MASTER THESIS

On State-Restoration Knowledge Soundness from Special Soundness

Mathias Marty
mathias.marty@epfl.ch
EPFL

Responsible
Prof. Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Supervisor
Ziyi Guan
ziyi.guan@epfl.ch
EPFL

School of Computer and Communication Sciences

December 21, 2024

Abstract

In the realm of cryptographic protocols, ensuring the integrity and security of proofs is paramount. Proof systems enable a verifier to check the validity of a statement provided by a prover, ensuring correctness with high probability. Interactive proofs (IPs) are a specific type of proof system that involves a multi-round exchange between a prover and a verifier.

The security of IPs against malicious provers is captured through the concept of *soundness*. There are multiple strengthenings of soundness such as *special soundness*, which ensures that multiple accepting transcripts with distinct challenges can be used to extract a witness, and *knowledge soundness*, which asserts that any convincing prover actually knows a witness for the statement. In particular, we focus on a stronger version of knowledge soundness called *strong knowledge soundness*. Another strengthening of knowledge soundness is *state restoration knowledge soundness*. State restoration knowledge soundness ensures the Fiat–Shamir protocol transformation is knowledge sound. We define state restoration knowledge soundness in the *random oracle model (ROM)*, which is extensively used in cryptography.

Special soundness is usually easier to prove than strong knowledge soundness and state restoration knowledge soundness. This work demonstrates that special soundness implies both strong knowledge soundness and state restoration knowledge soundness, making it a helpful security notion.

Keywords: cryptographic protocols, special soundness, state restoration, sigma protocols, interactive proofs, knowledge soundness, random oracle model

Acknowledgments

I would like to express my deepest gratitude to Professor Alessando Chiesa for his thorough, frequent and invaluable feedback and discussions on the subject, which allowed me to gain the necessary intuition to compile this thesis. I am also very grateful to Ziyi Guan, who has spent many hours spotting small issues in the thesis and coaching me on academic writing.

Contents

1 Introduction

In cryptographic protocols, maintaining the integrity and security of proofs is crucial. IPs and their non-interactive counterparts (in the ROM) non-interactive random oracle proofs (NIROPs) play a crucial role in this context, providing the foundation for secure communication and computation in numerous applications. One of the critical challenges in this field is to ensure that these proofs are sound, meaning that they cannot be convincingly generated without possessing a valid witness for the statement being proven. This thesis delves into the intricate relationship between special soundness, knowledge soundness, and state-restoration knowledge soundness, presenting significant current research advancements and generalizing these results.

1.1 Motivation

Special soundness is a well-established property of sigma protocols (SPs) and their generalization to multiple rounds, public coin IPs. It ensures that an efficient extractor can leverage the knowledge of multiple accepting transcripts to extract a witness. This property has been instrumental in constructing secure protocols; however, its implications for more complex and practical scenarios have not been fully explored. Specifically, the challenge arises when considering *state-restoration attacks*, where a malicious prover can reset and replay parts of the protocol to gain an unfair advantage. Such attacks, however, have a concrete interpretation; they give the adversary the same advantage he would have when attacking the *Fiat–Shamir* transformation of the protocol.

Another crucial security property of public coin IPs and their Fiat–Shamir transformations is *knowledge soundness*. Briefly, this property ensures that one cannot prove a statement without *knowing* a witness for this statement. We introduce later a stronger notion of knowledge soundness called *strong knowledge soundness*. In short, strong knowledge soundness captures in its definition the transcript of an actual execution. This strengthening brings us closer to an *adaptive* security property, where adaptive means that the malicious provers are allowed to *choose* the problem statement before attacking. In this work, we refer to strong knowledge soundness simply as knowledge soundness, while we refer to standard knowledge soundness as *weak knowledge soundness*. Finally, we consider another strengthening of knowledge soundness known as *state restoration knowledge soundness*, which protects against state restoration attacks. Moreover, this property protects against *adaptive provers*.

Weak knowledge soundness, strong knowledge soundness, and state restoration knowledge soundness are more challenging to demonstrate than special soundness. All of these security properties rely on the existence of an efficient extractor. Still, the special sound extractor is deterministic and leverages the combinatorial properties of the IPs to extract a witness. In contrast, the knowledge extractors are probabilistic and must consider all possible malicious provers.

The primary objective of this work is to investigate whether special soundness can imply state restoration knowledge soundness for SPs and, more generally, public coin IPs. Achieving this would enhance the security guarantees of these proof systems, particularly when transformed into non-interactive proofs using the Fiat–Shamir transformation. A weaker result has been demonstrated in [AFK22]. In this work, the authors prove that the Fiat–Shamir transformation of a special sound IP is knowledge sound; however, they prove this result without relying on the intermediate notion of state restoration knowledge soundness, which is the focus of our work.

To gain some intuition about this question, we begin, as a warmup, by asking ourselves if special sound IPs are also knowledge sound. A similar result, yet weaker, is shown in [ACK21]. This paper demonstrates that a special sound IP is *weak knowledge sound*. We exhibit the main results of this thesis in the section below.

Recently, a book on succint non-interactive arguments has been published by Chiesa and Yogev. The results presented in this Master's thesis also appear in this book [CY24].

1.2 Results

Let $\mathbb{x}$ denote any problem instance. Our warmup question leads to the following statement.

Theorem 1.1 (Informal)**.** *Let* $\mathsf{IP} = (\mathbf{P}_{\mathsf{IP}}, \mathbf{V}_{\mathsf{IP}})$ *be a public-coin interactive proof for a relation* R *with round complexity* k*,* $(a_1, \ldots, a_{\mathsf{k}})$*-special soundness and randomness complexity* $(\mathsf{r}_1, \ldots, \mathsf{r}_{\mathsf{k}})$*. Then* IP *has knowledge soundness error*

$$\kappa_{\mathsf{IP}}(\mathbb{x}) \leq 1 - \prod_{i=1}^{\mathsf{k}} \left(1 - \frac{a_i - 1}{2^{\mathsf{r}_i}}\right) \ .$$

We now turn to the state restoration setup. Since the latter is defined over the ROM, we must consider malicious provers with access to random oracles. Thus, let t denote an upper bound on the number of queries the malicious prover does to these random oracles.

Theorem 1.2 (Informal)**.** *Let* $\mathsf{IP} = (\mathbf{P}_{\mathsf{IP}}, \mathbf{V}_{\mathsf{IP}})$ *be a public-coin interactive proof for a relation* R *with* $(a_1, \ldots, a_{\mathsf{k}})$*-special soundness and verifier randomness complexity* $(\mathsf{r}_1, \ldots, \mathsf{r}_{\mathsf{k}})$*. It follows that* IP *has rewinding state restoration knowledge soundness error*

$$\kappa^{\mathsf{sr}}_{\mathsf{IP}}(\mathbb{x}, t) \leq (t+1)\left(1 - \prod_{i=1}^{\mathsf{k}} \left(1 - \frac{a_i - 1}{2^{\mathsf{r}_i}}\right)\right) \ .$$

The result above strengthens the security guarantees of the Fiat–Shamir transformation in the ROM. This thesis presents the building blocks necessary to prove these two theorems.

1.3 Thesis overview

First, Sections 2.1 to 2.4 address the required theoretical background and notations needed to understand this work. Then, we explore the concept of knowledge soundness and special soundness and their implications in cryptographic protocols, focusing on SPs and, more generally, IPs in Sections 2.5 and 2.6.

We familiarize ourselves with the concept of state restoration in Section 2.7. The introduction of the state restoration game provides a robust framework for analyzing the security of protocols against state restoration attacks. This game simulates scenarios where a malicious prover may try to reset its state to gain an advantage. Finally, we explain how state restoration knowledge soundness relates to the security of IPs' non-interactive Fiat–Shamir transformations over the ROM.

Then, we show that special soundness implies knowledge soundness. This proof provides a more straightforward way to show that an IP is knowledge sound without the necessity to consider adversarial strategies. We first prove this statement for SPs in Section 3 and then move to public coin IPs in Section 4. This step-by-step approach will be a common pattern in this thesis; indeed, one can have difficulty grasping the proofs of the latter sections without the intuition of the more straightforward ones.

Finally, we demonstrate that special soundness not only implies knowledge soundness, but also state restoration knowledge soundness, which, in turn, signifies that the non-interactive Fiat–Shamir transformation is knowledge sound. The proof is first done for SPs in Section 5 and then for public coin IPs in Section 6.

In summary, the key points we bring in this thesis are:

- *Strong knowledge soundness from special soundness.* We show that special soundness is sufficient to guarantee strong knowledge soundness for SPs. This involves constructing an efficient extractor and analyzing its performance and error rates, ensuring that it can reliably extract witnesses in polynomial time.
- *Extension to interactive proofs.* We extend this result from SPs to public coin IPs, demonstrating that the same principles apply. We achieve this by adapting the extraction techniques and ensuring they were compatible with the more complex structure of IPs.
- *Implication for non-interactive proofs.* By proving that special soundness implies state restoration knowledge soundness, we establish that the a special sound interactive proof is also secure against state-restoration attacks. This implies that its Fiat–Shamir transformation is also knowledge sound.
- *Natural extension to interactive oracle proofs.* Our work generalizes naturally to interactive oracle proofs (IOPs). Although not proved formally, every extractor constructed in this thesis has a straightforward generalization that allows it to work on IOPs instead of IPs.

This work underscores the importance of soundness properties in cryptographic protocols and provides tools for further research into secure-proof systems.

1.4 Future work

There are many examples of IPs where the notion of special soundness leads to insufficient knowledge soundness error, motivating research toward more fine-grained relaxation of special soundness for these cases. Attema, Fehr, and Resch have proposed in [AFR23] the notion of Γ-*special soundness*. In short, if Γ is a monotone subset of the power set of all possible challenges, then Γ-special soundness asserts that there exists an efficient extractor such that given any fork $(\alpha_1, ((\rho_1, \alpha_{2,1}), \ldots, (\rho_a, \alpha_{2,a}))$, where $\{\rho_1, \ldots, \rho_a\} \in \Gamma$, can retrieve a witness. It would be interesting to see if Γ-special soundness also implies state restoration knowledge soundness. If this turns out to be true, it would automatically extend their result to the non-interactive transformation of IPs.

2 Preliminaries

This section introduces the necessary theoretical background to understand the proofs and concepts presented in this thesis. We use the symbols $\mathbb{N}$ and $\mathbb{N}^*$ to denote the natural numbers, including zero, and the natural numbers, excluding zero. For any $q \in \mathbb{N}^*$, we denote by $\mathbb{Z}_q$ the ring of integer modulo q. Remark that this group forms a field when q is a prime number. Moreover, we write $\mathsf{poly}(x)$ to refer to some unspecified function in x, which is upper bounded by some monomial in x.

2.1 Languages and relations

A *formal language*, which we refer to *language* in this thesis, is a set of words composed of symbols that lie in an *alphabet*. The latter is a set of symbols, such as the digits or the alphanumeric characters. Throughout this work, we will focus our attention on the *binary alphabet*, which is, as its name suggests, the set $\{0,1\}$. Therefore, we restrict all subsequent definitions to the binary alphabet. This restriction is not limiting because we can encode any formal language over an alphabet of cardinality $b > 2$ (the case $b = 2$ is trivial) into a binary alphabet where every word of length n in the larger alphabet has a binary representation of length $\log_2(b) \cdot n$, which is linear in n. Consequently, studying languages over the binary alphabet encompasses all possible formal languages, maintaining generality while simplifying analysis.

Notation 1. *For every $n \in \mathbb{N}$, we denote by $\{0,1\}^n$ the set of binary strings of length n. Moreover,*

$$\{0,1\}^{\leq n} \coloneqq \bigcup_{i=0}^{n} \{0,1\}^i$$

is the set of all binary strings of length smaller or equal to n. Finally, we capture by

$$\{0,1\}^* \coloneqq \bigcup_{i \in \mathbb{N}} \{0,1\}^i \ ,$$

where we define $\{0,1\}^0$ to be the empty string, all finite strings.

We use the previous notation to define the concept of language formally.

Definition 2.1. *A* **language** *L is any subset of $\{0,1\}^*$. We say that an* **instance** *$\mathtt{x} \in \{0,1\}^*$ is in the language L if and only if $\mathtt{x} \in L$.*

Languages capture the notion of *decision problems*, which are problems with "yes" or "no" answers. For example, one might study the language of numbers that are *not prime*

$$\bar{L}_{\mathsf{primes}} \coloneqq \{n \in \mathbb{N}^* \mid n \text{ is not prime}\} \ .$$

Another crucial concept is the notion *relations*.

Definition 2.2. *A* **relation** *R is any subset of $\{0,1\}^* \times \{0,1\}^*$. The set R consists of instance-witness pairs, usually denoted by $(\mathtt{x}, \mathtt{w})$.*

Any relation implicitly defines the following language.

Definition 2.3. *Given a relation R, we define its corresponding language as*

$$L(R) \coloneqq \{\mathtt{x} \mid \exists \mathtt{w}. \quad (\mathtt{x}, \mathtt{w}) \in R\} \ .$$

Relations capture another type of problem, the *search problems*. These problems possess pairs of instance-witness that we can interpret as a problem instance and its solution. For example, we could extend the decision problem $\bar{L}_{\mathsf{primes}}$ to the search problem $\bar{R}_{\mathsf{primes}}$ where the witnesses are the prime decompositions of the composite numbers. We give below another concrete search problem called the *discrete logarithm problem*. This problem will follow us throughout this thesis as a running example.

Example 1. Given any group $\mathbb{G}$ of prime order q, written multiplicatively, and g a generator of the latter, the ***discrete logarithm problem over the group generated by*** g is defined as

$$R_{\mathsf{dlog}}(g) \coloneqq \{(h, x) \in \mathbb{Z}_q \times \mathbb{G} \mid g^x = h\} \ .$$

For clarity, we should always assume that $\mathbb{G}$ is the group generated by g and q its order.

Informally, given any element $h \in \mathbb{G}$, this problem asks to retrieve the unique integer $x \in \mathbb{Z}_q$ such that the equation $g^x = h$ is verified. For some groups, this problem is known to be *hard*, meaning that one cannot design a probabilistic adversary solving this problem that runs rapidly enough on average. Hence, it is a suitable candidate for cryptographic primitives. For examples of cryptographic systems leveraging this computational problem, the reader might take a look at the *ElGamal public key cryptosystem* [EG85] and the *Diffie–Hellman key exchange* [DH76].

2.2 Algorithms and random oracles

We classify algorithms into two categories: those that use random coins and those that only depend on their input. The first category is called *probabilistic algorithms* while the other *deterministic algorithms*. Throughout this thesis, we will sometimes refer to probabilistic algorithms simply as algorithms.

Definition 2.4. *Let $\mathcal{A}$ be any probabilistic algorithm that uses r coin tosses and takes as input an x. We denote by $\zeta \in \mathcal{Z}$ the algorithm's* **randomness***, where $\mathcal{Z} \coloneqq \{0,1\}^{\mathsf{r}}$ is called its* **random space** *and r its* **randomness complexity**.

Note that we can view a deterministic algorithm as a probabilistic algorithm with randomness complexity zero; hence, the latter is a generalization of the former. We use the following notation to indicate that we run the algorithm $\mathcal{A}$ with its randomness fixed to $\zeta \in \mathcal{Z}$.

Notation 2. *We symbolize by*

$$y \leftarrow \mathcal{A}(x; \zeta) \ ,$$

where y is the value returned, an instantiation of the probabilistic algorithm $\mathcal{A}$ on randomness ζ and input x.

Note that to differentiate parameters from randomness, we use the semicolon symbol. On the other hand, we use the following notation to sample the randomness at random and then run the probabilistic algorithm.

Notation 3. *We denote by*

$$y \leftarrow \mathcal{A}(x) \ ,$$

where y is the value returned, an instantiation of the probabilistic algorithm $\mathcal{A}$ on randomness sampled uniformly with the random space of $\mathcal{A}$. Moreover, we will also use this notation when instantiating a deterministic algorithm.

Finally, we give the arrow notation one more meaning.

Notation 4. *For any finite set S, we denote by*

$$x \leftarrow S$$

the action of sampling uniformly an element x within S.

When describing the pseudocode of an algorithm, we will use the symbol ":=" to indicate a deterministic assignment. However, we make an exception for deterministic algorithms, where we keep the arrow notation. This exception simplifies notations, especially when introducing *traces*.

Another essential aspect of algorithms, whether deterministic or probabilistic, is their *running time*. The running time is a crucial metric in complexity theory, as it determines how efficiently an algorithm can solve a problem relative to the input size. Analyzing the running time helps understand the algorithm's performance and scalability, which is vital for comparing different algorithms and choosing the most appropriate one for practical applications. Below, we define a notation to capture the running time of *one specific execution*.

Definition 2.5. *Let $\mathcal{A}$ be any probabilistic algorithm. Given its randomness ζ and an input x, its* **running time***, denoted by $\tau_{\mathcal{A}}(x;\zeta)$, computes the number of elementary operations performed by*

$$\mathcal{A}(x;\zeta) \quad .$$

Moreover, we write $\tau_{\mathcal{A}}(x)$ if the algorithm is deterministic.

This definition is too fine-grained for our purpose; for example, complexity theory usually deals with *worst-case time complexity*, aggregating all running time for inputs of a fixed length by taking only the *longest one*.

Definition 2.6. *Let $\mathcal{A}$ be any deterministic algorithm. Its* **worst-case running time** *is a function of the input length $n \in \mathbb{N}^*$ defined as*

$$\max_{x \in \{0,1\}^n} \tau_{\mathcal{A}}(x) \quad .$$

Hash functions are cryptographic tools that allow the construction of various cryptographic primitives. A helpful tool to analyze such primitives is the *ROM*. In this model, we replace the hash functions with *random oracles*. Random oracles are idealized hash functions that map any bitstrings to strings of fixed length, distributed uniformly at random, in constant time. This model allows for detaching the implementation details of the hash function from the cryptographic primitive itself and, thus, makes the analysis much more manageable. Compiling protocols defined in the ROM into concrete cryptographic protocols is a challenging problem and constitutes its own field of research. Nonetheless, this thesis focuses exclusively on the ROM.

Definition 2.7. *A random oracle* rnd *with output size* r *is a function that takes any binary string of finite length as input, picks a string of length* r *uniformly at random, and outputs it. Moreover,* rnd *keeps a map from inputs to outputs, so if it is called twice with the same input, it will reply with the same output. Finally, we assume that this function runs instantly.*

Since they execute instantaneously, we view random oracles as functions instead of algorithms. Therefore, we will use the notation

$$y := \mathsf{rnd}(x)$$

when calling random oracles in pseudocodes.

The following is a helpful notation when dealing with probability experiments, which we formally define later in Section 2.3.

Definition 2.8. *For every* $\mathbf{r} = (\mathsf{r}_1, \ldots, \mathsf{r}_\mathsf{k}) \in (\mathbb{N}^*)^\mathsf{k}$*, let* $\mathcal{U}(\mathbf{r})$ *be the uniform distribution over all ordered tuple* $(\mathsf{rnd}_1, \ldots, \mathsf{rnd}_\mathsf{k})$ *of random oracles such that* rnd_i *maps finite bitstrings to a bitstrings of length* 2^{r_i}*. Especially, if* $\mathsf{k} = 1$ *we use the notation* $\mathcal{U}(\mathsf{r})$ *instead of* $\mathcal{U}((\mathsf{r}))$ *for convenience.*

Random oracles are only helpful if algorithms can use them. So, we are interested in combining algorithms with the idea of random oracles.

