the number of recursive applications of Π_2 or impose an upper bound on the size of the input statement $\mathbb{x}$. We will not fix a specific choice here, since the purpose of this section is only to illustrate the parameter selection.

Furthermore, we bound the number of random-oracle queries an honest prover can make to construct a Fischlin proof π_i on level i by

$$q_{\mathrm{RO},P}^{(i)} = 2^{15}$$

and we bound the number of random-oracle queries an attacker can make to forge a proof π_i on level i by

$$q_{\mathrm{RO},\mathcal{A}}^{(i)} = 2^{30}$$

This bound is admittedly low and not entirely realistic; however, since this is only an example, it keeps the discussion easy to follow.

Instantiation with Fischlin Furthermore, we set the value of the Fischlin-specific parameter $l = 15$. By the bound derived in Equation (4.4), the number of parallel repetitions at recursion level i must satisfy

$$t_i \geq \frac{\log_2(q^{(i)}_{\mathrm{RO},\mathcal{A}}) + \lambda_i}{l - \log_2(n_i - 1)}.$$

First two levels. For the first two recursion levels, we have $n_1 = n_2 = 2$. Therefore,

$$t_1, t_2 \geq \frac{\log_2(2^{30}) + 128}{15 - \log_2(2 - 1)} = \frac{30 + 128}{15 - 0} = 10,53$$

Since the number of parallel repetitions must be an integer, we round up and obtain

$$t_1 = t_2 = 11$$

All subsequent levels. For every level $i \geq 3$, we have $n_i = 3$. Hence,

$$t_i \geq \frac{\log_2(2^{30}) + 128}{15 - \log_2(3 - 1)} = \frac{30 + 128}{15 - 1} \approx 11,286$$

Again rounding up to the next integer gives

$$t_i = 12 \qquad \text{for all } i \geq 3.$$

Implications for efficiency. We can now plug these concrete values into the efficiency bounds derived in Section 4.5.2.1.

First, the prover runtime grows exponentially in the number of rounds. More precisely, the prover runtime is dominated by the recursive search performed at each level and scales roughly like

$$T_{P_{\mathrm{ni}}} \approx \left(\prod_{i=1}^{\mu} q^{(i)}_{\mathrm{RO},P} \right) \cdot T_{\Pi_\mu}.$$

Under our simplifying assumption $q^{(i)}_{\mathrm{RO},P} = 2^{15}$ for all i, this becomes

$$T_{P_{\mathrm{ni}}} \approx (2^{15})^{\mu} \cdot T_{\Pi_\mu}.$$

This illustrates concretely the exponential blowup already identified in the general analysis and explains why the number of rounds μ must remain constant.

The verifier does not perform the same exhaustive search as the prover, but its work still grows with the total number of leaves in the recursion tree. In our example, this quantity is approximately

$$\prod_{i=1}^{\mu} t_i$$

For instance, if $\mu = 4$, then the verifier has to process

$$11 \cdot 11 \cdot 12 \cdot 12 = 17424$$

branches. While this is still much better than the prover's work, it shows that the verification cost can also become substantial once μ increases.

Finally, the communication complexity grows in the same recursive manner. At each level, the proof contains information for all proofs generated at deeper levels, so the total proof size grows rapidly.

In summary, this chapter shows that the modular compiler C can indeed be instantiated concretely on compressed Σ-protocols. However, it also highlights the central limitation of the modular approach: even when the required repetition numbers at each level are moderate, the recursive composition still causes an exponential increase in prover runtime, verifier runtime, and communication complexity.

6. Summary

This thesis explored the theoretical construction and analysis of a modular straight-line extractability (sle) compiler for multi-round Σ-protocols. While the recursive, layer-by-layer compilation approach offers a structurally elegant framework, our analysis reveals significant practical limitations that render the current construction virtually unusable for real-world applications.