Definition 2.9. *An algorithm* $\mathcal{A}$ *is a* **random-oracle** *algorithm if, in addition to its input* x*, it takes a random oracle* rnd *as an additional parameter. The algorithm* $\mathcal{A}$ *can* **query** rnd *as much as it would like. The process of running* $\mathcal{A}$ *on input* x *by giving it oracle access to* **rnd** *is denoted by*

$$y \leftarrow \mathcal{A}^{\mathsf{rnd}}(x) \ .$$

We will not only *use* these random oracles but also *simulate* them to run algorithms that can run other random-oracle algorithms by presenting a fake random oracle. Since storing a random oracle's function table requires infinite memory, we cannot hope to store the entire thing and must find a way around it. A simple trick is constructing the random oracle in a *lazy* way. A lazy oracle only keeps track of what has been already queried by the random-oracle algorithm. It picks at random for everything new, keeping its function table as small as the number of queries. In this regard, we formally introduce the following notation.

Notation 5. *To distinguish between a real random oracle and a lazy one, we use the* **dot notation**. *We write* $\dot{\mathsf{rnd}}$ *when talking about a random oracle, which we implement as a lazy random oracle instead of simply* rnd *for a real random oracle.*

Note that a lazy random oracle is a stateful algorithm, while a real random oracle is a function. We will use random oracles in definition and lazy random oracles in algorithmic constructions.

2.3 Probabilities and experiments

Throughout this thesis, we will extensively use the concept of an *experiment*. An experiment is a sequence of instructions where an instruction can either run an algorithm or sample a value within a set. We use experiments when stating probabilities, expectations, and random variables; they are beneficial when defining the security properties of cryptographic schemes, as they make them more concise and easier to read.

Definition 2.10. *An experiment is a sequence of instructions that we write vertically. We divide this sequence into two phases. The first is a* **sampling phase** *where we sample elements within finite sets. Typically, this would be sampling within random spaces of algorithms. The second phase is an* **algorithmic phase**. *In this phase, we run probabilistic algorithms and retrieve their outputs. These algorithms can take their inputs from external parameters, the values sampled during the first phase or the value returned by other algorithms. We can either fix their randomness or, as we will often do, use the values of the first phase as their randomness.*

Interestingly, we can use the results of these experiments to construct random variables through functions. Since the concept of an experiment is very generic, we will focus on a single example to introduce the subsequent notations.

Notation 6. *Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be some probabilistic algorithms and $\mathcal{Z}_1, \ldots, \mathcal{Z}_n$ their respective random spaces. Let f be any function depending on the outputs of these algorithms. We denote by*

$$\left\{ f(y_1, \ldots, y_n) \;\middle|\; \begin{array}{rcl} \zeta_1 & \leftarrow & \mathcal{Z}_1 \\ & \vdots & \\ \zeta_n & \leftarrow & \mathcal{Z}_n \\ y_1 & \leftarrow & \mathcal{A}_1(x_1; \zeta_1) \\ & \vdots & \\ y_n & \leftarrow & \mathcal{A}_n(x_n; \zeta_n) \end{array} \right\},$$

where $x_1, \ldots, x_n$ are the inputs of these algorithms, the random variable $f(\mathcal{A}_1(x_1; \zeta_1), \ldots, \mathcal{A}_n(x_n; \zeta_n))$ with sample space $\mathcal{Z}_1 \times \cdots \times \mathcal{Z}_n$. Moreover, if f has an expected value, we denote it by

$$\mathbb{E}\left[f(y_1, \ldots, y_n) \;\middle|\; \begin{array}{rcl} \zeta_1 & \leftarrow & \mathcal{Z}_1 \\ & \vdots & \\ \zeta_n & \leftarrow & \mathcal{Z}_n \\ y_1 & \leftarrow & \mathcal{A}_1(x_1; \zeta_1) \\ & \vdots & \\ y_n & \leftarrow & \mathcal{A}_n(x_n; \zeta_n) \end{array} \right].$$

The advantage of experiments is that they make all randomness of algorithms explicit. Experiments will be beneficial when defining algorithms that take the randomness of other algorithms as input.

We also use the vertical bar notation for the set-builder notation. However, it will be evident from the context whether we are describing about random variables with experiments or building set. Moreover, since we use the vertical bar notation to define experiments, we may what notation do we use to condition a random variable on an event E? We describe the way we remedy this issue in the following notation.

Notation 7. *Let E be any event over the sample space $\mathcal{Z}_1 \times \cdots \times \mathcal{Z}_n$ that may depend on some external parameters. We denote by*

$$\left\{ \begin{array}{l} f(y_1, \ldots, y_n) \\ \textit{conditioned on} \\ E \end{array} \;\middle|\; \begin{array}{rcl} \zeta_1 & \leftarrow & \mathcal{Z}_1 \\ & \vdots & \\ \zeta_n & \leftarrow & \mathcal{Z}_n \\ y_1 & \leftarrow & \mathcal{A}_1(x_1; \zeta_1) \\ & \vdots & \\ y_n & \leftarrow & \mathcal{A}_n(x_n; \zeta_n) \end{array} \right\}$$

the random variable $f(\mathcal{A}_1(x_1; \zeta_1), \ldots, \mathcal{A}_n(x_n; \zeta_n))$ conditioned on the event E. Naturally, we extend this notation to expectations and probabilities. Moreover, we will also use the experiment notation to designate the probability of the event E. Finally, we should write everything in one line if we can.

For convenience, we only explicitly write the randomness of an algorithm, including how the experiment chooses it, when it appears elsewhere than in the algorithm's call.

As discussed in Section 2.2, the notion of running time defined in Definition 2.5 is too fine-grained for our needs. Regarding probabilistic algorithms, we only need to know the *expected running time*. As its name suggests, this is the expected running time of the algorithm taken over all possible inputs and randomness. Suppose that the algorithm takes bit strings of length n as inputs, then we capture the notion of expected running time with the following.

Definition 2.11. *Let $\mathcal{A}$ be any probabilistic algorithm with random space $\mathcal{Z}$ that takes bit strings of length n as input, then*

$$\tau_{\mathcal{A}}(n) \coloneqq \mathbb{E}\left[\tau_{\mathcal{A}}(x;\zeta) \;\middle|\; \begin{array}{ccc} x & \leftarrow & \{0,1\}^n \\ \zeta & \leftarrow & \mathcal{Z} \end{array}\right]$$

denotes the **expected running time** *of $\mathcal{A}$.*

The reason we do not define the running time of a probabilistic algorithm in the same way as a deterministic algorithm, *i.e.* by taking the longest running time over all fixed-length instances and all prover randomness (see Definition 2.6), is that this makes little sense for real-world applications. In reality, we are not interested in *worst-case scenarios*, which may be avoidable edge cases or happen with negligible probability, but in the average behavior of our protocol.

A crucial class of algorithms, for both probabilistic and deterministic algorithms, are the ones that are *efficient*.

Definition 2.12. *We say that a deterministic algorithm $\mathcal{A}$ is* **efficient** *if its worst-case running time, which is a function of the input length n, is upper bounded by a polynomial in n. Similarly, a probabilistic algorithm $\mathcal{A}$ is considered* **efficient** *if its expected running time, also a function of the input length n, is upper bounded by a polynomial in n.*

These definitions of efficiency are typically too optimistic since a probabilistic algorithm with expected running time n^{2024} would be impractical. Nevertheless, efficient algorithms still form a fundamental class in complexity theory. One of the reasons is that polynomials are closed under composability, which in the algorithmic world translates into "composing several efficient algorithms yields an efficient algorithm".

We can now define the complexity class NP. This is a set of language that captures nondeterministic polynomial time (NP) problems, which we define as follows.

Definition 2.13. *A language L is in NP if and only if for all $n \in \mathbb{N}^*$ and $\mathtt{x} \in \{0,1\}^n$, there exists an efficient deterministic algorithm $\mathcal{A}$ such that*

$$\mathtt{x} \in L \Leftrightarrow \left(\exists \mathtt{w} \in \{0,1\}^{\leq \mathsf{poly}(n)}. \quad \mathcal{A}(\mathtt{x},\mathtt{w}) = 1\right) .$$

From such a language, we can explicitly construct a relation; the relation

$$R \coloneqq \bigcup_{i \in \mathbb{N}^*} \{(\mathtt{x},\mathtt{w}) \in \{0,1\}^n \times \{0,1\}^{\leq \mathsf{poly}(n)} \mid \mathcal{A}(\mathtt{x},\mathtt{w}) = 1\}$$

is know as a NP relation.

The class NP captures problems with small solutions that can be verified efficiently by an algorithm. On the other hand, the class of NP relation captures the problems *and* their small solution. For example, the decision problem $\bar{L}_{\mathsf{primes}}$ is in NP since we could compute the product of the factors and check it is equal to the instance. Its NP relation is $\bar{R}_{\mathsf{primes}}$. Moreover, the relation $R_{\mathsf{dlog}}(g)$ is a NP relation, where the solutions are the discrete logarithms.

Finally, we introduce the following probability distribution.

Definition 2.14. *Consider the following probabilistic experiment. We sample balls, without replacement, from a set of size N consisting of K success balls and $N-K$ failure balls until we have found $r \leq K$ successes. The* **negative hypergeometric distribution** *reflects the number of draws during this experiment, and its mean, denoted $\mu_{N,K,r}$, is given by*

$$\mu_{N,K,r} \coloneqq r\frac{N+1}{K+1} .$$

2.4 Interactive proofs

We start with the definition of SPs and then define the more general IPs.

Definition 2.15. *A* **SP for a relation** R **with randomness complexity** r *is a pair of probabilistic algorithms* $\mathsf{SP} = (\mathbf{V}_{\mathsf{SP}}, \mathbf{P}_{\mathsf{SP}})$ *such that, given access to an instance* $\mathbb{x}$*, the prover tries to convince the verifier that there exists a witness for* $\mathbb{x}$*. It is performed through an interactive protocol that works as follows.*

- *Both parties receive an instance* $\mathbb{x}$*, and the prover is given an additional input: a witness* $\mathbb{w}$*;*
- *The prover sends a message to the verifier, computed as* $\alpha_1 \leftarrow \mathbf{P}_{\mathsf{SP}}(\mathbb{x}, \mathbb{w})$*;*
- *The verifier responds with a random message* ρ*, computed as* $\rho \leftarrow \{0,1\}^{\mathsf{r}}$*;*
- *The prover responds to this challenge with the message* α_2*, computed as* $\alpha_2 \leftarrow \mathbf{P}_{\mathsf{SP}}(\mathbb{x}, \mathbb{w}, \alpha_1, \rho, \alpha_2)$*;*
- *Finally, the verifier decides deterministically if the prover knows a witness for* $\mathbb{x}$ *or not by returning a bit* b*, which is set to one if it is convinced.*

The verifier message is also called a **challenge***. We also consider malicious prover* $\tilde{\mathbf{P}}_{\mathsf{SP}}$*, which acts differently as* $\mathbf{P}_{\mathsf{SP}}$ *but still follows the protocol structure. The following notation captures the complete execution of the protocol when the prover is* $\tilde{\mathbf{P}}_{\mathsf{SP}}$*.*

$$b \xleftarrow{((\alpha_1,\alpha_2),\rho)} \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}), \mathbf{V}_{\mathsf{SP}}(\mathbb{x}) \rangle_{\mathsf{SP}} \ .$$

The tuple $((\alpha_1, \alpha_2), \rho)$ *is called the* **transcript** *of the execution. Moreover, we say that the transcript is an accepting transcript for* $\mathbb{x}$ *if the verifier is convinced, or in other words, if* $b = 1$*, where* b *is called the* **decision bit***. Finally, for* $i \in [2]$*, we define* pv_i *as the length of all possible* i*-th messages the prover can send.*

Note that all verifier's randomness is used to pick the challenge ρ. This a particular case of a *public-coin interactive proof*, introduced later.

By allowing the prover and verifier to send more messages, we obtain the more generic concept of IPs.

Definition 2.16. *An* **IP for a relation** R **with round complexity** k *is a pair of probabilistic algorithms* $\mathsf{IP} = (\mathbf{V}_{\mathsf{IP}}, \mathbf{P}_{\mathsf{SP}})$ *such that, given access to an instance* $\mathbb{x}$*, the prover tries to convince the verifier that there exists a witness for* $\mathbb{x}$*. The protocol works iteratively, as follows. For* i *going from* 1 *to* k*:*

- *The prover sends* α_i *to the verifier, computed as* $\alpha_i \leftarrow \mathbf{P}_{\mathsf{IP}}(\mathbb{x}, \mathbb{w}, (\alpha_1, \ldots, \alpha_{i-1}), (\rho_1, \ldots, \rho_{i-1}))$*;*
- *The verifier sends* ρ_i *to the verifier computed as* $\rho_i \leftarrow \mathbf{V}_{\mathsf{IP}}(\mathbb{x}, (\alpha_1, \ldots, \alpha_i), (\rho_1, \ldots, \rho_{i-1}))$*.*

Finally, the prover sends a last message $\alpha_{\mathsf{k}+1} \leftarrow \mathbf{P}_{\mathsf{IP}}(\mathbb{x}, \mathbb{w}, (\alpha_1, \ldots, \alpha_{\mathsf{k}}), (\rho_1, \ldots, \rho_{\mathsf{k}}))$ *and then the verifier returns a bit* b *which it sets to one if it is convinced. The tuple of messages*

$$((\alpha_1, \ldots, \alpha_{\mathsf{k}+1}), (\rho_1, \ldots, \rho_{\mathsf{k}})) \ ,$$

also denoted $(\boldsymbol{\alpha}, \boldsymbol{\rho})$*, sent during an execution of this protocol is called the* **transcript** *of this execution. We symbolize by*

$$b \xleftarrow{(\boldsymbol{\alpha},\boldsymbol{\rho})} \langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}; \xi), \mathbf{V}_{\mathsf{IP}}(\mathbb{x}; \boldsymbol{\rho}) \rangle_{\mathsf{IP}}$$

a complete execution of the protocol, where $(\boldsymbol{\alpha}, \boldsymbol{\rho})$ *is the transcript of the execution. Moreover, we say that the transcript is an accepting transcript for* $\mathbb{x}$ *if the verifier is convinced by the malicious prover* $\tilde{\mathbf{P}}_{\mathsf{IP}}$*, or in other words, if* $b = 1$*, where* b *is called the* **decision bit***. For every* $i \in [\mathsf{k}+1]$*, we define* pv_i *as the lengths of all possible* i*-th prover's messages.*

SPs and, more generally, IPs are abstract machines where both parties, the verifier and the prover, interact to show that a given instance $\mathbb{x}$ belongs to $L(R)$. In this model of computation, the prover has unlimited power while the verifier is efficient. If $\mathsf{IP} = (\mathbf{V}_{\mathsf{IP}}, \mathbf{P}_{\mathsf{IP}})$ is an IP, we say, respectively, that $\mathbf{V}_{\mathsf{IP}}$ and $\mathbf{P}_{\mathsf{IP}}$ are the *honest verifier* and *honest prover*. The word honest emphasizes that these parties rightly follow the protocol. On the other hand, we say that a prover is malicious if it does not follow the protocol IP. Such provers could act dishonestly to convince the verifier that $\mathbb{x}$ belongs to the language while it does not, which would break *soundness*, or try to convince that it knows a witness for $\mathbb{x}$ while it does not, which would break *knowledge soundness*. Soundness is defined below while we discuss knowledge soundness in Section 2.6. We often ask IPs to possess soundness and an additional property known as *completeness*. We define these two properties as follows.

1. **Completeness.** If $\mathbb{x} \in L(R)$, then the honest prover always convinces the honest verifier that $\mathbb{x} \in L(R)$.

2. **Soundness.** If $\mathbb{x} \notin L(R)$, then any malicious prover cannot convince the honest verifier, except with low probability, that $\mathbb{x} \in L(R)$.

These properties are necessary when designing cryptographic protocols; thus, they are usually required within the IP's definition. This thesis focuses on *special soundness*, which is stronger than standard soundness.

The notions of completeness, knowledge soundness, and special soundness are of interest in other computation models such as *probabilistically checkable proofs (PCPs)* and *IOPs*. The former are non-interactive models where the verifier is probabilistic and only looks at a bounded number of bits of the prover's message. The latter are cryptographic protocols that combine IPs and PCPs. More precisely, these protocols are interactive, like IPs, but instead of looking at every prover's message, the verifier only looks at parts of these messages, similarly to PCPs. For conciseness, we restrict our attention to IPs in this thesis. Our work extends nicely to IOP since we can naturally transform our extractors, defined over the world IPs, into extractors for IOPs.

Returning to our concrete example of the discrete logarithm problem over the group generated by g, for any generator g of prime order q, we introducethe so-called *Schnorr's protocol* [Sch90], which is a SP for $R_{\mathsf{dlog}}(g)$. The Schnorr protocol $\mathsf{SP}_{\mathsf{Schnorr}} = (\mathbf{V}_{\mathsf{SP}}, \mathbf{P}_{\mathsf{SP}})$ is defined as follows.

- The prover, given the instance h and the witness x, first picks $r \leftarrow \mathbb{Z}_q$ randomly and sends $\alpha_1 = g^r$ to the verifier;
- The verifier picks $\rho \in \mathbb{Z}_q$ and sends it to the prover;
- The prover computes and sends $\alpha_2 = r + \rho \cdot x$ to the verifier;
- The verifier checks that $g^{\alpha_2} = \alpha_1 \cdot h^{\rho}$.

This protocol is complete since if the prover is honest, *i.e.* $\alpha_2 = r + \rho \cdot x$ as expected by the protocol, then

$$g^{\alpha_2} = g^{r+\rho \cdot x} = g^r \cdot h^{\rho} = \alpha_1 \cdot h^{\rho} \ .$$

We do not prove that this protocol is sound; we will focus on the stronger notion of special soundness in Section 2.5.

The attentive reader might ask why, instead of the Schnorr protocol, we are not designing a simpler protocol where the prover only sends the value of the discrete logarithm. Indeed, such a protocol would

be complete and sound (assuming the discrete logarithm problem is hard in the group $\mathbb{G}$). Such protocols would work for *any NP relations*. One property we have omitted so far is *zero-knowledge*. In short, this security property ensures that the witness, in our case, the discrete logarithm, is not revealed to the verifier (or to anyone listening to their conversation). Our naive protocol is not zero-knowledge, but on the contrary, the Schnorr protocol *is* zero-knowledge. Even though zero-knowledge is a marvelous topic, we will not discuss it in this thesis.

On the contrary to SP, the IP verifier's messages are not necessarily strings picked uniformly at random, and the verifier can use randomness. However, in this thesis, we only focus on IP such that the verifier only sends random bit strings and is deterministic otherwise. These interactive proofs are called *public-coin interactive proofs*. From this point, we assume that all our IPs are public coin.