The most critical bottleneck is the exponential blowup in both computational overhead and communication complexity. Because the required number of parallel repetitions, t_i, must be applied multiplicatively at each recursive level to maintain negligible knowledge soundness and completeness errors, the overall costs scale exponentially with the round complexity μ. Furthermore, the reliance on the 2-Stage Special Honest-Verifier Zero-Knowledge (2SsHVZK) property imposes a stringent cryptographic requirement. Since not all multi-round protocols naturally satisfy this strict condition, the general applicability of our compiler is restricted. Consequently, in its current form, the modular compiler serves primarily as a theoretical thought experiment in cryptography rather than a deployable tool.

However, this thesis opens several promising opportunities for future research to mitigate these limitations:

- **Tighter Error Analysis:** The current upper bounds used throughout this thesis are not perfectly tight. It would be interesting to lower these bounds as much as possible to see, whether a better round-complexity can be achieved.
- **Global Repetition Architecture:** The exponential blowup stems from amplifying security separately at every recursive level. A possible alternative would be to redesign the repetition strategy so that, instead of performing t_i repetitions within each layer, one uses only a fixed, smaller number of repetitions per layer and then repeats the resulting construction in parallel. This could potentially reduce the overall exponential growth to a more manageable polynomial overhead while still preserving straight-line extractability.

Ultimately, while the strictly modular compilation of multi-round protocols introduces prohibitive inefficiencies, the insights gained regarding transcript tree extraction and

recursive zero-knowledge simulation provide a valuable stepping stone for the future optimization of non-interactive arguments.

Bibliography

[AC20a] Thomas Attema and Ronald Cramer. *Compressed Σ-Protocol Theory and Practical Application to Plug Play Secure Algorithmics*. Cryptology ePrint Archive, Paper 2020/152. 2020. URL: https://eprint.iacr.org/2020/152.

[AC20b] Thomas Attema and Ronald Cramer. "Compressed *varSigma*-Protocol Theory and Practical Application to Plug & Play Secure Algorithmics". In: *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Lecture Notes in Computer Science. Springer, 2020, pp. 513–543. DOI: 10.1007/978-3-030-56877-1_18. URL: https://doi.org/10.1007/978-3-030-56877-1%5C_18.

[AFK23] Thomas Attema, Serge Fehr, and Michael Klooß. "Fiat-Shamir Transformation of Multi-round Interactive Proofs". In: Jan. 2023, pp. 113–142. ISBN: 978-3-031-22317-4. DOI: 10.1007/978-3-031-22318-1_5.

[Boo+16] Jonathan Bootle et al. "Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting". In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 327–357. DOI: 10.1007/978-3-662-49896-5_12. URL: https://doi.org/10.1007/978-3-662-49896-5%5C_12.

[CY24] Alessandro Chiesa and Eylon Yogev. *Building Cryptographic Proofs from Hash Functions*. 2024. URL: https://github.com/hash-based-snargs-book.

[Fis05] Marc Fischlin. "Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors". In: *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*. Ed. by Victor Shoup. Lecture Notes in Computer Science. Springer, 2005, pp. 152–168. DOI: 10.1007/11535218_10. URL: https://doi.org/10.1007/11535218%5C_10.

[FS86] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*. Ed. by Andrew M. Odlyzko. Lecture Notes in Computer Science. Springer, 1986, pp. 186–194. DOI: 10.1007/3-540-47721-7_12. URL: https://doi.org/10.1007/3-540-47721-7%5C_12.

[GRY25] Ziyi Guan, Artur Riazanov, and Weiqiang Yuan. "Breaking Verifiable Delay Functions in the Random Oracle Model". In: *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part VII*. Ed. by Yael Tauman Kalai and Seny F. Kamara. Lecture Notes in Computer Science. Springer, 2025, pp. 161–191. DOI: 10.1007/978-3-032-01907-3_6. URL: https://doi.org/10.1007/978-3-032-01907-3%5C_6.

[KR25] Michael Klooß and Michael Reichle. "Blind Signatures from Proofs of Inequality". In: *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part VI*. Ed. by Yael Tauman Kalai and Seny F. Kamara. Lecture Notes in Computer Science. Springer, 2025, pp. 157–189. DOI: 10.1007/978-3-032-01887-8_6. URL: https://doi.org/10.1007/978-3-032-01887-8%5C_6.