Definition 2.17. *An* **interactive proof** $\mathsf{IP} = (\mathbf{V}_{\mathsf{IP}}, \mathbf{P}_{\mathsf{IP}})$ **for a relation** R **with round complexity** k *is said to be* **public-coin** *if the i-th message of $\mathbf{V}_{\mathsf{IP}}$ is computed by the latter as $\rho_i \leftarrow \{0,1\}^{\mathsf{r}_i}$, that is, it is picked uniformly at random within strings of fixed length r_i. Moreover, we require that the verifier never flips coins except to pick its messages. The tuple $(\mathsf{r}_1, \ldots, \mathsf{r}_\mathsf{k})$ is called* **the randomness complexity** *of* IP.

Fig. 1 depicts an execution of an IP. The protocol proceeds from top to bottom. Each message is computed by both parties using the previous messages. If the protocol is public-coin with randomness complexity $(\mathsf{r}_1, \ldots, \mathsf{r}_\mathsf{k})$, then the verifier's messages are computed as shown after the double slashes in red.

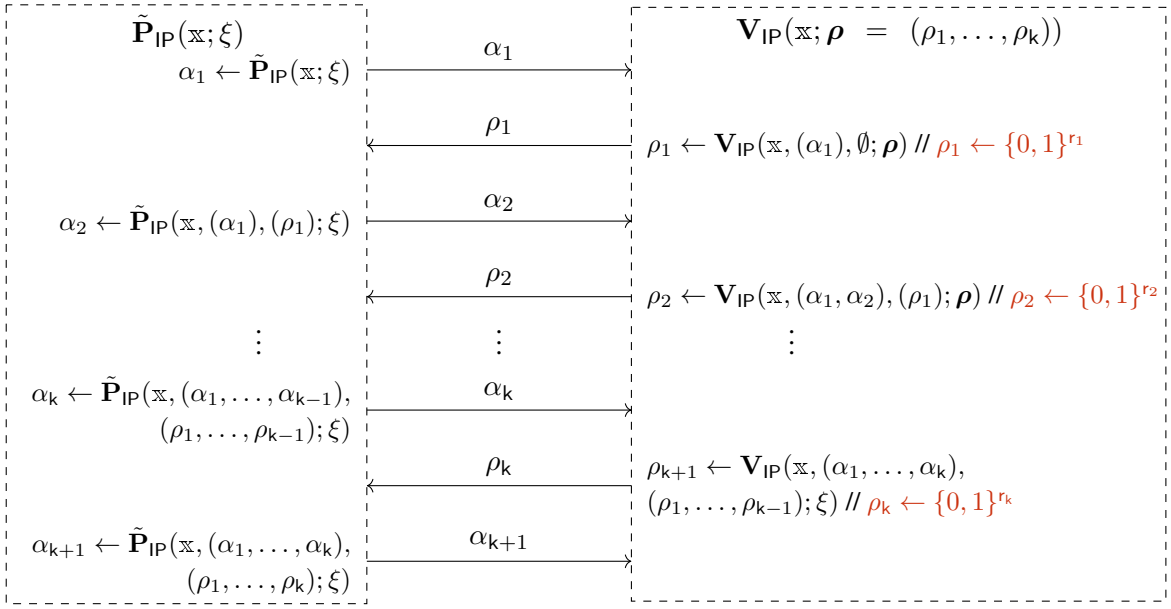

Figure 1: Execution of $\langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x};\xi), \mathbf{V}_{\mathsf{SP}}(\mathbb{x};\boldsymbol{\rho}) \rangle_{\mathsf{IP}}$. If the interactive proof is public coin, the verifier's challenges are computed as marked after the two slashes in red.

Later in this thesis, we will construct transcripts message by message. We define a notion that will be useful for this task.

Definition 2.18. *Given any transcript $((\alpha_1, \ldots, \alpha_{\mathsf{k}+1}), (\rho_1, \ldots, \rho_\mathsf{k}))$, any tuple $((\alpha_1, \ldots, \alpha_i), (\rho_1, \ldots, \rho_i))$, for $i \in [\mathsf{k}]$ is called a* **partial transcript**.

One may ask how to capture that the two parties of an IP can have an intermediate internal state between

the messages they send. Indeed, the notation introduced in Definition 2.16 lacks flexibility in this regard. Thankfully, our current notation is sufficient for our specific needs for three reasons:

1. We will never need to execute a verifier step by step; hence, we only need to capture the internal state of provers;

2. If the internal state of the prover before sending the message i only depends on the partial transcript $((\alpha_1, \ldots, \alpha_{i-1}), (\rho_1, \ldots, \rho_{i-1}))$ and the instance $\mathbb{x}$, then $\mathbf{P}_{\mathsf{IP}}(\mathbb{x}, \mathbb{w}, (\alpha_1, \ldots, \alpha_{i-1}), (\rho_1, \ldots, \rho_{i-1}))$ can retrieve its previous internal state; and

3. We only consider public-coin IP, and therefore, if the prover's internal state depends on some coin tossing, the result of the latter must be available in the partial transcript and, thus, to the malicious prover by the previous point.

These three reasons also apply to malicious provers.

Public-coin IPs can be compilable to a non-interactive proof through the so-called *Fiat–Shamir transformation* [FS86]. It transforms the non-practical interactive proof into a single message proof. This transformation requires random oracles; hence, the resulting non-interactive proof is a *NIROP*. NIROPs are protocols consisting of only one message; the message sent by the prover. Moreover, both the verifier and the prover may use random oracles. The Fiat–Shamir transformation makes public-coin protocols an essential subclass of interactive proofs.

At a higher level, the Fiat–Shamir transformation compiles *abstract model of computations* into *concrete protocols in the ROM*. In these regards, variants of this transform, such as the *Ben-SassonChiesaSpooner (BCS)* [BCS16] and *Micali* [Mic00] transformations, have been developed by researchers to respectively compile PCPs and IOPs into concrete, non-interactive, cryptographic protocols in the ROM, or NIROP for short. As we will see later, most of the definitions exposed in this thesis extend naturally to these two models and their transformations.

We will not use the Fiat–Shamir transformation in this thesis; our main point is to introduce *state restoration*, which will serve as a proxy to prove the security of the Fiat–Shamir transformation by removing any need to work in the world of NIROPs. Nevertheless, we still present this transformation for the sake of explanation.

Definition 2.19. *For every salt size $s \in \mathbb{N}$, the* **Fiat–Shamir transformation** *of a public-coin interactive protocol $\mathsf{IP} = (\mathbf{V}_{\mathsf{IP}}, \mathbf{P}_{\mathsf{IP}})$ with round complexity k for a relation R, denoted by $\mathsf{FS}[\mathsf{IP}]$, is tuple $\mathsf{NIROP} = (\mathbf{P}_{\mathsf{NIROP}}, \mathbf{V}_{\mathsf{NIROP}})$ where $\mathbf{P}_{\mathsf{NIROP}}$ and $\mathbf{V}_{\mathsf{NIROP}}$ are probabilistic random-oracle algorithms.* NIROP *is known as a non-interactive oracle proof. These algorithms are constructed as follows:*

- *Both algorithms are given oracle access to k random oracles $\mathsf{rnd}_1, \ldots, \mathsf{rnd}_{\mathsf{k}}$ with respective output size $\mathsf{r}_1, \ldots, \mathsf{r}_{\mathsf{k}}$;*

- *Instead of querying $\mathbf{V}_{\mathsf{IP}}$ to get the i-th challenge, $\mathbf{P}_{\mathsf{NIROP}}$ first sample the salt $\sigma_i \in \{0,1\}^s$ and then compute $\rho_i \coloneqq \mathsf{rnd}_i(\mathbb{x}, (\alpha_1, \ldots, \alpha_i), (\sigma_1, \ldots, \sigma_i))$;*

- *The proof is $\pi \coloneqq ((\alpha_1, \ldots, \alpha_k), (\sigma_1, \ldots, \sigma_{\mathsf{k}}))$;*

- *The verifier $\mathbf{V}_{\mathsf{NIROP}}$ simulate the interaction between $\mathbf{P}_{\mathsf{IP}}$ and $\mathbf{V}_{\mathsf{IP}}$ by using the messages prover messages in π and its random oracle $\mathsf{rnd}_1, \ldots, \mathsf{rnd}_{\mathsf{k}}$ to produce challenges that are consistent with messages in π.*

Remark 2.20. If the output size of rnd_i is not larger than r_i, which can happen when realizing the random oracles into concrete hash functions, then we can call rnd_i multiple times to have enough random bit to pick ρ_i.

Our definition of the Fiat–Shamir transformation includes *salt strings*, which might differ from what one might find in literature. Salt strings are a well-known cryptographic tool that protects cryptographic primitives against pre-computation attacks such as *rainbow tables*. We illustrate in Fig. 2 the Fiat–Shamir transformation *with salt strings*, highlighting the main differences with the original IP with colors.

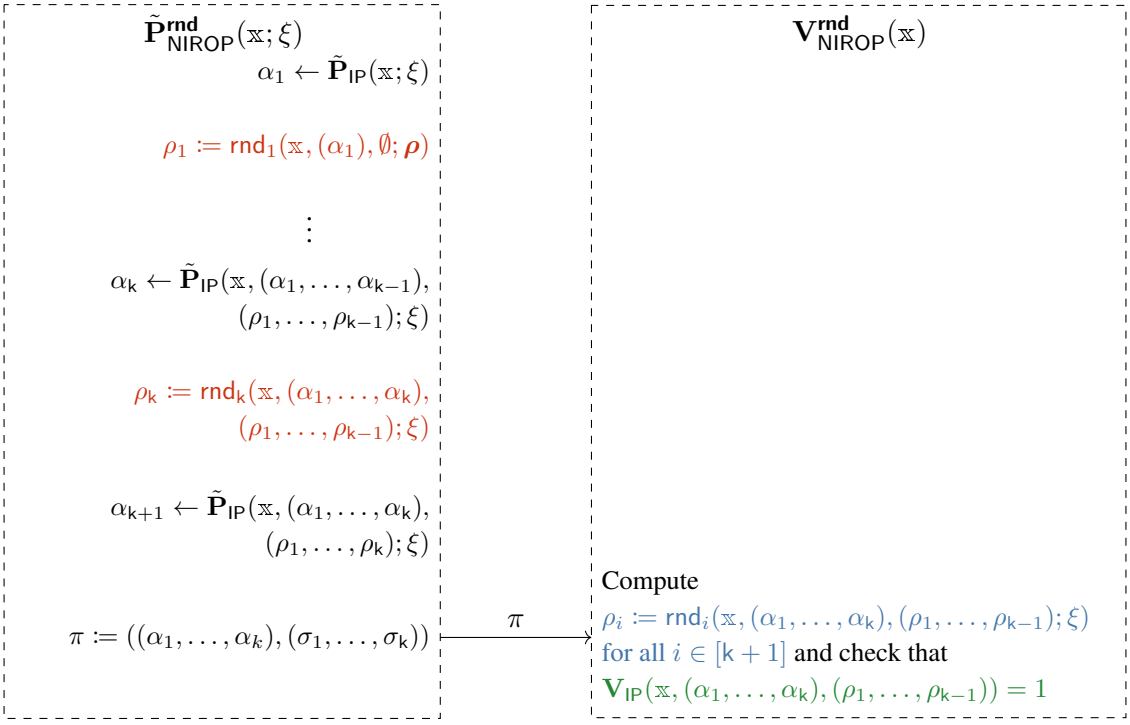

Figure 2: NIROP resulting from the Fiat–Shamir transformation of $\mathsf{IP} = (\mathbf{P}_{\mathsf{IP}}, \mathbf{V}_{\mathsf{IP}})$. The uses of random oracles are highlighted in red, the check for the consistency of the verifier's messages is in blue, and the acceptance check is in green.

Let $\mathsf{IP} = (\mathbf{V}_{\mathsf{IP}}, \mathbf{P}_{\mathsf{IP}})$ be any public coin interactive proof for a relation R with round complexity k. Throughout these notes, we mostly focus on deterministic malicious prover. Note that if $\tilde{\mathbf{P}}_{\mathsf{IP}}$ is a deterministic malicious prover for IP, its first i messages depend only on $\mathbb{x}$ and the first i verifier's messages. This observation motivates the following notation.

Notation 8. *For every $i \in [\mathsf{k}+1]$, deterministic malicious prover $\tilde{\mathbf{P}}_{\mathsf{IP}}$, problem instance $\mathbb{x}$ and i first verifier's messages $(\rho_1,\ldots,\rho_i)$, we denote by*

$$(\alpha_1,\ldots,\alpha_i) \leftarrow \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x},(\rho_1,\ldots,\rho_{i-1}))$$

the i first prover's messages.

In the case of deterministic malicious prover, fixing the k verifier's messages $\rho_1,\ldots,\rho_{\mathsf{k}}$, does not only fix all prover's messages but also the protocol's instantiation's decision bit.

Definition 2.21. *Let* $\mathsf{IP} = (\mathbf{V}_{\mathsf{IP}}, \mathbf{P}_{\mathsf{SP}})$ *be an* **IP for a relation** R **with round complexity** k *is a pair of algorithms and* $\tilde{\mathbf{P}}_{\mathsf{IP}}$ *any deterministic malicious prover for* IP*. Given any verifier's messages* $(\rho_1, \ldots, \rho_{\mathsf{k}})$*, we say that it is* **accepting for** $\mathbb{x}$ *if*

$$1 \leftarrow \langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}), \mathbf{V}_{\mathsf{IP}}(\mathbb{x}; (\rho_1, \ldots, \rho_{\mathsf{k}})) \rangle_{\mathsf{IP}} \ .$$

In these notes, we use the concept of an *extractor*. The notion of *knowledge soundness* is formally defined using this idea. Knowledge soundness captures that when the prover, or the malicious prover, convinces the verifier that a specific instance is in the language, it should know a witness for the instance. We will discuss this concept in detail later Section 2.6.

To sum up, the three types of algorithms we will use are:

1. Verifiers;
2. Honest and malicious provers; and
3. Extractors.

We use three distinct symbols to refer to the randomness of these algorithms.

Notation 9. *We denote by* ξ*,* ρ*, and* ν *the respective randomness of the prover or malicious prover, the verifier, and the extractor. Moreover, we symbolize their respective random spaces by* $\mathcal{X}$*,* $\mathcal{R}$*, and* $\mathcal{N}$*. These notations generalize naturally to the multi-round case. Especially if the protocol has* $2\mathsf{k} + 1$ *moves, the verifier's randomness has the form* $\mathcal{R} = \mathcal{R}_1 \times \cdots \times \mathcal{R}_{\mathsf{k}}$*, where for every* $i \in [\mathsf{k}]$*,* $\mathcal{R}_i \coloneqq \{0,1\}^{\mathsf{r}_i}$ *is the random space of the challenge* ρ_i*. Moreover, we let* $N_i \coloneqq |\mathcal{R}_i| = 2^{\mathsf{r}_i}$ *be the size of these sets. In the multi-round case, to highlight that* $\rho \in \mathcal{R}$ *is a tuple, we use* $\boldsymbol{\rho} \in \boldsymbol{\mathcal{R}}$ *instead, where* $\boldsymbol{\mathcal{R}} = \mathcal{R}_1 \times \cdots \times \mathcal{R}_{\mathsf{k}}$*. Generally, we will denote tuples and Cartesian products of sets with bold symbols.*

We can think of an interactive protocol as a probabilistic algorithm taking as a parameter an instance and as randomness the combination of the prover's and verifier's randomness. Therefore, taking inspiration from complexity theory, we can define the *running time* of a malicious prover as follows.

Definition 2.22. *Let* $\mathsf{IP} = (\mathbf{P}_{\mathsf{IP}}, \mathbf{V}_{\mathsf{IP}})$ *be an interactive proof. For every prover or malicious prover* $\tilde{\mathbf{P}}_{\mathsf{IP}}$*, we define its* **running time** *to be*

$$\tau_{\tilde{\mathbf{P}}_{\mathsf{IP}}} \coloneqq \max_{\mathbb{x}, \xi, \boldsymbol{\rho}} \tau_{\langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}; \xi), \mathbf{V}_{\mathsf{IP}}(\mathbb{x}; \boldsymbol{\rho}) \rangle_{\mathsf{IP}})} \ ,$$

where $\tau_{\langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}; \xi), \mathbf{V}_{\mathsf{IP}}(\mathbb{x}; \boldsymbol{\rho}) \rangle_{\mathsf{IP}})}$ *denotes the running time to execute* $\langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}; \xi), \mathbf{V}_{\mathsf{IP}}(\mathbb{x}; \boldsymbol{\rho}) \rangle_{\mathsf{IP}}$.

Note that the granularity of this time operator is the whole protocol's execution, not a single round nor a single message, so it coincides with our future definition of *invocation*.

2.5 Special soundness

In the SP case, special soundness characterizes the fact that given multiple accepting transcripts that start with the same message but have different challenges, we can efficiently recover a witness for the instance. A *fork* captures these transcripts' structure.

Definition 2.23. *Let* $\mathsf{SP} = (\mathbf{P}_{\mathsf{SP}}, \mathbf{V}_{\mathsf{SP}})$ *be a SP and* $a \in \mathbb{N}$ *an arity parameter. A tuple*

$$\mathbf{F} = \left(\alpha_1, \left((\rho^i, \alpha_2^i) \right)_{i \in [a]} \right)$$

is a a**-fork** *for an instance* $\mathbb{x}$ *if*

- α_1 *is a prover first message;*
- $(\rho^i)_{i\in[a]}$ *are pairwise distinct verifier challenges;*
- $(\alpha_2^i)_{i\in[a]}$ *are prover responses; and*
- $\mathbf{V}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho^i, \alpha_2^i) = 1$ *for all* $i \in [a]$.

Extending this definition to interactive proof with more rounds induces trees.

Definition 2.24. *A* $(a_1, \dots a_{\mathsf{k}})$**-tree of transcripts for an instance** $\mathbb{x}$ *is a tree with the following structure. The nodes in the tree correspond to the prover messages and edges to the verifier's challenges when the protocol is executed on* $\mathbb{x}$*. At depth* i*, each node has* a_i *children connected through edges corresponding to* a_i *distinct verifier's challenges. If* $\mathsf{k} = 1$*, we simply write* a*-tree instead of* (a)*-tree. Moreover, we say a tree of transcripts for* $\mathbb{x}$ *is an* **accepting tree** *if all transcripts, which are paths from the root to any leaves, are accepting. We do not explicitly write the dependency on* $\mathbf{V}_{\mathsf{IP}}$ *and* $\tilde{\mathbf{P}}_{\mathsf{IP}}(\xi)$.

A useful class of trees of transcript is the set of trees with a given *prefix*.

Definition 2.25. *Given any public-coin interactive proof with round complexity* k *and randomness complexity* $(\mathsf{r}_1, \dots, \mathsf{r}_{\mathsf{k}})$*, we call, for every* $i \in [\mathsf{k}]$*, any tuple*

$$(\rho_1, \dots, \rho_i) \in \{0,1\}^{\mathsf{r}_1} \times \cdots \times \{0,1\}^{\mathsf{r}_i}$$

a **prefix**. *Moreover, we refer to any* $(1, \dots, 1, a_{i+1}, \dots, a_{\mathsf{k}})$*-tree of transcripts that start with the challenges* $(\rho_1, \dots, \rho_i)$ *as an* $(a_i, \dots, a_{\mathsf{k}})$**-tree of transcripts with prefix** $(\rho_1, \dots, \rho_i)$.