[KS22] Yashvanth Kondi and Abhi Shelat. "Improved Straight-Line Extraction in the Random Oracle Model with Applications to Signature Aggregation". In: *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part II*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13792. Lecture Notes in Computer Science. Springer, 2022, pp. 279–309. DOI: 10.1007/978-3-031-22966-4_10. URL: https://doi.org/10.1007/978-3-031-22966-4%5C_10.

[Pas03] Rafael Pass. "On Deniability in the Common Reference String and Random Oracle Model". In: *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*. Ed. by Dan Boneh. Lecture Notes in Computer Science. Springer, 2003, pp. 316–337. DOI: 10.1007/978-3-540-45146-4_19. URL: https://doi.org/10.1007/978-3-540-45146-4%5C_19.

[RT25] Lior Rotem and Stefano Tessaro. "Straight-Line Knowledge Extraction for Multi-Round Protocols". In: *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part VII*. Ed. by Yael Tauman Kalai and Seny F. Kamara. Lecture Notes in Computer Science. Springer, 2025, pp. 95–127. DOI:

10.1007/978-3-032-01907-3_4. URL: https://doi.org/10.1007/978-3-032-01907-3%5C_4.

A. Appendix

A.1. Pass Knowledge-Soundness error

Theorem 30 (Pass Knowledge-soundness-error)**.** *Let Π^{Σ} be a n-special-sound Σ-protocol and let $\mathcal{VC} = (\mathit{VC.Commit}, \mathit{VC.Open}, \mathit{VC.Verify})$ be a vector commitment scheme that is position binding with position binding error $Adv^{posbind}_{\mathcal{VC},q,A}(\lambda)$ and straight-line extractable with error $Adv^{ext}_{\mathcal{VC},q,B}(\lambda)$ ([RT25, Def. 4.1 and 4.2]). Furthermore let A, B be PPT adversaries and let $m \geq n$.*

The Knowledge-Soundness-error of $S_{Pass}(\Pi^{\Sigma})$ is

$$\kappa_{Pass} \leq q_{\mathrm{RO},\mathcal{A}} \cdot \left(\frac{n-1}{m}\right)^{t} + Adv^{posbind}_{\mathcal{VC},q_{\mathrm{RO},\mathcal{A}},A}(\lambda) + Adv^{ext}_{\mathcal{VC},q_{\mathrm{RO},\mathcal{A}},B}(\lambda)$$

where $q_{\mathrm{RO},\mathcal{A}}$ is the maximum number of Random Oracle queries the attacker can make and t is the amount of parallel repetitions.

Proof. (sketch) First we describe the Extractor for the Pass compiler. It checks all queries made to the Random Oracle to find the specific query that yielded the challenge indices $(q_1, \cdots, q_t)$ in the proof. Then because the commitment scheme is straight-line extractable in the ROM, $\mathcal{E}$ recovers the underlying responses for all commitments. Finally, $\mathcal{E}$ checks if at least one repetition allows the extraction of n valid transcripts and therefore a witness.

We bound the probability, that the verifier accepts the proof π but $\mathcal{E}$ outputs $\perp$ by first analysing the probability, that the prover didn't commit to n valid transcripts in any of the t repetitions and then bounding the probability that the commitment extraction did not work.

If we consider that the commitment extraction works perfectly, for $\mathcal{E}$ to output $\perp$, the adversary $\mathcal{A}$ must have committed to *at most* $n-1$ valid transcripts in *every* repetition i. If $\mathcal{A}$ had committed to n valid transcripts in any single repetition, $\mathcal{E}$ would have successfully extracted the witness.

Let E_{fake} be the event that $\mathcal{A}$ commits to at most $n-1$ valid transcripts in all t repetitions. For the verifier to accept the proof under E_{fake}, the Random Oracle must output a challenge vector $(q_1, \ldots, q_t)$ that selects one of the $n-1$ valid transcripts $\mathcal{A}$ actually prepared in every single repetition.

For a single Random Oracle query, the probability of the oracle outputting "lucky" indices $(q_1, \ldots, q_t)$ that land on valid transcripts across all t repetitions is at most:

$$\left(\frac{n-1}{m}\right)^t$$

However, $\mathcal{A}$ is allowed to make up to $q_{\mathrm{RO},\mathcal{A}}$ queries to the Random Oracle to hunt for a favourable challenge vector. By applying the Union Bound over all $q_{\mathrm{RO},\mathcal{A}}$ queries, the probability that $\mathcal{A}$ finds *any* successful challenge vector while E_{fake} holds is bounded by:

$$q_{\mathrm{RO},\mathcal{A}} \cdot \left(\frac{n-1}{m}\right)^t$$

Finally, we must account for the cryptographic failures of the vector commitment scheme. There are two possibilities:

- The event where the valid proofs outputted by the adversary do not match the values extracted by $\mathcal{E}$ corresponds to breaking the **straight-line extractability** of the commitment scheme. This is bounded by the advantage of a PPT adversary B, denoted as $\mathrm{Adv}^{\mathrm{ext}}_{\mathcal{VC},q,B}(\lambda)$.
- The event where the adversary successfully opens a commitment to a different value than what was extracted corresponds to breaking the **position binding** property. This is bounded by the advantage of an adversary A, denoted as $\mathrm{Adv}^{\mathrm{posbind}}_{\mathcal{VC},q_{\mathrm{RO},\mathcal{A}},A}(\lambda)$.

Per definition $\mathrm{Adv}^{\mathrm{ext}}_{\mathcal{VC},q_{\mathrm{RO},\mathcal{A}},B}(\lambda)$ and $\mathrm{Adv}^{\mathrm{posbind}}_{\mathcal{VC},q_{\mathrm{RO},\mathcal{A}},A}(\lambda)$ are negligible. Summing these independent failure possibilities gives us the final knowledge soundness error:

$$\kappa_{Pass} \leq q_{\mathrm{RO},\mathcal{A}} \cdot \left(\frac{n-1}{m}\right)^t + \mathrm{Adv}^{\mathrm{posbind}}_{\mathcal{VC},q_{\mathrm{RO},\mathcal{A}},A}(\lambda) + \mathrm{Adv}^{\mathrm{ext}}_{\mathcal{VC},q_{\mathrm{RO},\mathcal{A}},B}(\lambda)$$

□

A.2. Fischlin Knowledge Soundness Error

Theorem 31 (Fischlin Knowledge Soundness Error). *Let Π^{Σ} be an n-strong-special-sound Σ-protocol. We show that the compiled non-interactive protocol $S(\Pi^{\Sigma})$ has a knowledge soundness error of:*

$$\kappa_{Fischlin} \leq q_{\mathrm{RO},\mathcal{A}} \cdot \left(\frac{n-1}{2^l}\right)^t$$

where $q_{\mathrm{RO},\mathcal{A}}$ is the maximum number of Random Oracle queries the attacker can make, l is the target number of leading zeros for the proof-of-work hash, and t is the number of parallel repetitions.

Proof. (sketch) To prove knowledge soundness, we must bound the probability that an adversary $\mathcal{A}$ outputs a valid proof π that the verifier accepts, but the straight-line extractor $\mathcal{E}$ fails to extract a witness $\mathbb{w}$ from the adversary's Random Oracle query list, $Q_{\mathcal{RO}}$.

The proof π consists of a committed first-message vector $\vec{a} = (a_1, \ldots, a_t)$ and t transcripts (a_i, c_i, z_i). To extract a witness from a protocol with n-strong-special-soundness, the extractor $\mathcal{E}$ needs to find n distinct valid transcripts in $Q_{\mathcal{RO}}$ for at least one repetition i.

The extractor $\mathcal{E}$ works by going through all $q_{\mathrm{RO},\mathcal{A}}$ oracle queries in $Q_{\mathcal{RO}}$ and checks if there is a set of n queries $H(\mathbb{x}, \vec{a}, i, c_i, z_i)$ for the same i with distinct (c_i, z_i).