We now define special soundness.

Definition 2.26. *A public-coin interactive proof* $\mathsf{IP} = (\mathbf{P}_{\mathsf{IP}}, \mathbf{V}_{\mathsf{IP}})$ *with round complexity* k *for a relation* R *has* $(a_1, \dots, a_{\mathsf{k}})$-**special soundness** *if there exists a deterministic polynomial time algorithm* $\mathbf{E}_{\mathsf{IP}}^{\mathsf{SS}}$ *(the extractor) such that for every* $\mathbb{x}$ *and* $(a_1, \dots, a_{\mathsf{k}})$*-tree* T *of accepting transcripts for* $\mathbb{x}$ *we have* $(\mathbb{x}, \mathbf{E}_{\mathsf{IP}}^{\mathsf{SS}}(\mathbb{x}, T)) \in R$.

In the particular case of a-special soundness for SP, the tree given to the extractor is represented through an a-fork. Moreover, we also say special soundness with arity a in that case.

We now prove that our running example, the Schnorr protocol $\mathsf{SP}_{\mathsf{Schnorr}}$, is 2-special sound.

Lemma 2.27. *Let* g *be a generator of a group of prime order* q*. Then the Schnorr protocol* $\mathsf{SP}_{\mathsf{Schnorr}}$ *for* $R_{\mathsf{dlog}}(g)$ *is* 2*-special sound.*

Proof. Pick arbitrarily any $h \in \mathbb{G}$, where $\mathbb{G}$ is the group generated by g. Let $(\alpha_1, \rho, \alpha_2)$ and $(\alpha_1, \rho', \alpha_2')$ be two accepting transcripts for h where $\rho \neq \rho'$. Their first message is identical, and their challenges are distinct, making $(\alpha_1, ((\rho, \alpha_2), (\rho', \alpha_2')))$ a 2-fork for h. There exists a unique $r \in \mathbb{Z}_q$ such that $\alpha_1 = g^r$ by definition of a generator. Remark now that

$$\frac{\alpha_2' - \alpha_2}{\rho' - \rho} = (r - r + x \cdot (\rho' - \rho)) \cdot (\rho' - \rho)^{-1} = x \ ,$$

since $\rho' \neq \rho$ and where the inverse is taken in $\mathbb{Z}_q$. It follows from this equation that

$$g^{\frac{\alpha_2' - \alpha_2}{\rho' - \rho}} = g^{x \cdot (\rho' - \rho)} \cdot \left(g^{(\rho' - \rho)}\right)^{-1} = g^x \ ,$$

where we take the inverse in $\mathbb{G}$. Therefore, by the definition of a generator, we must have

$$\left(\rho' - \rho\right)^{-1} \cdot \left(\alpha_2' - \alpha_2\right) \frac{\alpha_2' - \alpha_2}{\rho' - \rho} = x \ .$$

The inverse of $\rho' - \rho$ exists since we are working on the group of prime order $\mathbb{Z}_q$ and the difference is not zero. Moreover, we can compute it efficiently with the *extended Euclidean algorithm*. Thus, we have constructed an efficient extractor for $\mathsf{SP}_{\mathsf{Schnorr}}$. □

Note that by definition of special soundness, this proof relies solely on the combinatorial property of the protocol. According to the results of this thesis, Lemma 2.27 is enough to show that Schnorr's protocol is knowledge sound and state restoration knowledge sound. However, we are still interested in proving the former to compare and discuss their proofs' complexities.

The notion of special soundness might seem abstract and not practical. However, it has two essential priorities, making it a vital property of an IP. Firstly, special soundness implies classical soundness. Secondly, special soundness implies *knowledge soundness* and *state restoration knowledge soundness*, which are security properties we exhibit below. Finally, special soundness is usually simpler to demonstrate than knowledge soundness and state restoration knowledge soundness.

We end this section with a word about *post-quantum security*. In general, protocols that are special sound in the ROM are not special sound under its quantum version, the *quantum random oracle model (QROM)* [BDFLSZ11; YZ22]. However, if an IP or a PCP is special sound, then its respective Fiat–Shamir and Micali transformation, which we prove to be special sound in the ROM, is also special sound in the QROM. See [DFMS19; LZ19] for the proof regarding the Fiat-Shamir transformation and [CMS19] for the Micali transformation. Hence, the results that we prove in this thesis also hold against quantum adversaries in the QROM. The picture is more complicated regarding IOP. The implication holds, for its respective BCS transformation, when the IP is *round-by-round sound*, which is stronger than special soundness. We do not discuss round-by-round soundness in this thesis [CMS19].

2.6 Knowledge soundness

Let $\mathsf{IP} = (\mathbf{P}_{\mathsf{IP}}, \mathbf{V}_{\mathsf{IP}})$ be an interactive proof for R. Knowledge soundness is a property of an IP that asserts that when a prover, or malicious prover, convinces the verifier that an instance $\mathbb{x}$ has a witness $\mathbb{w}$ in the relation R, it knows a witness for $\mathbb{x}$. Note the subtle difference with the notion of special soundness; special soundness asserts that when the prover, given $\mathbb{x}$, convinces the verifier then $\mathbb{x}$ is indeed in $L(R)$.

Defining the notion of knowledge soundness is a tricky problem. Indeed, how can we formulate the knowledge of an algorithm? Given what it has seen, we define the algorithm's knowledge as everything it can compute in polynomial time. In other words, what an algorithm *does not know* is anything it cannot compute efficiently using the information provided to it. Therefore, to show that a prover, or a malicious prover $\tilde{\mathbf{P}}_{\mathsf{IP}}$, knows a witness, it is sufficient to construct an algorithm with a black box access to $\tilde{\mathbf{P}}_{\mathsf{IP}}$ that can retrieve a witness for $\mathbb{x}$. This algorithm is known as the *extracor*.

Definition 2.28. *A public-coin interactive proof* $\mathsf{IP} = (\mathbf{P}_{\mathsf{IP}}, \mathbf{V}_{\mathsf{IP}})$ *with round complexity* k *for a relation* R *has* **weak rewinding knowledge soundness error** κ_{IP} **with extraction time** $\mathsf{et}_{\mathsf{IP}}$ *if there exists a polynomial* $q\colon \mathbb{N} \to \mathbb{N}$ *and a probabilistic algorithm* $\mathbf{E}_{\mathsf{IP}}^{\mathsf{KS}}$*, called a knowledge extractor, with the following properties: The extractor* $\mathbf{E}_{\mathsf{IP}}^{\mathsf{KS}}$*, given an input* $\mathbb{x}$ *and rewinding oracle access to any prover* $\tilde{\mathbf{P}}_{\mathsf{IP}}$ *runs in* **expected time** $\mathsf{et}_{\mathsf{IP}}(\mathbb{x}, \tilde{\mathbf{P}}_{\mathsf{IP}})$ *over its internal randomness and the prover's randomness, and whenever*

$$\Pr\left[b = 1 \;\middle|\; b \leftarrow \langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}), \mathbf{V}_{\mathsf{IP}}(\mathbb{x}) \rangle_{\mathsf{IP}}\right] := \epsilon(\mathbb{x}) \geq \kappa_{\mathsf{IP}}(\mathbb{x}) \ ,$$

we have

$$\Pr\left[(\mathbb{x},\mathbb{w})\in R \;\middle|\; \mathbb{w}\leftarrow \mathbf{E}^{\mathsf{KS}}_{\mathsf{IP}}(\mathbb{x},\tilde{\mathbf{P}}_{\mathsf{IP}})\right]\geq \frac{\epsilon(\mathbb{x})-\kappa_{\mathsf{IP}}(\mathbb{x})}{q(|\mathbb{x}|)}\ .$$

Rewinding oracle access *allows the extractor to rewind the malicious prover* $\tilde{\mathbf{P}}_{\mathsf{IP}}$ *to any previous state.*

Remark that this definition depends on the probability that the malicious convinces the verifier on input $\mathbb{x}$. Indeed, if this probability is negligible for a given $\mathbb{x}$ and malicious prover, then we cannot hope for the extractor to retrieve a witness with high probability. If you want to convince yourself, think about a dumb prover always sending random messages. Note also that the probability expression $\epsilon(\mathbb{x})$ does not specifically consider each first prover message but instead samples over all possible prover's randomness. We could be interested in a more fine-grained variant where we expressly ask for the success probability of the extractor to depend on what transcript we are currently attacking. Such an improvement would be a first step toward *adpative* security, which considers malicious provers that choose the problem instance.

It may seem like this definition contradicts the notion of *zero-knowledge* since any malicious verifier could run this extractor against the honest prover and retrieve a witness for $\mathbb{x}$, therefore breaking the zero-knowledge property. However, the extractor has an extra ability which is not available to the malicious verifier. In the definition above, this additional power is that the extractor has *rewinding* access to the malicious prover.

One may ask why we do not simply define weak knowledge soundness as a boolean property of an IP, which holds if and only if the knowledge soundness error is negligible in the security parameter. The problem with such a formulation is that it would be impossible to set the security parameter of the cryptographic primitive to ensure that the winning probability is below a particular constant; hence, we must define the knowledge error as a non-asymptotic function of the different parameters. Nevertheless, we still use the informal statement "IP is weak knowledge sound" to designate that its weak knowledge error is "small". We will use this abuse of notation with all the security definitions we introduce throughout this thesis, but all results are stated formally with the error function expressed explicitly.

So far, we have considered *non-adaptive* provers, meaning that the instance $\mathbb{x}$ is fixed and not chosen by any party. However, real-life protocols are usually adaptive; provers adaptively select and send the instance to the verifier. Our previous definition of knowledge soundness universally quantifies over every $\mathbb{x}$, which could not reflect the adaptive nature of such malicious provers. To extend this definition to adaptive protocols, we need a more robust experiment that considers the messages the malicious prover sends. The idea is to transform the probability experiment of Definition 2.28 to take into account the transcript of the execution of the protocol.

Definition 2.29. *A public-coin interactive proof* $\mathsf{IP}=(\mathbf{P}_{\mathsf{IP}},\mathbf{V}_{\mathsf{IP}})$ *with round complexity* k *for a relation* R *has* **strong rewinding knowledge soundness error***, or simply* **knowledge soundness error***,* κ_{IP} **with extraction time** $\mathsf{et}_{\mathsf{IP}}(\mathbb{x},\tilde{\mathbf{P}}_{\mathsf{IP}})$ *if there exists a probabilistic algorithm* $\mathbf{E}^{\mathsf{SKS}}_{\mathsf{IP}}$ *(the extractor) such that the following holds: for every instance* $\mathbb{x}$ *and malicious prover* $\tilde{\mathbf{P}}_{\mathsf{IP}}$ *we have*

$$\Pr\left[\begin{array}{l|lll}(\mathbb{x},\mathbb{w})\notin R & \xi & \leftarrow & \mathcal{X}\\ \wedge\, b=1 & b & \xleftarrow{(\boldsymbol{\alpha},\boldsymbol{\rho})} & \langle\tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x};\xi),\mathbf{V}_{\mathsf{IP}}(\mathbb{x})\rangle_{\mathsf{IP}}\\ & \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SKS}}_{\mathsf{IP}}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\rho},\tilde{\mathbf{P}}_{\mathsf{IP}}(\xi))\end{array}\right]\leq\kappa_{\mathsf{IP}}(\mathbb{x})\ .$$

Additionally, $\mathbf{E}^{\mathsf{SKS}}_{\mathsf{IP}}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\rho},\tilde{\mathbf{P}}_{\mathsf{IP}}(\xi))$ *runs in expected time* $\mathsf{et}_{\mathsf{IP}}(\mathbb{x},\tilde{\mathbf{P}}_{\mathsf{IP}})$ *over* ξ*,* $\boldsymbol{\rho}$ *and* ν*.*

Remark 2.30. We can restrict ourselves to deterministic malicious provers in both weak and strong knowledge soundness. Indeed, if $\tilde{\mathbf{P}}_{\mathsf{IP}}$ is probabilistic, we could use the law of total probability to condition over its randomness.

If the public-coin interactive proof is a SP, the probability expression of the previous definition has the form

$$\Pr\left[\begin{array}{l|lll} & \xi & \leftarrow & \mathcal{X} \\ (\mathbb{x},\mathbb{w}) \notin R & \rho & \leftarrow & \mathcal{R} \\ \wedge\, b = 1 & b & \xleftarrow{((\alpha_1,\alpha_2),\rho)} & \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x};\xi), \mathbf{V}_{\mathsf{SP}}(\mathbb{x};\rho)\rangle_{\mathsf{SP}} \\ & \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}(\mathbb{x},\alpha_1,\rho,\alpha_2,\tilde{\mathbf{P}}_{\mathsf{SP}}(\xi)) \end{array}\right] \leq \kappa_{\mathsf{SP}}(\mathbb{x}) \ ,$$

In this strengthening of knowledge soundness, the extra power given to the extractor *is not* rewinding oracle access but that the randomness of the malicious prover is *hard-coded in its code*. Indeed, in real life, the prover picks a fresh randomness every time the protocol runs. Hence, the verifier could not get consistent messages from the malicious prover even if it fixes the problem instance $\mathbb{x}$ and its randomness ρ. On the other hand, if the randomness of the prover given to the extractor is hard-coded in its code, then we can see this prover has a deterministic function. Thus, the extractor can obtain consistent messages when the extractor fixes $\mathbb{x}$ and ρ. As we will prove with Theorem 2.32, strong knowledge soundness implies weak knowledge soundness.

Soundness and knowledge soundness are convenient in other computation models such as PCPs and IOPs. They have natural variants for these models that capture the same security notions.

We now show that our running example, the Schnorr protocol, is knowledge sound.

Lemma 2.31. *Let g be a generator of a group of prime order q. Then the Schnorr protocol $\mathsf{SP}_{\mathsf{Schnorr}}$ for $R_{\mathsf{dlog}}(g)$ has rewinding knowledge soundness error $\kappa_{\mathsf{SP}} = \frac{1}{q}$.*

Proof. Let $\mathsf{SP}_{\mathsf{Schnorr}} = (\mathbf{P}_{\mathsf{SP}}, \mathbf{V}_{\mathsf{SP}})$ be the Schnorr protocol and $\mathbf{E}^{\mathsf{SS}}_{\mathsf{SP}}$ the special sound extractor that we have constructed in the proof of Lemma 2.27. Fix an instance $h \in \mathbb{G}$ and a malicious prover $\tilde{\mathbf{P}}_{\mathsf{SP}}$ arbitrarily. Consider the following extractor.

$\mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}(h,\alpha_1,\rho,\alpha_2,\tilde{\mathbf{P}}_{\mathsf{SP}})$:
1. If $\mathbf{V}_{\mathsf{SP}}(h,\alpha_1,\rho,\alpha_2) = 0$, abort.
2. Pick a random challenge $\rho' \leftarrow \mathbb{Z}_q$.
3. Get the malicious prover response $\alpha_2' \leftarrow \tilde{\mathbf{P}}_{\mathsf{SP}}(h,\alpha_1,\rho')$.
4. Get the discrete logarithm $x \leftarrow \mathbf{E}^{\mathsf{SS}}_{\mathsf{SP}}(\alpha_1,((\rho,\alpha_2),(\rho',\alpha_2')))$.
5. Return (h,x).

Inserting this extractor into the knowledge soundness probability expression leads to:

$$\Pr\left[\begin{array}{l|lll} & \xi & \leftarrow & \mathcal{X} \\ (h,x) \notin R_{\mathsf{dlog}}(g) & \rho & \leftarrow & \mathbb{Z}_q \\ \wedge\, b = 1 & b & \xleftarrow{((\alpha_1,\alpha_2),\rho)} & \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(h;\xi), \mathbf{V}_{\mathsf{SP}}(h;\rho)\rangle_{\mathsf{SP}} \\ & \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}(h,\alpha_1,\rho,\alpha_2,\tilde{\mathbf{P}}_{\mathsf{SP}}(\xi)) \end{array}\right]$$

$$= \Pr\left[\begin{array}{l|lll} & \xi & \leftarrow & \mathcal{X} \\ (h,x) \notin R_{\mathsf{dlog}}(g) & \rho & \leftarrow & \mathbb{Z}_q \\ \text{conditioned on} & b & \xleftarrow{((\alpha_1,\alpha_2),\rho)} & \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(h;\xi), \mathbf{V}_{\mathsf{SP}}(h;\rho)\rangle_{\mathsf{SP}} \\ b = 1 & x & \leftarrow & \mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}(h,\alpha_1,\rho,\alpha_2,\tilde{\mathbf{P}}_{\mathsf{SP}}(\xi)) \end{array}\right]$$

$$\cdot \Pr\left[b = 1 \;\middle|\; \begin{array}{lll} \rho & \leftarrow & \mathcal{R} \\ b & \xleftarrow{((\alpha_1,\alpha_2),\rho)} & \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(h;\xi), \mathbf{V}_{\mathsf{SP}}(h;\rho)\rangle_{\mathsf{SP}} \end{array}\right]$$

$$
\begin{aligned}
&\leq \Pr\left[\begin{array}{l|ccc} & \xi & \leftarrow & \mathcal{X} \\ & \rho & \leftarrow & \mathbb{Z}_q \\ (h,x) \notin R_{\mathsf{dlog}}(g) & b & \xleftarrow{((\alpha_1,\alpha_2),\rho)} & \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(h;\xi), \mathbf{V}_{\mathsf{SP}}(h;\rho)\rangle_{\mathsf{SP}} \\ \text{conditioned on} & \rho' & \leftarrow & \mathbb{Z}_q \\ b = 1 & \alpha_2' & \leftarrow & \tilde{\mathbf{P}}_{\mathsf{SP}}(h,\alpha_1,\rho') \\ & x & \leftarrow & \mathbf{E}^{\mathsf{SS}}_{\mathsf{SP}}(\alpha_1,((\rho,\alpha_2),(\rho',\alpha_2'))) \end{array}\right] \\
&\leq \Pr\left[\begin{array}{l|ccc} & \xi & \leftarrow & \mathcal{X} \\ & \rho & \leftarrow & \mathbb{Z}_q \\ (h,x) \notin R_{\mathsf{dlog}}(g) & b & \xleftarrow{((\alpha_1,\alpha_2),\rho)} & \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(h;\xi), \mathbf{V}_{\mathsf{SP}}(h;\rho)\rangle_{\mathsf{SP}} \\ \text{conditioned on} & \rho' & \leftarrow & \mathbb{Z}_q \\ b = 1 & \alpha_2' & \leftarrow & \tilde{\mathbf{P}}_{\mathsf{SP}}(h,\alpha_1,\rho') \\ \wedge\, b' = 1 & b' & \leftarrow & \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(h;\xi), \mathbf{V}_{\mathsf{SP}}(h;\rho')\rangle_{\mathsf{SP}} \\ & x & \leftarrow & \mathbf{E}^{\mathsf{SS}}_{\mathsf{SP}}(\alpha_1,((\rho,\alpha_2),(\rho',\alpha_2'))) \end{array}\right] \\
&\leq \Pr\left[\rho = \rho' \;\middle|\; \begin{array}{ccc} \rho & \leftarrow & \mathbb{Z}_q \\ \rho' & \leftarrow & \mathbb{Z}_q \end{array}\right] \\
&= \frac{1}{q} \ .
\end{aligned}
$$

The last inequality follows from the fact that if $b = 1$, $b' = 1$ and $\rho \neq \rho'$then the second argument of $\mathbf{E}^{\mathsf{SS}}_{\mathsf{SP}}$ is a 2-fork for h, and thus, by the property of $\mathbf{E}^{\mathsf{SS}}_{\mathsf{SP}}$, it returns x such that $g^x = h$. □

First, note that this proof relies on the proof of special soundness, making it strictly more complicated than the latter. Moreover, we must consider that the messages sent by the malicious prover do not necessarily follow the protocol definition. In this example, we only consider cases where the malicious prover convinces the verifier. This does not imply that the malicious prover has followed the protocol. Still, in our case, $b = 1$ and $b' = 1$ respectively show that $g^{\alpha_2} = \alpha_1 \cdot h^{\rho}$ and $g^{\alpha_2'} = \alpha_1 \cdot h^{\rho'}$, which allows retrieving x if $\rho \neq \rho'$ as demonstrated in the proof of Lemma 2.27. The same problem arises when proving state restoration knowledge soundness or proving that the Fiat–Shamir transformation is knowledge sound.