Extraction fails if $\mathcal{A}$ queries at most $n-1$ distinct, valid transcripts for every repetition $i \in \{1, \ldots, t\}$. Let this be the event E_{fake}.

For the verifier to accept the proof under E_{fake}, $\mathcal{A}$ must find a transcript (a_i, c_i, z_i) in every repetition i that satisfies the proof-of-work condition: $H(\mathbb{x}, \vec{a}, i, c_i, z_i) = 0^l$. Because the hash values in the Random Oracle Model are uniformly distributed, finding l leading zeros for any specific valid transcript occurs with probability 2^{-l}.

Since $\mathcal{A}$ knows at most $n-1$ valid transcripts per repetition under E_{fake}, we apply the Union Bound to find the maximum probability that $\mathcal{A}$ successfully finds a valid proof-of-work for a single repetition i:

$$\Pr(\text{Success in repetition } i) \leq \sum_{j=1}^{n-1} 2^{-l} = \frac{n-1}{2^l}$$

For a fixed first-message vector $\vec{a}$, the probability that $\mathcal{A}$ succeeds in finding a valid proof-of-work for all t independent repetitions is:

$$\Pr(\text{Success across all } t \text{ repetitions}) \leq \left(\frac{n-1}{2^l}\right)^t$$

However, $\mathcal{A}$ is not restricted to a single vector $\vec{a}$. The adversary can make up to $q_{\text{RO},\mathcal{A}}$ total queries to the Random Oracle, allowing them to try multiple different vectors $\vec{a}$ to mine for a successful proof. By applying the Union Bound over all $q_{\text{RO},\mathcal{A}}$ queries, the total probability that $\mathcal{A}$ finds *any* vector $\vec{a}$ that satisfies the proof-of-work for all t repetitions while E_{fake} holds is bounded by:

$$\kappa_{\text{Fischlin}} \leq \frac{q_{\text{RO},\mathcal{A}}}{t} \cdot \left(\frac{n-1}{2^l}\right)^t \leq q_{\text{RO},\mathcal{A}} \cdot \left(\frac{n-1}{2^l}\right)^t$$

□

A.3. Kondi-Shelat Knowledge Soundness Error

Theorem 32 (Kondi-Shelat Knowledge Soundness Error)**.** *Let Π^Σ be an n-strong-special-sound Σ-protocol. We show that the compiled non-interactive protocol $S_{KS}(\Pi^\Sigma)$ utilizing t parallel repetitions and requiring r-way hash collisions of l-bit hash values has a knowledge soundness error bounded by:*

$$\kappa_{KS} \leq q_{\text{RO},\mathcal{A}} \cdot \left(\binom{n-1}{r} \frac{1}{2^{l(r-1)}}\right)^t$$

where $q_{\text{RO},\mathcal{A}}$ is the maximum number of Random Oracle queries the attacker can make.

Proof. To prove knowledge soundness, we must bound the probability that an adversary $\mathcal{A}$ outputs a valid proof that the verifier accepts, but the straight-line extractor $\mathcal{E}$ fails to extract a witness $\mathbb{w}$ from the adversary's Random Oracle query list, $Q_{\mathcal{RO}}$.

The proof consists of a committed first-message vector $\vec{a} = (a_1, \ldots, a_t)$ and t sets of r colliding transcripts. To extract a witness from a protocol with n-strong-special-soundness, the extractor $\mathcal{E}$ needs to find n valid transcripts in $Q_{\mathcal{RO}}$ for at least one repetition i.

Extraction fails if $\mathcal{A}$ queries at most $n-1$ distinct, valid transcripts for every repetition $i \in \{1, \ldots, t\}$. Let this be the event E_{fake}.

For the verifier to accept the proof under E_{fake}, $\mathcal{A}$ must find r valid transcripts in every repetition i that satisfy the multi-collision condition: $H(x, \vec{a}, i, c_i^{(1)}, z_i^{(1)}) = \cdots = H(x, \vec{a}, i, c_i^{(r)}, z_i^{(r)})$. Because the hash values in the Random Oracle Model are uniformly distributed over $\{0,1\}^l$, the probability that any specific set of r transcripts collides to the same arbitrary value is $(2^{-l})^{r-1}$.

Since $\mathcal{A}$ knows at most $n-1$ valid transcripts per repetition under E_{fake}, there are at most $\binom{n-1}{r}$ possible subsets of size r they can form. We apply the Union Bound to find the maximum probability that $\mathcal{A}$ successfully finds an r-way collision for a single repetition i:

$$\Pr(\text{Success in repetition } i) \leq \binom{n-1}{r} \frac{1}{2^{l(r-1)}}$$

For a fixed first-message vector $\vec{a}$, the probability that $\mathcal{A}$ succeeds in finding a valid r-way collision for all t independent repetitions is:

$$\Pr(\text{Success across all } t \text{ repetitions}) \leq \left(\binom{n-1}{r} \frac{1}{2^{l(r-1)}} \right)^t$$

However, $\mathcal{A}$ is not restricted to a single vector $\vec{a}$. The adversary can make up to $q_{\text{RO},\mathcal{A}}$ total queries to the Random Oracle, allowing them to try multiple different vectors $\vec{a}$ to mine for successful collisions. By applying the Union Bound over all $q_{\text{RO},\mathcal{A}}$ queries, the total probability that $\mathcal{A}$ finds any vector $\vec{a}$ that satisfies the collision condition for all t repetitions while E_{fake} holds is bounded by:

$$\kappa_{KS} \leq q_{\text{RO},\mathcal{A}} \cdot \left(\binom{n-1}{r} \frac{1}{2^{l(r-1)}} \right)^t$$

Thus, the knowledge soundness error for the Kondi-Shelat compiler is bounded as required. □

A.4. Communication complexity example for Fischlin

To illustrate the exponential scaling of the communication complexity, we examine a protocol with $\mu = 3$ and repetition parameters t_1, t_2, and t_3.

1. The Base Interactive Protocol ($\mu = 3$):

$$a_1, c_1, a_2, c_2, a_3, c_3, a_4$$

2. Compressing Round 3 (t_3 repetitions):
The compiler first targets the last three messages. The prefix (a_1, c_1, a_2, c_2) is needed only once, while the final messages are repeated t_3 times to form π_3:

$$a_1, c_1, a_2, c_2, \underbrace{\left[(a_3^{(1)}, c_3^{(1)}, a_4^{(1)}), \ldots, (a_3^{(t_3)}, c_3^{(t_3)}, a_4^{(t_3)})\right]}_{\pi_3}$$

The communication complexity for one π_3 is $t_3 \cdot (W_{a_3} + W_{c_3} + W_{a_4})$

3. Compressing Round 2 (t_2 repetitions):
Next, the compiler compresses round 2. The prefix (a_1, c_1) remains. To create π_2, one transcript which gets sent consists of a_2, c_2, π_3. Since we need to send t_2 transcirpts the communication overhead for one π_2 is $W_{\pi_2} = t_2 \cdot (W_{a_2} + W_{c_2} + W_{\pi_3}) = t_2 \cdot (W_{a_2} + W_{c_2} + [t_3 \cdot (W_{a_3} + W_{c_3} + W_{a_4})])$

The new "last message" is the entire π_3 block from the previous step, which now gets duplicated t_2 times alongside a_2 and c_2:

$$a_1, c_1, \underbrace{\begin{bmatrix} a_2^{(1)}, c_2^{(1)}, & \left[(a_3^{(1,1)}, c_3^{(1,1)}, a_4^{(1,1)}), \ldots, (a_3^{(1,t_3)}, c_3^{(1,t_3)}, a_4^{(1,t_3)})\right] \\ \vdots & \vdots \\ a_2^{(t_2)}, c_2^{(t_2)}, & \left[(a_3^{(t_2,1)}, c_3^{(t_2,1)}, a_4^{(t_2,1)}), \ldots, (a_3^{(t_2,t_3)}, c_3^{(t_2,t_3)}, a_4^{(t_2,t_3)})\right] \end{bmatrix}}_{\pi_2}$$