As their names suggest, strong knowledge soundness implies weak knowledge soundness. We formalize this statement in the following theorem.

Theorem 2.32. *Let* $\mathsf{IP} = (\mathbf{P}_{\mathsf{IP}}, \mathbf{V}_{\mathsf{IP}})$ *be a public-coin interactive proof for a relation* R *with round complexity* k *strong rewinding knowledge soundness error* κ_{IP} *and extraction time* $\mathsf{et}_{\mathsf{IP}}$*. Then, it is weak rewinding knowledge soundness with the same knowledge soundness error and extraction time* $\mathsf{et}'_{\mathsf{IP}}(\mathbb{x}, \tilde{\mathbf{P}}_{\mathsf{IP}}) = \Theta(\mathsf{k} + \tau_{\tilde{\mathbf{P}}_{\mathsf{IP}}}) + \mathsf{et}_{\mathsf{IP}}(\mathbb{x}, \tilde{\mathbf{P}}_{\mathsf{IP}})$.

We defer the proof of this statement in Appendix A to be more concise.

As discussed previously, public-coin IPs are compilable into no interactive proof employing the Fiat–Shamir transformation. The security loss after applying the Fiat–Shamir transformation depends on the number of times the malicious prover calls the random oracle. Moreover, the running times of the extractors we will construct later are closely related to the number of calls made to the malicious prover.

Definition 2.33. *Let* $\tilde{\mathbf{P}}_{\mathsf{IP}}$ *be any malicious prover for the interactive proof* $\mathsf{IP} = (\mathbf{P}_{\mathsf{IP}}, \mathbf{V}_{\mathsf{IP}})$ *with round complexity* k*. We say that an extractor* $\mathbf{E}_{\mathsf{IP}}$*, given black-box access to* $\tilde{\mathbf{P}}_{\mathsf{IP}}$*, does* I **invocations** *if* $\mathbf{E}_{\mathsf{IP}}$ *gets* I *transcripts. In other words, an invocation corresponds to asking every prover's messages, leading to a complete execution of* IP*. We reserve the term* **queries** *for calls to random oracles.*

The knowledge soundness property is insufficient to show that the actual Fiat–Shamir transformation of an IP is still knowledge sound. Indeed, a malicious prover for the non-interactive variant has the following advantage compared to a malicious prover for the interactive case. In the interactive case, the malicious prover cannot anticipate its attack against the verifier since it does not know what randomness it will use. However, the malicious prover for the Fiat–Shamir transformation can simulate, due to the definition of the transformation, any possible execution with the verifier before attacking. In other words, this malicious prover can reset its state to an older one, for example, just before sending its i-th message, and retry with another message. We call this principle *state restoration*. However, one can prove that knowledge soundness is sufficient for its NIROP to be knowledge sound in the specific case of SPs.

Below, we introduce another security notion related to IPs called *state-restoration knowledge soundness*. The latter implies the knowledge soundness property of the non-interactive transform, even for IPs.

2.7 State restoration

State restoration soundness is a stronger security notion than classical soundness. It requires security against a stronger class of malicious provers called *state restoration prover*. These malicious provers can execute the protocol multiple times before trying to convince the verifier. Similarly, *state restoration knowledge soundness* is a notion of knowledge soundness that ensures security against this stronger class of malicious provers. In these notes, we only focus on state restoration knowledge soundness. State restoration allows a malicious prover to have the same advantage as a malicious prover for the Fiat–Shamir transform but without the overhead of the transformation.

The state restoration variants of soundness and knowledge soundness are essential because they imply the security of the protocol's non-interactive Fiat–Shamir transformation. State restoration knowledge soundness is generally stronger than knowledge soundness; if an interactive proof is knowledge sound, then it is not necessarily state restoration knowledge sound. However, we can prove that special sound protocols are state restoration knowledge sound. Therefore, special soundness is sufficient to show the security of the non-interactive Fiat–Shamir transformation. Sections 5 and 6 are devoted to this claim. Fig. 3 represents implication relations within all the security definitions we have seen so far. The left part of this figure depicts security implications within the world of IPs while the right illustrates NIROPs, which are modeled, by definition, in the ROM.

State restoration knowledge soundness and state restoration soundness rely on a game called *state restoration game*. We start by giving some necessary definitions and then introduce this game.

Definition 2.34. *Given any random-oracle algorithm $\mathcal{A}$ that takes some input x, we are interested in the list of queries it has done to its random oracle* rnd. *This motivates the following definition and notation. Suppose that during an execution of $\mathcal{A}^{\mathsf{rnd}}(x)$, the algorithm $\mathcal{A}$ queried the points $(\alpha_1, \ldots, \alpha_t)$, got the responses $(\rho_1, \ldots, \rho_t)$ and eventually returns y. We say that*

$$\mathsf{tr} = ((\alpha_1, \rho_1), \ldots, (\alpha_t, \rho_t))$$

is the **trace of the execution** *$\mathcal{A}^{\mathsf{rnd}}(x)$. We denote the whole process by*

$$y \xleftarrow{\mathsf{tr}} \mathcal{A}^{\mathsf{rnd}}(x) \ .$$

Moreover, if $\mathcal{A}$ does at most t queries, we say that $\mathcal{A}$ is a t-query oracle algorithm.

Without loss of generality, we assume that all queries from the subsequent algorithms to the random oracles are distinct. Indeed, one can always efficiently store the queries and their responses. Moreover, we use the following notation for simplicity.

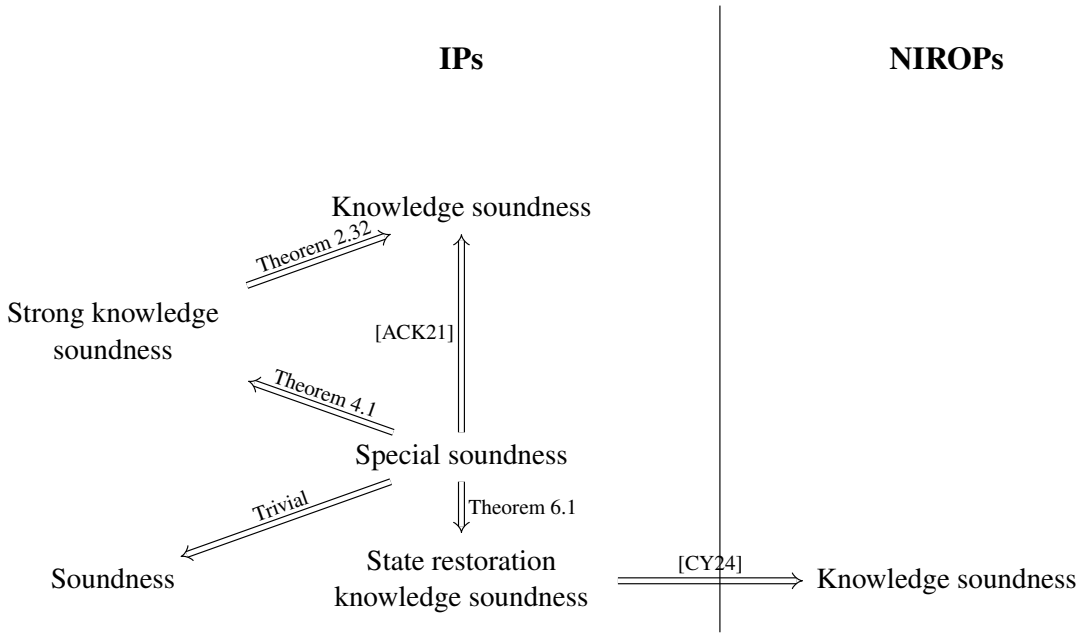

Figure 3: Overview of the currently known implications and those proved in this thesis.

Notation 10. *Inspired by Python's notation, for every $i \in [\mathsf{k}]$, we denote by $\mathcal{R}_{[:i]}$ the set $\mathcal{R}_1 \times \cdots \times \mathcal{R}_{i-1}$ and by $\mathcal{R}_{[i:]}$ the set $\mathcal{R}_i \times \cdots \times \mathcal{R}_{\mathsf{k}}$. We naturally extend this notation to tuples.*

Finally, the state restoration game is defined as follows.

Definition 2.35. *For every salt size $s \in \mathbb{N}$, ordered tuple of random oracles $\mathbf{rnd} \in \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big)$ and t-queries random-oracle algorithm $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}$, the* **state restoration game** Γ^{sr} *is defined as:*

$\Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}})$*:*

1. *Run the state restoration prover $(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} (\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}})^{\mathbf{rnd}}$.*
2. *Get challenges $\rho_i \coloneqq \mathsf{rnd}_i(\mathbb{x}, \boldsymbol{\alpha}_{[:i+1]}, \boldsymbol{\sigma}_{[:i+1]})$, for every $i \in [\mathsf{k}]$.*
3. *Return $(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, (\rho_1, \ldots, \rho_{\mathsf{k}}))$.*

The symbol $\mathbf{tr}^{\mathsf{sr}}$ is the tuple $(\mathsf{tr}^{\mathsf{sr}}_1, \ldots, \mathsf{tr}^{\mathsf{sr}}_{\mathsf{k}})$ of respective random oracle's trace. In addition, the game also returns $\mathbf{tr}^{\mathsf{sr}}$. We denote the whole process by

$$(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, (\rho_1, \ldots, \rho_{\mathsf{k}})) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \ .$$

If we consider SP, we slightly change the notation and write instead:

$\Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$*:*

1. *Run the state restoration prover $(\mathbb{x}, \alpha_1, \sigma, \alpha_2) \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} (\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})^{\mathsf{rnd}}$.*
2. *Get a challenge $\rho \coloneqq \mathsf{rnd}(\mathbb{x}, \alpha_1, \sigma)$.*
3. *Return $(\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2)$.*

The experiment uses the random oracles **rnd** to generate verifier challenges. This is similar to how the Fiat–Shamir transformation works and intuitively shows how state restoration security relates to the security of the non-interactive Fiat–Shamir transformation. Fig. 4 underscores how a malicious state restoration prover could use the state restoration game to simulate the Fiat–Shamir transformation. We use the same color as in Fig. 2 to make the similarity between the Fiat–Shamir transformation and state restoration explicit.

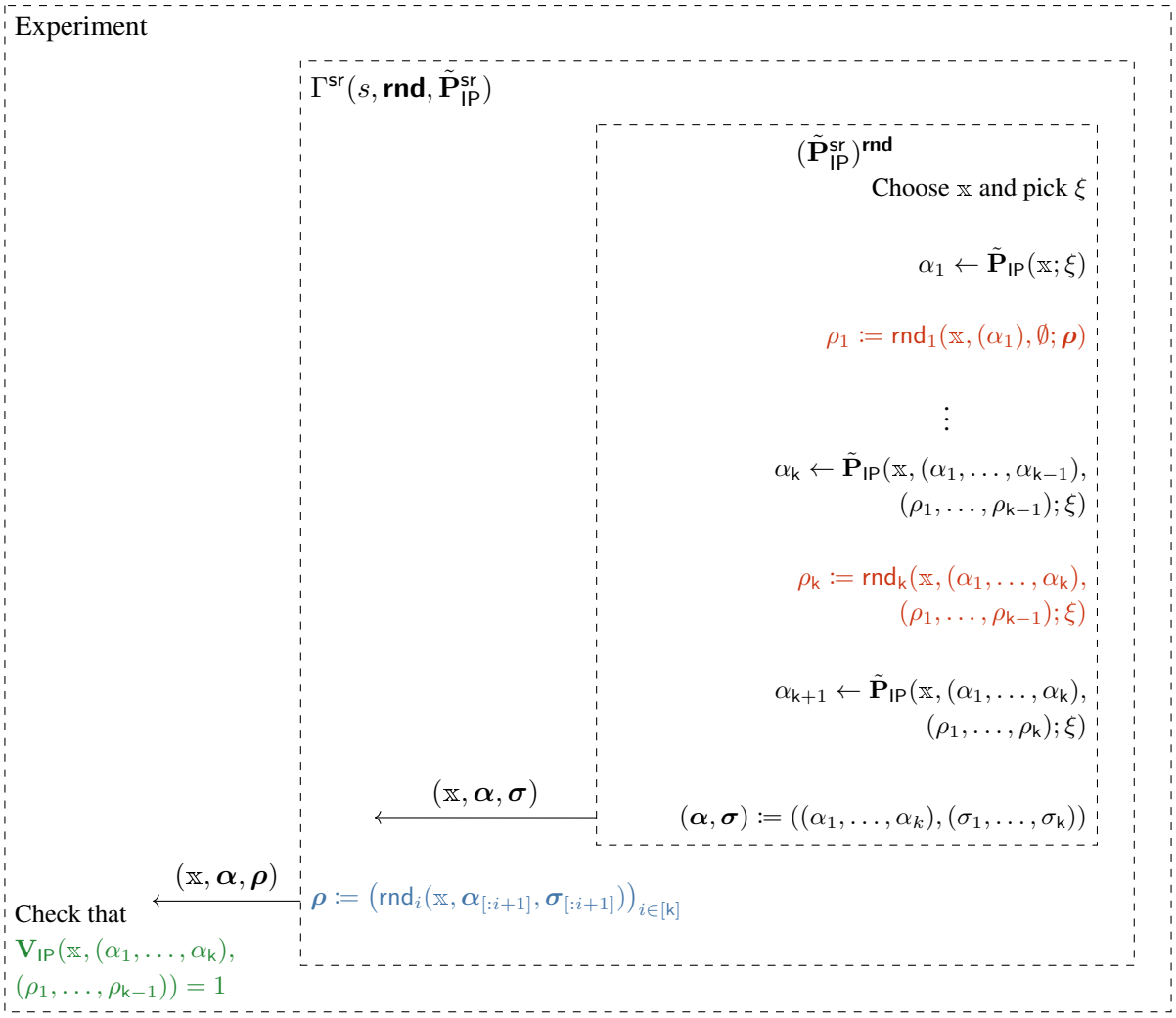

Figure 4: A malicious state restoration prover playing the state restoration game to simulate the Fiat–Shamir transformation. The uses of random oracles are highlighted in red, the check for the consistency of the verifier's messages is in blue, and the acceptance check is in green.

Remark 2.36. The malicious prover is allowed to query **rnd** multiple times. It reflects that the malicious prover could query random oracles, represented by hash functions in real life, beforehand and then proceed to the attack. In particular, it can run the protocol multiple times in its head. The response to these queries may give an advantage to the malicious prover. Therefore, security definitions in the state restoration setup must depend on the number of queries done to the random oracle.

The following remark will allow us to alleviate ourselves by considering only a family of adversaries that is easier to analyze.

Remark 2.37. We can restrict ourselves to deterministic malicious state restoration prover when playing Γ^{sr}. Indeed, if $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}$ is probabilistic, we could use the law of total probability to condition over its randomness.

We now introduce the state restoration variant of knowledge soundness.

Definition 2.38. *A $\mathsf{SP} = (\mathbf{P}_{\mathsf{SP}}, \mathbf{V}_{\mathsf{SP}})$ be a SP for a relation R. It has* **rewinding state restoration knowledge soundness error $\kappa^{\mathsf{sr}}_{\mathsf{SP}}$ with extraction time** $\mathsf{et}^{\mathsf{sr}}_{\mathsf{SP}}$ *if there exists a probabilistic algorithm $\mathbf{E}^{\mathsf{SRKS}}_{\mathsf{SP}}$ (the extractor) such that for every salt size $s \in \mathbb{N}^*$, query bound t, t-query deterministic malicious adaptive prover $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}$ and instance size bound $n \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{l|rcl} |\mathbb{x}| \leq n & \mathsf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \wedge\, (\mathbb{x}, \mathbb{w}) \notin R & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge\, b = 1 & b & \leftarrow & \mathbf{V}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\rho}) \\ & \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SRKS}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathsf{tr}^{\mathsf{sr}}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array}\right] \leq \kappa^{\mathsf{sr}}_{\mathsf{IP}}(\mathbb{x}, s, t) \ ,$$

where r_i is the verifier's randomness complexity for the i-th challenge. Additionally, $\mathbf{E}^{\mathsf{SRKS}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathsf{tr}^{\mathsf{sr}}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}})$ runs in expected time $\mathsf{et}^{\mathsf{sr}}_{\mathsf{IP}}(n, s, t, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}})$ over its internal randomnes. Moreover, as usual, we have a special notation for the case of SPs:

$$\Pr\left[\begin{array}{l|rcl} |\mathbb{x}| \leq n & \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ \wedge\, (\mathbb{x}, \mathbb{w}) \notin R & (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ \wedge\, b = 1 & b & \leftarrow & \mathbf{V}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho, \alpha_2) \\ & \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SRKS}}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2, \mathsf{tr}^{\mathsf{sr}}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \end{array}\right] \leq \kappa^{\mathsf{sr}}_{\mathsf{SP}}(\mathbb{x}, s, t) \ .$$

Note that the prover is fully adaptive, meaning it chooses the problem instance while running, compared to the non-adaptive version, where we quantify universally over all problem instances outside of the probability expression.

One of the benefits of the notion of state restoration is that it extends naturally to PCPs and IOPs. As with IPs, state restoration knowledge soundness also implies that their respective non-interactive transformations are knowledge sound under the ROM. We devote the following two sections to showing that a special sound SP and IP are state restoration knowledge soundness. In the cases of SPs and PCPs, soundness is enough for state restoration knowledge soundness. Nonetheless, we are still interested in proving that a special sound SP is state restoration knowledge sound since the proof will give us insight into the more general case of IPs.

3 Strong knowledge soundness from special soundness for SPs

In this section, we prove that special soundness implies strong knowledge soundness. But instead of proving it directly in the general setup of public-coin IPs, we start with the particular case of SPs. The main idea behind the proof is the same as in the public-coin IP case and, thus, will give us insights into the more generic proof.

Theorem 3.1. *Let* $\mathsf{SP} = (\mathbf{P}_{\mathsf{SP}}, \mathbf{V}_{\mathsf{SP}})$ *be a SP for a relation* R *with special soundness with arity* a *and verifier randomness complexity* r*. It follows that* SP *has strong rewinding knowledge soundness error*

$$\kappa_{\mathsf{SP}}(\mathbb{x}) \leq \frac{a-1}{2^{\mathsf{r}}}$$

and extraction time

$$\mathsf{et}_{\mathsf{SP}}(\mathbb{x}, \tilde{\mathbf{P}}_{\mathsf{SP}}) \leq (a-1) \cdot \left(\tau_{\tilde{\mathbf{P}}_{\mathsf{SP}}} + \mathsf{poly}(|\mathbb{x}|)\right) \ .$$

Moreover, the number of times the knowledge soundness extractor invokes $\tilde{\mathbf{P}}_{\mathsf{IP}}$ *is* $a-1$*.*

If we relax the above statement only to imply weak knowledge soundness instead of the strong version of the latter, we obtain a result demonstrated in [ACK21]. They also prove this statement's more general IPs case.

We prove this theorem in three steps. In the section below, we present the knowledge soundness extractor. Then, we demonstrate in Section 3.2 that it has the correct strong knowledge soundness error. Finally, we show in Section 3.3 that this extractor has the desired extraction time. Note that by Remark 2.30, we know that we can restrict ourselves to deterministic provers. For the rest of this section, we fix the protocol $\mathsf{SP} = (\mathbf{P}_{\mathsf{SP}}, \mathbf{V}_{\mathsf{SP}})$ for a relation R, the deterministic malicious prover $\tilde{\mathbf{P}}_{\mathsf{SP}}$ and the instance $\mathbb{x}$ arbitrarily. Suppose that SP has special soundness with arity a and that the verifier's randomness complexity is r. Moreover, let α_1 be the first message sent by $\tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x})$, which is fixed since we assume that $\tilde{\mathbf{P}}_{\mathsf{SP}}$ is deterministic.

3.1 Construction of the extractor

Let $\mathbf{E}^{\mathsf{SS}}_{\mathsf{SP}}$ be SP's special soundness extractor. Below, we introduce the notion of an accepting challenge.

Definition 3.2. *Given an instance* $\mathbb{x}$ *and a deterministic malicious prover* $\tilde{\mathbf{P}}_{\mathsf{SP}}$*, we say that a challenge* ρ **is accepting** *if* $\mathbf{V}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho, \alpha_2) = 1$*, where* $\alpha_2 \leftarrow \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho, \alpha_2)$*. Note that* α_2 *is uniquely determined by* $\tilde{\mathbf{P}}_{\mathsf{SP}}$*,* $\mathbb{x}$ *and* ρ *since* $\tilde{\mathbf{P}}_{\mathsf{SP}}$ *is deterministic.*

Construction 3.3. Consider the following knowledge extractor for SP:

$\mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\mathsf{SP}})$:
1. If $\mathbf{V}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho, \alpha_2) = 0$, return 0.
2. Add the given challenge to a set of picked challenges $S \coloneqq \{\rho\}$, store the accepting challenge and its answer $B \coloneqq \{(\rho, \alpha_2)\}$.
3. While it remains unpicked challenges $|\mathcal{R} \setminus S| > 0$ and we have not found enough accepting challenges $|B| \neq a$:
   (a) Get a new challenge unpicked $\rho' \leftarrow \mathcal{R} \setminus S$.
   (b) Add the new challenge to a set of picked challenges $S \coloneqq S \cup \{\rho'\}$.
   (c) Get the response for the challenge $\alpha'_2 \leftarrow \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}, \rho')$.
   (d) If $\mathbf{V}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho', \alpha'_2) = 1$, store the accepting challenge and its response $B \coloneqq B \cup \{(\rho', \alpha'_2)\}$.

4. If $|B| = a$, create an a-fork $\mathbf{F} \coloneqq (\alpha_1, B)$ and return $\mathbf{E}^{\mathsf{SS}}_{\mathsf{SP}}(\mathbb{x}, \mathbf{F})$ otherwise return 0.

The algorithm $\mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}$ first checks at Line 1 that the given transcript is accepting for $\mathbb{x}$. We refer to this test as the *early abort strategy*. Then, it randomly picks the verifier's randomness until it has found enough accepting to construct a a-tree of accepting transcripts for $\mathbb{x}$. If it succeeds, it calls the special soundness extractor and retrieves a witness for $\mathbb{x}$.

3.2 Knowledge soundness error

The following definition helps us understand our extractor's probability of success.

Definition 3.4. *Let*

$$\mathsf{Acc} \coloneqq \{\rho \in \mathcal{R} \mid 1 \leftarrow \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}), \mathbf{V}_{\mathsf{SP}}(\mathbb{x}; \rho)\rangle_{\mathsf{SP}}\}$$

be the **set of accepting randomness for** $\mathbb{x}$*, or in other words, the set of verifier's messages that are accepting for* $\mathbb{x}$*.*

We assert the following statement.

Lemma 3.5. *The following holds for the extractor* $\mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}$ *of Construction 3.3:*

$$\Pr\left[\begin{array}{l|ccc} (\mathbb{x}, \mathbb{w}) \notin R & \rho & \leftarrow & \mathcal{R} \\ \wedge\, b = 1 & b & \xleftarrow{((\alpha_1,\alpha_2),\rho)} & \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}), \mathbf{V}_{\mathsf{SP}}(\mathbb{x}; \rho)\rangle_{\mathsf{SP}} \\ & \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\mathsf{SP}}) \end{array}\right] \leq \frac{a-1}{2^{\mathsf{r}}} \ .$$

Proof. Since SP has special soundness with arity a there is a special sound extractor $\mathbf{E}^{\mathsf{SS}}_{\mathsf{SP}}$ that, given an a-tree of accepting transcripts for $\mathbb{x}$ returns a witness for $\mathbb{x}$. Recall that Construction 3.3 is instantiated with this special soundness extractor. We split the proof into two distinct cases: whether or not there are enough accepting challenges. More formally, we are interested in knowing if the quantity $|\mathsf{Acc}|$ is strictly smaller or not than the arity a.

There are enough accepting challenges. In that case, it is impossible for the success bit b to be one and for the extractor to *not* retrieve a witness. Indeed, if $b = 1$, the extractor will not early abort. Since there are enough accepting challenges, the extractor can retrieve an a-tree of accepting transcripts for $\mathbb{x}$, making it return a correct witness for $\mathbb{x}$. In other words, the probability equals zero in that case.

There are not enough accepting challenges. There is no a-tree of accepting transcripts for $\mathbb{x}$. Therefore the extractor cannot return a witness for $\mathbb{x}$. This implies that the probability is the same as the probability that $\langle \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}), \mathbf{V}_{\mathsf{SP}}(\mathbb{x}; \rho)\rangle_{\mathsf{SP}} = 1$ which is $|\mathsf{Acc}|/2^{\mathsf{r}}$ by definition of $|\mathsf{Acc}|$. Since we are in the case $|\mathsf{Acc}| \leq a-1$, the probability is at most $(a-1)/2^{\mathsf{r}}$.

Therefore, the probability is upper-bounded by $(a-1)/2^{\mathsf{r}}$ in both cases. □

3.3 Extraction time

Definition 3.6. *Let* $I(\rho, \nu)$ *be a random variable that counts the number of invocations*

$$\mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\mathsf{SP}}; \nu)$$

does to $\tilde{\mathbf{P}}_{\mathsf{SP}}$*, where the extractor* $\mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}$ *is implemented as Construction 3.3 and* α_1 *and* α_2 *are* $\tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x})$*'s messages when it interacts with* $\mathbf{V}_{\mathsf{SP}}(\mathbb{x}; \rho)$*.*

We prove the following upper bound on the expected number of invocations to $\tilde{\mathbf{P}}_{\mathsf{SP}}$ performed by $\mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}$.

Lemma 3.7. *We have*

$$\mathbb{E}\left[\, I(\rho,\nu) \;\middle|\; \begin{array}{lcl} \nu & \leftarrow & \mathcal{N} \\ \rho & \leftarrow & \mathcal{R} \\ b & \xleftarrow{((\alpha_1,\alpha_2),\rho)} & \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}), \mathbf{V}_{\mathsf{SP}}(\mathbb{x};\rho)\rangle_{\mathsf{SP}} \\ \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}(\mathbb{x},\alpha_1,\rho,\alpha_2,\tilde{\mathbf{P}}_{\mathsf{SP}};\nu) \end{array} \right] \leq a-1 \ .$$

Proof. Let $\ell := |\mathsf{Acc}|$. We can now compute

$$\begin{aligned}
&\mathbb{E}\left[\, I(\rho,\nu) \;\middle|\; \begin{array}{lcl} \nu & \leftarrow & \mathcal{N} \\ \rho & \leftarrow & \mathcal{R} \\ b & \xleftarrow{((\alpha_1,\alpha_2),\rho)} & \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}), \mathbf{V}_{\mathsf{SP}}(\mathbb{x};\rho)\rangle_{\mathsf{SP}} \\ \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}(\mathbb{x},\alpha_1,\rho,\alpha_2,\tilde{\mathbf{P}}_{\mathsf{SP}};\nu) \end{array} \right] \\
&= \mathbb{E}\left[\begin{array}{l} I(\rho,\nu) \\ \text{conditioned on} \\ b=1 \end{array} \;\middle|\; \begin{array}{lcl} \nu & \leftarrow & \mathcal{N} \\ \rho & \leftarrow & \mathcal{R} \\ b & \xleftarrow{((\alpha_1,\alpha_2),\rho)} & \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}), \mathbf{V}_{\mathsf{SP}}(\mathbb{x};\rho)\rangle_{\mathsf{SP}} \\ \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}(\mathbb{x},\alpha_1,\rho,\alpha_2,\tilde{\mathbf{P}}_{\mathsf{SP}};\nu) \end{array} \right] \\
&\quad \cdot \Pr\left[b=1 \;\middle|\; b \leftarrow \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}), \mathbf{V}_{\mathsf{SP}}(\mathbb{x})\rangle_{\mathsf{SP}} \right] \\
&\quad + \underbrace{\mathbb{E}\left[\begin{array}{l} I(\rho,\nu) \\ \text{conditioned on} \\ b=0 \end{array} \;\middle|\; \begin{array}{lcl} \nu & \leftarrow & \mathcal{N} \\ \rho & \leftarrow & \mathcal{R} \\ b & \xleftarrow{((\alpha_1,\alpha_2),\rho)} & \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}), \mathbf{V}_{\mathsf{SP}}(\mathbb{x};\rho)\rangle_{\mathsf{SP}} \\ \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}(\mathbb{x},\alpha_1,\rho,\alpha_2,\tilde{\mathbf{P}}_{\mathsf{SP}};\nu) \end{array} \right]}_{\text{0 because the algorithm will directly abort.}} \\
&\quad \cdot \Pr\left[b=0 \;\middle|\; b \leftarrow \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}), \mathbf{V}_{\mathsf{SP}}(\mathbb{x})\rangle_{\mathsf{SP}} \right] \\
&= \frac{\ell}{N}\,\mathbb{E}\left[\begin{array}{l} I(\rho,\nu) \\ \text{conditioned on} \\ b=1 \end{array} \;\middle|\; \begin{array}{lcl} \nu & \leftarrow & \mathcal{N} \\ \rho & \leftarrow & \mathcal{R} \\ b & \xleftarrow{((\alpha_1,\alpha_2),\rho)} & \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}), \mathbf{V}_{\mathsf{SP}}(\mathbb{x};\rho)\rangle_{\mathsf{SP}} \\ \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}(\mathbb{x},\alpha_1,\rho,\alpha_2,\tilde{\mathbf{P}}_{\mathsf{SP}};\nu) \end{array} \right] ,
\end{aligned}$$

where $N := 2^{\mathsf{r}}$ and

$$\Pr\left[b=1 \;\middle|\; b \leftarrow \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}), \mathbf{V}_{\mathsf{SP}}(\mathbb{x})\rangle_{\mathsf{SP}} \right] = \frac{\ell}{N}$$

by definition of Acc and since $|\mathsf{Acc}| = \ell$.

Let

$$e := \mathbb{E}\left[\begin{array}{l} I(\rho,\nu) \\ \text{conditioned on} \\ b=1 \end{array} \;\middle|\; \begin{array}{lcl} \nu & \leftarrow & \mathcal{N} \\ \rho & \leftarrow & \mathcal{R} \\ b & \xleftarrow{((\alpha_1,\alpha_2),\rho)} & \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}), \mathbf{V}_{\mathsf{SP}}(\mathbb{x};\rho)\rangle_{\mathsf{SP}} \\ \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}(\mathbb{x},\alpha_1,\rho,\alpha_2,\tilde{\mathbf{P}}_{\mathsf{SP}};\nu) \end{array} \right] .$$

We can show that $e \leq (a-1)\frac{N}{\ell}$. Indeed, if $\ell < a$ the number of invocations will be $N-1$ which is smaller than $(a-1)\frac{N}{\ell}$. Otherwise, if $\ell \geq a$, the algorithm will enter the negative hypergeometric distribution (NHG) distribution. Therefore, in the latter case, $e = \mu_{N-1,\ell-1,a-1} = (a-1)\frac{N}{\ell}$ where $\mu_{N-1,\ell-1,a-1}$ is the mean

of the NHG, defined in Definition 2.14. We can substitute these upper bounds in the expected number of invocations:

$$\begin{aligned}
&\mathbb{E}\left[I(\rho,\nu) \;\middle|\; \begin{array}{ccc} \nu & \leftarrow & \mathcal{N} \\ \rho & \leftarrow & \mathcal{R} \\ b & \xleftarrow{((\alpha_1,\alpha_2),\rho)} & \langle \tilde{\mathbf{P}}_{\mathsf{SP}}(\mathbb{x}), \mathbf{V}_{\mathsf{SP}}(\mathbb{x};\rho)\rangle_{\mathsf{SP}} \\ \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SKS}}_{\mathsf{SP}}(\mathbb{x},\alpha_1,\rho,\alpha_2,\tilde{\mathbf{P}}_{\mathsf{SP}};\nu) \end{array} \right] \\
&\leq \frac{\ell}{N}(a-1)\frac{N}{\ell} \\
&\leq a-1 \ ,
\end{aligned}$$

which is the desired result. □

Every time the extractor invokes $\tilde{\mathbf{P}}_{\mathsf{SP}}$, it also runs some polynomial time computation. Hence, the overall extraction time is

$$(a-1)\cdot\left(\tau_{\tilde{\mathbf{P}}_{\mathsf{SP}}} + \mathsf{poly}(|\mathbb{x}|)\right) \ .$$

4 Strong knowledge soundness from special soundness for IPs

We focus now on extending Theorem 3.1 to all public-coin interactive proofs, regardless of the number of rounds.

Theorem 4.1. *Let* $\mathsf{IP} = (\mathbf{P}_{\mathsf{IP}}, \mathbf{V}_{\mathsf{IP}})$ *be a public-coin interactive proof for a relation* R *with round complexity* k*,* $(a_1, \ldots, a_{\mathsf{k}})$*-special soundness and randomness complexity* $(\mathsf{r}_1, \ldots, \mathsf{r}_{\mathsf{k}})$*. Then* IP *has strong rewinding knowledge soundness error*

$$\kappa_{\mathsf{IP}}(\mathbb{x}) \leq 1 - \prod_{i=1}^{\mathsf{k}} \left(1 - \frac{a_i - 1}{2^{\mathsf{r}_i}}\right)$$

and extraction time

$$\mathsf{et}_{\mathsf{IP}}(\mathbb{x}, \tilde{\mathbf{P}}_{\mathsf{IP}}) \leq \left(\left(\prod_{i=1}^{\mathsf{k}} a_i\right) - 1\right) \cdot \left(\tau_{\tilde{\mathbf{P}}_{\mathsf{IP}}} + \mathsf{poly}(|\mathbb{x}|)\right) \ .$$

Moreover, the number of times the knowledge soundness extractor invokes $\tilde{\mathbf{P}}_{\mathsf{IP}}$ *is* $\left(\prod_{i=1}^{\mathsf{k}} a_i\right) - 1$*.*

As in Section 3, we leverage that our malicious provers can be deterministic without losing generality. Fix arbitrarily a protocol $\mathsf{IP} = (\mathbf{P}_{\mathsf{IP}}, \mathbf{V}_{\mathsf{IP}})$ for a relation R, a deterministic malicious prover $\tilde{\mathbf{P}}_{\mathsf{IP}}$ and an instance $\mathbb{x}$. Suppose that IP has $(a_1, \ldots, a_{\mathsf{k}})$-special soundness and randomness complexity $(\mathsf{r}_1, \ldots, \mathsf{r}_{\mathsf{k}})$. For every $i \in [\mathsf{k}]$, we define

$$N_i := 2^{\mathsf{r}_i} \ .$$

Moreover, let α_1 be the first message sent by $\tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x})$.

4.1 Construction of the extractor

Let $\mathbf{E}^{\mathsf{SS}}_{\mathsf{IP}}$ be IP's special soundness extractor, which is given by assumption. We construct a knowledge soundness extractor $\mathbf{E}^{\mathsf{SKS}}_{\mathsf{IP}}$ that first searches for a $(a_1, \ldots, a_k)$-tree of accepting transcripts for $\mathbb{x}$ using a helper algorithm called $\mathsf{TreeFinder}_0$, and finally gets a witness by calling $\mathbf{E}^{\mathsf{SS}}_{\mathsf{IP}}$. The extractor $\mathbf{E}^{\mathsf{SKS}}_{\mathsf{IP}}$ is given a transcript

$$(\underbrace{(\alpha_1, \ldots, \alpha_{\mathsf{k}+1})}_{=\boldsymbol{\alpha}}, \underbrace{(\rho_1, \ldots, \rho_{\mathsf{k}})}_{=\boldsymbol{\rho}})$$

and an instance $\mathbb{x}$ by the definition of strong knowledge soundness. We state the pseudocode of $\mathbf{E}^{\mathsf{SKS}}_{\mathsf{IP}}$ below.

Construction 4.2. Using the helper algorithm $\mathsf{TreeFinder}_0$, the extractor is

$\mathbf{E}^{\mathsf{SKS}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \tilde{\mathbf{P}}_{\mathsf{IP}})$:
1. $T \leftarrow \mathsf{TreeFinder}_0(\mathbb{x}, (\alpha_1), \emptyset, \tilde{\mathbf{P}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \top)$.
2. If $T = \perp$ return $\perp$ otherwise return $\mathbf{E}^{\mathsf{SS}}_{\mathsf{IP}}(\mathbb{x}, T)$.

Note that we can fix the randomness of this extractor by fixing the randomness of $\mathsf{TreeFinder}_0$.

The following lemma follows directly from the properties of $\mathbf{E}^{\mathsf{SS}}_{\mathsf{IP}}$.

Lemma 4.3. *This extractor succeeds if and only if*

$$\mathsf{TreeFinder}_0(\mathbb{x}, (\alpha_1), \emptyset, \tilde{\mathbf{P}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \top)$$

finds a $(a_1, \ldots, a_{\mathsf{k}})$*-tree of accepting transcripts for* $\mathbb{x}$*. In other words, their success probabilities are the same.*

We describe below the pseudocode of the $\mathsf{TreeFinder}_i$'s procedure. The notion of *good challenge* is defined later.