4. Final Compilation Step (Fully Non-Interactive Proof π_1 with t_1 repetitions):
Finally, round 1 is compressed. The *entire* π_2 matrix above must now be repeated t_1 times!

$$\pi_1 = \begin{bmatrix} a_1^{(1)}, c_1^{(1)}, & \pi_2^{(1)} \\ \vdots & \vdots \\ a_1^{(t_1)}, c_1^{(t_1)}, & \pi_2^{(t_1)} \end{bmatrix}$$

Resulting Communication Complexity:

$$W_{C(\Pi^3)} = t_1(W_{a_1} + W_{c_1}) + t_1 t_2 (W_{a_2} + W_{c_2}) + t_1 t_2 t_3 (W_{a_3} + W_{c_3} + W_{a_4})$$

A.5. Table of Notation

Table A.1.: Table of Notation

Symbol	Description
1. Global Cryptographic Parameters	
λ	The security parameter.
s	The maximum length of the input statement x.
σ	The bit-length of the output of the Random Oracle ($\mathcal{RO}$).
$q_{\mathsf{RO},\mathcal{A}}$	The maximum number of random oracle queries an attacker/adversary can make.
$q_{\mathsf{RO},P}$	The maximum number of random oracle queries the honest prover can make.
2. Protocol Structures & Roles	
$\mathbb{x}$, $\mathbb{w}$	The input statement and the secret witness, respectively.
R	The verification relation such that $(\mathbb{x}, \mathbb{w}) \in R$.
$\tilde{R}$	The extraction relation used for knowledge soundness.
P, V	The interactive Prover and Verifier algorithms.
$P_{\mathsf{ni}}, V_{\mathsf{ni}}$	The non-interactive argument prover and verifier algorithms in the ROM.
Π^{Σ}	A 3-move, interactive Σ-protocol.
Π^{μ}	An interactive, public-coin protocol consisting of $2\mu + 1$ messages (i.e., μ rounds).
h_i	The partial transcript or history of the protocol up to round i.
3. Compilers & Modularity	
S	The underlying straight-line extractable (sle) compiler used as a black-box to compress 3-move Σ-protocols.
C	The modular, recursive sle-compiler that transforms the full multi-round protocol Π^{μ} into a non-interactive protocol.
C^1	The compiler C applied only to compress the last 3 moves of an interactive protocol.
Π_i^{Σ}	The intermediate Σ-protocol to which the sle-compiler S is applied in round i.
t_i	The number of parallel repetitions executed by the underlying compiler S in round i.
n_i	The branching factor in round i, representing the number of distinct challenges/transcripts needed for n_i-special soundness.
4. Error Bounds & Analytical Functions	

Continued on next page...

Table A.1 – continued from previous page

Symbol	Description
$\epsilon_{int}, \epsilon_{\mathsf{ni}}$	The completeness error for the interactive and non-interactive protocols, respectively.
ϵ_{2S}	The 2-Stage special Honest-Verifier Zero-Knowledge (2SsHVZK) error.
κ	The straight-line knowledge soundness error.
f_S	The iterative completeness error function of the underlying compiler S.
g_S	The iterative runtime function for the prover/verifier.
g_{com}	The iterative communication complexity function.
T_{Π^μ}	The runtime of the prover or verifier within the specified protocol Π^μ.
$W_{a_i}, W_{c_i}, W_{\pi_i}$	The communication complexity (size) of the prover message a_i, verifier challenge c_i, and proof π_i.
5. Extraction Variables	
$\mathcal{E}_C$	The recursive transcript-tree extractor for the fully compiled protocol $C(\Pi^\mu)$.
$\mathcal{E}_{S[n_i]}$	The extractor for the underlying compiler S that outputs exactly n_i valid transcripts.
T_i, τ_j	The extracted transcript trees or subtrees generated during the extraction process.