Construction 4.4. $\mathsf{TreeFinder}_i(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}', \tilde{\mathbf{P}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \mathfrak{f})$:

1. If we are treating the base case ($i = \mathsf{k}$):
   (a) Get the success bit $b \leftarrow \mathbf{V}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}')$.
   (b) If $b = 1$ return 1 otherwise return $\perp$.
2. $\rho_{i+1}^{\mathsf{ea}} \leftarrow \rho_{i+1}$ if we are on the left most branch ($\mathfrak{f} = \top$), $\rho_{i+1}^{\mathsf{ea}} \leftarrow \mathcal{R}_{i+1}$ otherwise.
3. Get the response: $\alpha_{i+2}^{\mathsf{ea}} \coloneqq \alpha_{i+2}$ if $\mathfrak{f} = \top$, $\alpha_{i+2}^{\mathsf{ea}} \leftarrow \tilde{\mathbf{P}}_{\mathsf{IP}}(\boldsymbol{\alpha}', \boldsymbol{\rho}' \parallel \rho_{i+1}^{\mathsf{ea}})$ otherwise.
4. First early abort recursive call $T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \boldsymbol{\alpha}' \parallel \alpha_{i+2}^{\mathsf{ea}}, \boldsymbol{\rho}' \parallel \rho_{i+1}^{\mathsf{ea}}, \tilde{\mathbf{P}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \mathfrak{f})$.
5. If $T = \perp$ return $\perp$.
6. Add the challenge to a set of picked challenges $S \coloneqq \{\rho_{i+1}^{\mathsf{ea}}\}$, store the good challenge in a list $E \coloneqq (\rho_{i+1}^{\mathsf{ea}})$ and its answer $N \coloneqq (\alpha_{i+2}^{\mathsf{ea}})$.
7. While it remains unpicked challenges $|\mathcal{R} \setminus S| > 0$ and we have not found enough good challenges $|E| \neq a$:
   (a) Get a new challenge unpicked $\rho'_{i+1} \leftarrow \mathcal{R}_{i+1} \setminus S$.
   (b) Add the new challenge to a set of picked challenges $S \coloneqq S \cup \{\rho'_{i+1}\}$.
   (c) Get the response for the challenge $\alpha'_{i+2} \leftarrow \tilde{\mathbf{P}}_{\mathsf{IP}}(\boldsymbol{\alpha}', \boldsymbol{\rho}' \parallel \rho'_{i+1})$.
   (d) Recursive call $T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \boldsymbol{\alpha}' \parallel \alpha'_{i+2}, \boldsymbol{\rho}' \parallel \rho'_{i+1}, \tilde{\mathbf{P}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \perp)$.
   (e) If $T \neq \perp$, add the good challenge to the list $E \coloneqq E \parallel \rho'_{i+1}$ and its response $N \coloneqq N \parallel \alpha'_{i+2}$.
8. If $|E| = a_{i+1}$, create a tree with α'_{i+1} as root and children N (verifier responses) connected by edges E (prover challenges) and return it, otherwise return $\perp$.

To begin with, we ignore the transcript $(\boldsymbol{\alpha}, \boldsymbol{\rho})$ and the last parameter of $\mathsf{TreeFinder}_i$ for convenience. The helper algorithm $\mathsf{TreeFinder}_i$ generally works as follows. It is given an instance $\mathbb{x}$, a partial transcript

$$(\underbrace{(\alpha'_1, \ldots, \alpha'_{i+1})}_{=\boldsymbol{\alpha}'}, \underbrace{(\rho'_1, \ldots, \rho'_i)}_{=\boldsymbol{\rho}'})$$

and access to the malicious prover $\tilde{\mathbf{P}}_{\mathsf{IP}}$ and is tasked to find a $(a_{i+1}, \ldots, a_{\mathsf{k}})$-tree of accepting transcripts for $\mathbb{x}$ with prefix $\boldsymbol{\rho}'$. Recall that the notion of a prefix is presented in Definition 2.25. Note that when $i = 0$, this correspond to finding a $(a_1, \ldots, a_{\mathsf{k}})$-tree of accepting transcripts for $\mathbb{x}$, as desired.

To do so, the algorithm tries to find a_{i+1} distinct challenges $\rho_{i+1}^1, \ldots, \rho_{i+1}^{a_{i+1}}$ such that it can retrieve, for every $j \in [a_{i+1}]$, a tree of accepting transcripts for $\mathbb{x}$ with prefix $(\rho_1, \ldots, \rho_i, \rho_{i+1}^j)$. By combining these a_{i+1} trees, it can construct a $(a_{i+1}, \ldots, a_{\mathsf{k}})$-tree of accepting transcripts for $\mathbb{x}$ with prefix $\boldsymbol{\rho}'$. If the algorithm cannot find enough of these challenges within $challengeSpacei + 1 \coloneqq \{0,1\}^{\mathsf{r}_{i+1}}$, it aborts.

To test whether a challenge ρ'_{i+1} could be used to create a tree of accepting transcripts for $\mathbb{x}$ with prefix $(\rho_1, \ldots, \rho_i, \rho'_{i+1})$, the algorithm first computes the prover's response $\alpha_{i+2} \leftarrow \tilde{\mathbf{P}}_{\mathsf{IP}}(\boldsymbol{\alpha}', \boldsymbol{\rho}' \parallel \rho'_{i+1})$ and then calls

$$\mathsf{TreeFinder}_{i+1}(\mathbb{x}, \boldsymbol{\alpha}' \parallel \alpha_{i+2}, \boldsymbol{\rho}' \parallel \rho'_{i+1}, \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \cdot) \ ,$$

which is the recursive part of the algorithm. We ignore the three last parameters for now since they are related to the given transcript $(\boldsymbol{\alpha}, \boldsymbol{\rho})$. The algorithm $\mathsf{TreeFinder}_i$ tasks the recursive call to retrieve a tree of accepting transcripts for $\mathbb{x}$ with prefix $(\rho_1, \ldots, \rho_i, \rho'_{i+1})$. If this is the case, we say that ρ'_{i+1} is *good*. More formally, we introduce the following definition.

Definition 4.5. *For every $i \in [\mathsf{k}]$, prefix $\boldsymbol{\rho} \in \mathcal{R}_{[:i]}$ and extractor randomness ν' we say that a challenge $\rho_i \in \mathcal{R}_i$* **is good for** *$(\nu', \boldsymbol{\rho})$ if*

$$\mathsf{TreeFinder}_i(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\rho} \parallel \rho_i, \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot; \nu') \neq \bot \ ,$$

where $\parallel$ denotes the concatenation operator and $\boldsymbol{\alpha} \leftarrow \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\rho} \parallel \rho_i)$. Moreover, let

$$Y_i(\nu', \boldsymbol{\rho}')$$

counts the number of good challenges for $(\nu', \boldsymbol{\rho}')$ within $\mathcal{R}_i$.

The construction of the $(a_{i+1}, \ldots, a_{\mathsf{k}})$-tree of accepting transcripts for $\mathbb{x}$ with prefix $\boldsymbol{\rho}'$ from the trees resulting from the recursive calls is depicted in Fig. 5. Nodes and triangles in green represent trees of accepting transcripts for $\mathbb{x}$ with prefix $\boldsymbol{\rho}'$ where $\boldsymbol{\rho}'$ is the path from the root node to the working node. Red edges and nodes illustrate that the recursive call could not return a tree of accepting transcript for $\mathbb{x}$ with the corresponding prefix. By trying out multiple challenges in $\{0,1\}^{\mathsf{r}_{i+1}}$, the node corresponding to the prover message α_{i+1} has found a tuple $(\rho_1, \ldots, \rho_{a_{i+1}})$ of good challenges that it can use to construct the $(a_{i+1}, \ldots, a_{\mathsf{k}})$-tree of accepting transcripts for $\mathbb{x}$ with prefix $\boldsymbol{\rho}'$ showed in green.

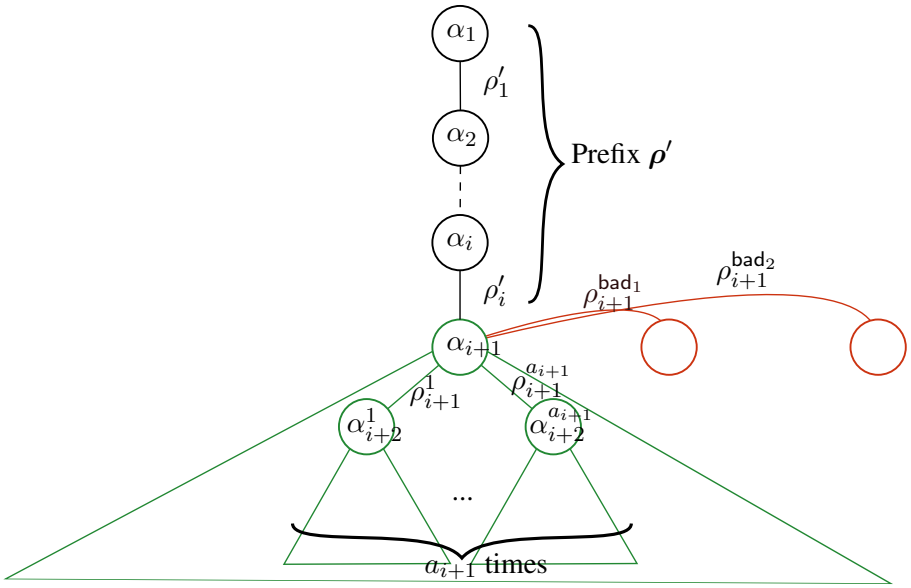

Figure 5: Example of the construction of a $(a_{i+1}, \ldots, a_{\mathsf{k}})$-tree of accepting transcripts for $\mathbb{x}$ with prefix $\boldsymbol{\rho}'$ from the trees obtained from the recursive call to $\mathsf{TreeFinder}_{i+1}$.

For the analysis to be correct, we must adapt the *early abort strategy* used in Construction 3.3 to the multi-round case. Hopefully, it generalizes naturally: for any instantiation $\mathsf{TreeFinder}_i$, if the first recursive call to $\mathsf{TreeFinder}_{i+1}$ performed by $\mathsf{TreeFinder}_i$ aborts, then $\mathsf{TreeFinder}_i$ also aborts. The result of the first recursive call is tested at Line 5.

It remains to explain how the algorithm depends on the given transcript $(\boldsymbol{\alpha}, \boldsymbol{\rho})$. The idea is to consider this transcript as a starting point to construct the $(a_1, \ldots, a_{\mathsf{k}})$-tree of transcripts for $\mathbb{x}$. We visualize this transcript as a complete branch *i.e.* a path from the root note to a leaf node in the tree we want to build. More precisely, we arbitrarily choose this branch to be the *left-most branch* of the tree. To implement this in our algorithm, it has to distinguish between two cases:

1. $\mathsf{TreeFinder}_i$ is on the left-most branch, *i.e.* the prefix is equal to $\boldsymbol{\rho}_{[:i+1]}$; and

2. $\mathsf{TreeFinder}_i$ is not on the left-most branch.

In the former case, $\mathsf{TreeFinder}_i$ chooses ρ_{i+1} as its first challenge and then picks all other challenges uniformly at random (without replacement). This way, we construct the left-most branch to represent the given transcript $(\boldsymbol{\alpha}, \boldsymbol{\rho})$. Note that this allows the $\mathsf{TreeFinder}_0$ to abort in k recursive steps if the given transcript is not an accepting transcript for $\mathbb{x}$. In the latter case, it picks all challenges uniformly at random (without replacement), ignoring the ones in the given transcript $(\boldsymbol{\alpha}, \boldsymbol{\rho})$.

To track whether we are on the left-most branch, we use an additional parameter $\mathfrak{f}$, a flag indicating whether we are on the left-most branch. Note that when calling $\mathsf{TreeFinder}_0$ from $\mathbf{E}^{\mathsf{SKS}}_{\mathsf{IP}}$, we pass $\mathfrak{f} = \top$ to $\mathsf{TreeFinder}_0$ to indicate that we start on the left-most branch. Indeed, the root note corresponds to the prover's message α_1.

We illustrate an example of a tree traveled by $\mathsf{TreeFinder}_0$ in Fig. 6. By looking at this tree, we recognize that the instantiation

$$\mathsf{TreeFinder}_1(\mathbb{x}, (\alpha_1, \alpha_2^2), (\rho_1^2), \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot)$$

is not able to find enough good challenges for $(\nu', (\rho_1^2))$, with ν' being the algorithm's randomness. Therefore, it cannot return an a_2-tree of accepting transcript for $\mathbb{x}$ with prefix (ρ_1^2). An example of an early abort is depicted by the instantiation of

$$\mathsf{TreeFinder}_{\mathsf{k}-1}(\mathbb{x}, (\alpha_1, \alpha_2^2, \alpha_3^{2,2}, \dots, \alpha_{\mathsf{k}}^{2,2,1,\dots,1}), (\rho_1^2, \rho_2^{2,2}, \dots), \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot) \ .$$

The early abort strategy allows it to directly abort since the challenge $\rho_2^{2,2,1,\dots,1}$ does not yield an accepting transcript.

Consider two distinct instantiations of $\mathsf{TreeFinder}_i$ with $\mathfrak{f} = \top$ and $\mathfrak{f} = \bot$ as two different algorithms. Their randomness differs since when $\mathfrak{f} = \bot$; the algorithm has to pick more challenges: the ones involved in the left-most branch, whereas these challenges are already given when $\mathfrak{f} = \top$.

Definition 4.6. *Let $\nu' \in \mathcal{N}'$ and $\nu \in \mathcal{N}$ be the respective randomness of*

$$\mathsf{TreeFinder}_i(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}', \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot)$$

and

$$\mathsf{TreeFinder}_i(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}', \tilde{\mathbf{P}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \top) \ .$$

4.2 Knowledge soundness error

Conditioned on the number of good challenges being ℓ, the probability that the extractor picks a good challenge is straightforward to compute.

Lemma 4.7. *For every $i \in [k]$, $\ell \in \{0, \dots, N_i\}$ and $\boldsymbol{\rho}' \in \mathcal{R}_{[:i]}$, we have*

$$\Pr\left[\begin{array}{l|lcl} T \neq \bot & \nu' & \leftarrow & \mathcal{N}' \\ \textit{conditioned on} & \alpha_i' & \coloneqq & \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}' \parallel \rho_i^{\mathsf{ea}}(\nu')) \\ Y_i(\nu', \boldsymbol{\rho}') = \ell & T & \leftarrow & \mathsf{TreeFinder}_i(\mathbb{x}, \boldsymbol{\alpha}' \parallel \alpha_i, \boldsymbol{\rho}' \parallel \rho_i^{\mathsf{ea}}(\nu'), \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot) \end{array}\right] = \frac{\ell}{N_i} \ ,$$

where $\rho_i^{\mathsf{ea}}(\nu')$ is the first challenge picked by

$$\mathsf{TreeFinder}_{i-1}(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}', \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot; \nu') \ ,$$

that is, the challenge involved in the early abort test.

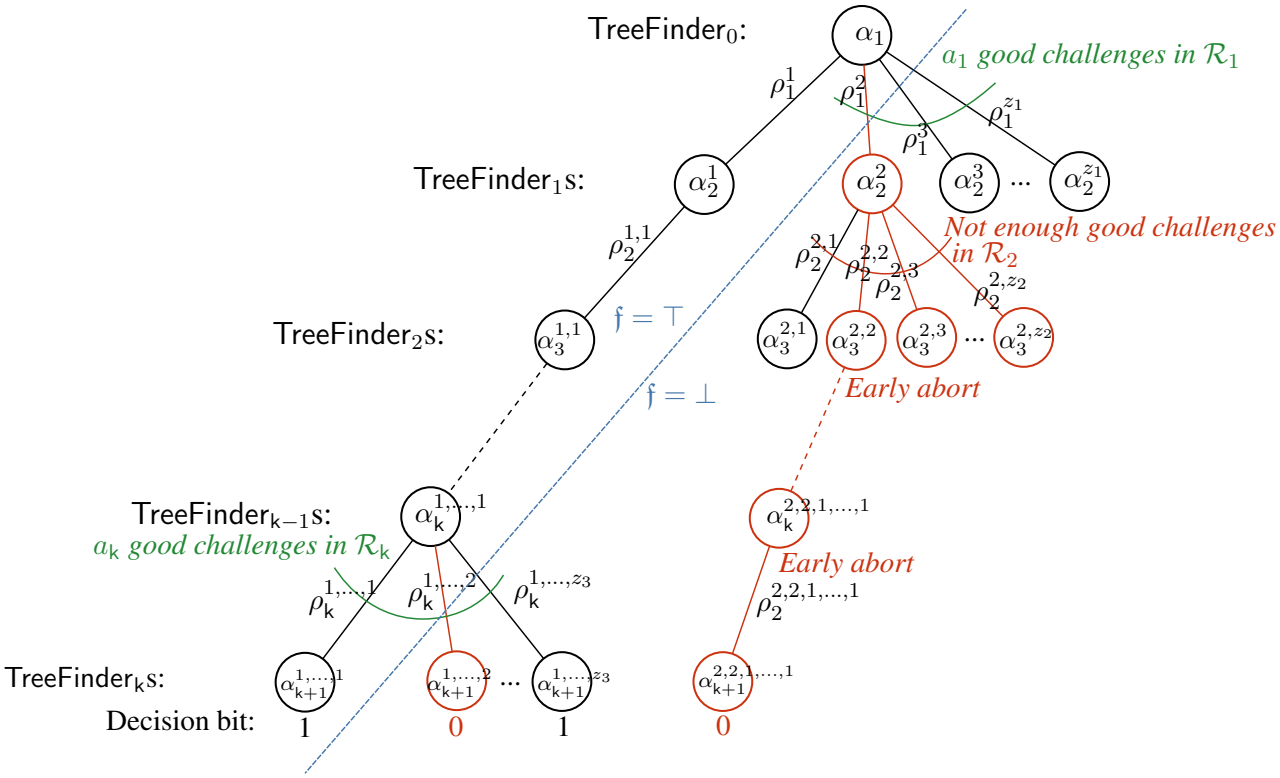

Figure 6: Big picture of the extractor given in Construction 4.4. Edges and nodes in red signify that the algorithm cannot return a tree of accepting transcripts. The blue line separates between calls with the flag $\mathfrak{f}$ equals true and false. The given transcript is $((\alpha_1^1, \alpha_2^{1,1}, \ldots, \alpha_{k+1}^{1,\ldots,1}), (\rho_1^1, \rho_2^{1,1}, \ldots, \rho_k^{1,\ldots,1}))$. Given the current transcript, the decision bit is the verifier's decision, and z_1, z_2 and z_3 are integers representing how much challenge has been tested by the current TreeFinder$_i$ instantiation.

Proof. TreeFinder$_{i-1}(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}', \tilde{\mathbf{P}}_{\mathsf{IP}}, \bot, \cdot, \cdot; \nu')$ chooses its first challenge $\rho_i^{\mathsf{ea}}(\nu')$ uniformly at random, independently of everything. Conditioned on $Y_i(\nu', \boldsymbol{\rho}') = \ell$, there are ℓ good challenges for $(\nu', \boldsymbol{\rho}')$ within the N_i challenges and thus, by definition of a good challenge we get the desired result. □

The following lemma will be useful later.

Lemma 4.8 ([ACK21])**.** *For every $i \in [k]$, prefix $\boldsymbol{\rho} \in \mathcal{R}_{[:i]}$ and extractor randomness ν' we have*

$$\sum_{\ell=0}^{a_i-1} \Pr\left[Y_i(\nu', \boldsymbol{\rho}') = \ell \;\middle|\; \nu' \leftarrow \mathcal{N}'\right] \leq \frac{N_i - \mathbb{E}[Y_i(\nu', \boldsymbol{\rho}') \mid \nu' \leftarrow \mathcal{N}']}{N_i - a_i + 1} \ .$$

Proof.

$$\begin{aligned}\mathbb{E}\left[Y_i(\nu', \boldsymbol{\rho}') = \ell \;\middle|\; \nu' \leftarrow \mathcal{N}'\right] &= \sum_{\ell=0}^{N_i} \ell \Pr\left[Y_i(\nu', \boldsymbol{\rho}') = \ell \;\middle|\; \nu' \leftarrow \mathcal{N}'\right] \\ &= \sum_{\ell=0}^{(a_i-1)} \ell \Pr\left[Y_i(\nu', \boldsymbol{\rho}') = \ell \;\middle|\; \nu' \leftarrow \mathcal{N}'\right]\end{aligned}$$

$$
\begin{aligned}
&+ \sum_{\ell=a_i}^{N_i} \ell \Pr\bigl[Y_i(\nu', \boldsymbol{\rho}') = \ell \;\bigm|\; \nu' \leftarrow \mathcal{N}'\bigr] \\
&\leq (a_i - 1) \sum_{\ell=0}^{(a_i-1)} \Pr\bigl[Y_i(\nu', \boldsymbol{\rho}') = \ell \;\bigm|\; \nu' \leftarrow \mathcal{N}'\bigr] \\
&\quad + N_i \sum_{\ell=a_i}^{N_i} \Pr\bigl[Y_i(\nu', \boldsymbol{\rho}') = \ell \;\bigm|\; \nu' \leftarrow \mathcal{N}'\bigr] \\
&= (a_i - 1) \sum_{\ell=0}^{(a_i-1)} \Pr\bigl[Y_i(\nu', \boldsymbol{\rho}') = \ell \;\bigm|\; \nu' \leftarrow \mathcal{N}'\bigr] \\
&\quad + N_i \left(1 - \sum_{\ell=0}^{(a_i-1)} \Pr\bigl[Y_i(\nu', \boldsymbol{\rho}') = \ell \;\bigm|\; \nu' \leftarrow \mathcal{N}'\bigr]\right) \\
&= N_i - (N_i - a_i + 1) \sum_{\ell=0}^{(a_i-1)} \Pr\bigl[Y_i(\nu', \boldsymbol{\rho}') = \ell \;\bigm|\; \nu' \leftarrow \mathcal{N}'\bigr] \ ,
\end{aligned}
$$

which implies the desired result. □

Definition 4.9. *For every $i \in \{0, \ldots, \mathsf{k}-1\}$ and prefix $\boldsymbol{\rho} \in \mathcal{R}_{[:i+1]}$ We define $\epsilon(\boldsymbol{\rho})$ to be the fraction of $\rho_{i+1}, \ldots, \rho_{\mathsf{k}}$ such that*

$$
\langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}), \mathbf{V}_{\mathsf{IP}}(\mathbb{x}; \boldsymbol{\rho} \parallel \rho_{i+1} \parallel \cdots \parallel \rho_{\mathsf{k}}) \rangle_{\mathsf{IP}} = 1 \ .
$$

For convinience, we write ϵ instead of $\epsilon(\emptyset)$.

The success probability of $\mathsf{TreeFinder}_0$ has the following lower bound.

Theorem 4.10 ([ACK21])**.** *It holds that*

$$
\Pr\Bigl[T \neq \perp \;\Bigm|\; T \leftarrow \mathsf{TreeFinder}_0(\mathbb{x}, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \perp)\Bigr] \geq \epsilon - \left(1 - \prod_{i=1}^{\mathsf{k}} \left(1 - \frac{a_i - 1}{N_i}\right)\right) \ .
$$

Proof. For every $i \in \{0, \ldots, \mathsf{k}-1\}$, we define

$$
\kappa_i \coloneqq 1 - \prod_{j=i+1}^{\mathsf{k}} \left(1 - \frac{a_j - 1}{N_j}\right) \ . \tag{1}
$$

Moreover, we let $\kappa_{\mathsf{k}} \coloneqq 0$. We can exhibit κ_i recursively. Note that for every $i \in \{0, \ldots, \mathsf{k}-2\}$, we have

$$
\begin{aligned}
&\kappa_{i+1} \prod_{j=i+1}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} + \frac{a_{i+1} - 1}{N_{i+1} - a_{i+1} + 1} \\
&= \left(1 - \prod_{j=i+2}^{\mathsf{k}} \left(1 - \frac{a_j - 1}{N_j}\right)\right) \cdot \prod_{j=i+1}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} + \frac{a_{i+1} - 1}{N_{i+1} - a_{i+1} + 1}
\end{aligned}
$$

$$= \left[\left(1 - \prod_{j=i+2}^{\mathsf{k}} \left(1 - \frac{a_j - 1}{N_j} \right) \right) + \frac{a_{i+1} - 1}{N_{i+1}} \prod_{j=i+2}^{\mathsf{k}} \left(1 - \frac{a_j - 1}{N_j} \right) \right] \cdot \prod_{j=i+1}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1}$$

$$= \left[1 - \prod_{j=i+2}^{\mathsf{k}} \left(1 - \frac{a_j - 1}{N_j} \right) \left(1 - \frac{a_{i+1} - 1}{N_{i+1}} \right) \right] \cdot \prod_{j=i+1}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1}$$

$$= \kappa_i \cdot \prod_{j=i+1}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} \ ,$$

which implies

$$\kappa_{i+1} \prod_{j=i+1}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} + \frac{a_{i+1} - 1}{N_{i+1} - a_{i+1} + 1} = \kappa_i \prod_{j=i+1}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} \ . \tag{2}$$

We prove the main statement by induction on i going from k to 0. More precisely, for every $\boldsymbol{\rho}' \in \mathcal{R}_{[:i+1]}$, the induction hypothesis is

$$\Pr\Big[T \neq \bot \ \Big|\ T \leftarrow \mathsf{TreeFinder}_i(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}', \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot) \Big] \geq \left(\prod_{j=i+1}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} \right) (\epsilon(\boldsymbol{\rho}') - \kappa_i) \ ,$$

where $\boldsymbol{\alpha}' \leftarrow \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\rho}')$. We obtain an upper bound for the desired result by fixing $i = 0$. Fix arbitrarly $\boldsymbol{\rho}' \in \mathcal{R}_{[:i+1]}$ and let $\boldsymbol{\alpha}' \leftarrow \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\rho}')$ be the prover's first messages. For every $\nu' \in \mathcal{N}'$, let $\rho_{i+1}^{\mathsf{ea}}(\nu')$ be the first challenges picked by

$$\mathsf{TreeFinder}_i(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}', \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot; \nu') \ .$$

The base case holds trivially since the right-hand side equals zero if $i = \mathsf{k}$. We now prove the inductive step. Fix $i \in \{0, \ldots, \mathsf{k} - 1\}$ and assume the induction hypothesis holds for all $j > i$. Note that $\mathsf{TreeFinder}_i$ succeeds if and only if the first call done to $\mathsf{TreeFinder}_{i+1}$ succeeds and $Y_{i+1}(\nu', \boldsymbol{\rho}) \geq a_{i+1}$. Thus,

$$\Pr\Big[T \neq \bot \ \Big|\ T \leftarrow \mathsf{TreeFinder}_i(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}', \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot) \Big]$$

$$= \sum_{\ell=\underbrace{a_{i+1}}_{\text{Min. \# of suc.}}}^{N_{i+1}} \underbrace{\Pr\left[\begin{array}{l|rcl} T \neq \bot & \nu' & \leftarrow & \mathcal{N}' \\ \text{conditioned on} & \alpha'_{i+1} & \leftarrow & \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}' \parallel \rho_{i+1}^{\mathsf{ea}}(\nu')) \\ Y_{i+1}(\nu', \boldsymbol{\rho}') = \ell & T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \boldsymbol{\alpha}' \parallel \alpha'_{i+1}, \boldsymbol{\rho}' \parallel \rho_{i+1}^{\mathsf{ea}}(\nu'), \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot; \nu') \end{array} \right]}_{\text{First rec. call succeeds}}$$

$$\cdot \Pr\big[Y_{i+1}(\nu', \boldsymbol{\rho}') = \ell \ \big|\ \nu' \leftarrow \mathcal{N}' \big] \ ,$$

which by Lemma 4.7 is

$$= \sum_{\ell=a_{i+1}}^{N_{i+1}} \frac{\ell}{N_{i+1}} \Pr\big[Y_{i+1}(\nu', \boldsymbol{\rho}') = \ell \ \big|\ \nu' \leftarrow \mathcal{N}' \big]$$

$$= \frac{1}{N_{i+1}} \left(\mathbb{E}\big[Y_{i+1}(\nu', \boldsymbol{\rho}') \ \big|\ \nu' \leftarrow \mathcal{N}' \big] - \sum_{\ell=0}^{a_{i+1}-1} \ell \Pr\big[Y_{i+1}(\nu', \boldsymbol{\rho}') = \ell \ \big|\ \nu' \leftarrow \mathcal{N}' \big] \right)$$

$$\geq \frac{1}{N_{i+1}} \left(\mathbb{E}\big[Y_{i+1}(\nu', \boldsymbol{\rho}') \;\big|\; \nu' \leftarrow \mathcal{N}'\big] - (a_{i+1}-1) \sum_{\ell=0}^{a_{i+1}-1} \Pr\big[Y_{i+1}(\nu', \boldsymbol{\rho}') = \ell \;\big|\; \nu' \leftarrow \mathcal{N}'\big] \right) \ .$$

Using Lemma 4.8 ($\sum_{\ell=0}^{a_{i+1}-1} \Pr[Y_{i+1}(\nu', \boldsymbol{\rho}') = \ell \mid \nu' \leftarrow \mathcal{N}'] \leq \frac{N_{i+1} - \mathbb{E}[Y_{i+1}(\nu', \boldsymbol{\rho}') \mid \nu' \leftarrow \mathcal{N}']}{N_{i+1} - a_{i+1} + 1}$).

$$\begin{aligned}
&\geq \frac{1}{N_{i+1}} \left(\mathbb{E}\big[Y_{i+1}(\nu', \boldsymbol{\rho}') \;\big|\; \nu' \leftarrow \mathcal{N}'\big] - (a_{i+1}-1) \frac{N_{i+1} - \mathbb{E}[Y_{i+1}(\nu', \boldsymbol{\rho}') \mid \nu' \leftarrow \mathcal{N}']}{N_{i+1} - a_{i+1} + 1} \right) \\
&= \frac{1}{N_{i+1}} \frac{1}{N_{i+1} - a_{i+1} + 1} \Big(\mathbb{E}\big[Y_{i+1}(\nu', \boldsymbol{\rho}') \;\big|\; \nu' \leftarrow \mathcal{N}'\big](N_{i+1} - a_{i+1} + 1) \\
&\quad - (a_{i+1}-1)(N_{i+1} - \mathbb{E}\big[Y_{i+1}(\nu', \boldsymbol{\rho}') \;\big|\; \nu' \leftarrow \mathcal{N}'\big]) \Big) \\
&= \frac{N_{i+1}}{N_{i+1} - a_{i+1} + 1} \left(\frac{\mathbb{E}[Y_{i+1}(\nu', \boldsymbol{\rho}') \mid \nu' \leftarrow \mathcal{N}']}{N_{i+1}} - \frac{a_{i+1}-1}{N_{i+1}} \right) \ . \qquad (3)
\end{aligned}$$

The final step is to find a lower bound for $\frac{\mathbb{E}[Y_{i+1}(\nu', \boldsymbol{\rho}') \mid \nu' \leftarrow \mathcal{N}']}{N_{i+1}}$. For every $\rho' \in \mathcal{R}_{i+1}$, let

$$\alpha'_{i+2}(\rho') := \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}' \parallel \rho')$$

be the prover's response. Recall that $\left|\mathcal{R}'_{i+1}\right| := N_{i+1}$. By the induction hypothesis, we have

$$\begin{aligned}
&\frac{\mathbb{E}[Y_{i+1}(\nu', \boldsymbol{\rho}') \mid \nu' \leftarrow \mathcal{N}']}{N_{i+1}} \\
&= \frac{1}{N_{i+1}} \cdot \underbrace{\sum_{\rho'_{i+1} \in \mathcal{R}_{i+1}}}_{\forall \text{ challenges}} \Pr\Big[T \neq \perp \;\Big|\; T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \boldsymbol{\alpha}' \parallel \alpha'_{i+2}(\rho'_{i+1}), \boldsymbol{\rho}' \parallel \rho'_{i+1}, \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \perp) \Big] \\
&\geq \frac{1}{N_{i+1}} \cdot \sum_{\rho'_{i+1} \in \mathcal{R}_{i+1}} \left(\prod_{j=i+2}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} \right) (\epsilon(\boldsymbol{\rho}' \parallel \rho'_{i+1}) - \kappa_{i+1}) \\
&= \left(\prod_{j=i+2}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} \right) \left(\frac{1}{N_{i+1}} \cdot \sum_{\rho'_{i+1} \in \mathcal{R}_{i+1}} \epsilon(\boldsymbol{\rho}' \parallel \rho'_{i+1}) - \kappa_{i+1} \frac{1}{N_{i+1}} \cdot \sum_{\rho'_{i+1} \in \mathcal{R}_{i+1}} 1 \right) \\
&= \left(\prod_{j=i+2}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} \right) (\epsilon(\boldsymbol{\rho}') - \kappa_{i+1}) \ .
\end{aligned}$$

Combining this with Eq. (3), we get

$$\begin{aligned}
&\Pr\Big[T \neq \perp \;\Big|\; T \leftarrow \mathsf{TreeFinder}_i(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}', \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \perp) \Big] \\
&\geq \frac{N_{i+1}}{N_{i+1} - a_{i+1} + 1} \left[\left(\prod_{j=i+2}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} \right) (\epsilon(\boldsymbol{\rho}') - \kappa_{i+1}) - \frac{a_{i+1}+1}{N_{i+1}} \right] \\
&= \epsilon(\boldsymbol{\rho}') \frac{N_{i+1}}{N_{i+1} - a_{i+1} + 1} \prod_{j=i+2}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1}
\end{aligned}$$

$$
\begin{aligned}
&- \kappa_{i+1} \frac{N_{i+1}}{N_{i+1} - a_{i+1} + 1} \prod_{j=i+2}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} - \frac{a_{i+1} + 1}{N_{i+1} - a_{i+1} + 1} \\
&= \epsilon(\boldsymbol{\rho}') \prod_{j=i+1}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} - \kappa_{i+1} \prod_{j=i+1}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} - \frac{a_{i+1} + 1}{N_{i+1} - a_{i+1} + 1} \\
&= \epsilon(\boldsymbol{\rho}') \prod_{j=i+1}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} - \left(\kappa_{i+1} \prod_{j=i+1}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} + \frac{a_{i+1} + 1}{N_{i+1} - a_{i+1} + 1} \right) .
\end{aligned}
$$

Injecting Eq. (2), it follows that

$$
\Pr\Big[T \neq \perp \;\Big|\; T \leftarrow \mathsf{TreeFinder}_i(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}', \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \perp) \Big] \geq \prod_{j=i+1}^{\mathsf{k}} \frac{N_j}{N_j - a_j + 1} \big(\epsilon(\boldsymbol{\rho}') - \kappa_i \big) \ .
$$

□

While we know a lower bound for the success probability if $\mathsf{TreeFinder}_i$ samples each of its challenges ($\mathfrak{f} = \perp$), we want to know what it is when $\mathsf{TreeFinder}_0$ is called with $\mathfrak{f} = \top$, since it is the success probability of $\mathbf{E}_{\mathsf{IP}}^{\mathsf{SKS}}$ (see Lemma 4.3). Hence, we seek to find an expression that relates their success probabilities.

Definition 4.11. *For every verifier randomness $\boldsymbol{\rho}$ and extractor randomness $\nu \in \mathcal{N}$, we define the* **tree of challenges** *$\mathcal{T}(\nu, \boldsymbol{\rho})$ to be the $(N_1, \ldots, N_{\mathsf{k}})$-tree of challenges that would pick*

$$
\mathsf{TreeFinder}_0(\mathbb{x}, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \top; \nu)
$$

if it, and all of its recursive calls, ignore the early abort strategy and always iterate through all challenges (even though it can return a tree of accepting transcripts for $\mathbb{x}$), where $\boldsymbol{\alpha} \leftarrow \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\rho})$ are prover's messages. Similarly for every $\nu' \in \mathcal{N}'$, the symbol $\mathcal{T}'(\nu')$ denotes the tree of picked challenges that would pick

$$
\mathsf{TreeFinder}_0(\mathbb{x}, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \perp; \nu')
$$

with the same modification.

An execution of $\mathsf{TreeFinder}_0$, where $\mathfrak{f}$ is $\top$ or $\perp$, is completely characterized by its respective tree of challenges. Moreover, the two instantiations compute the same tree if these trees are the same.

Lemma 4.12. *For every $\nu \in \mathcal{N}$, $\nu' \in \mathcal{N}'$ and $\boldsymbol{\rho} \in \boldsymbol{\mathcal{R}}$, if $\mathcal{T}(\nu, \boldsymbol{\rho}) = \mathcal{T}'(\nu')$, then*

$$
\mathsf{TreeFinder}_0(\mathbb{x}, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \top; \nu)
$$

succeeds if and only if

$$
\mathsf{TreeFinder}_0(\mathbb{x}, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \perp; \nu') \ ,
$$

succeeds. Also, for any $\boldsymbol{\rho} \in \boldsymbol{\mathcal{R}}$, then for all randomness $\nu \in \mathcal{N}$, there exists a randomness ν' such that $\mathcal{T}(\nu, \boldsymbol{\rho}) = \mathcal{T}'(\nu')$; we denote by ϕ this transformation, which maps every pair $(\nu, \boldsymbol{\rho})$ to their corresponding ν'. Note that the function ϕ is a bijection between $\mathcal{N} \times \boldsymbol{\mathcal{R}}$ and $\mathcal{N}'$.

Proof. Both instantiations discover the same tree. Therefore, they always succeed or fail together. For the second part of the proof, we construct ϕ explicitly. The latter fixes ν' so the left-most branch of $\mathcal{T}'(\nu')$ is composed of edges labeled with $\boldsymbol{\rho}$ (and the nodes with the respective prover's responses) and such that the remaining challenges are picked by

$$\mathsf{TreeFinder}_0(\mathbb{x}, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot; \nu') \ ,$$

accordingly to ν. It is straightforward to see that this function has the property we seek. □

Lemma 4.12 implies the following corollary.

Corollary 4.13. *It holds that*

$$\begin{aligned}
&\Pr\left[T \neq \bot \;\middle|\; \begin{array}{lcl} \nu' & \leftarrow & \mathcal{N}' \\ T & \leftarrow & \mathsf{TreeFinder}_0(\mathbb{x}, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \bot; \nu') \end{array}\right] \\
&= \Pr\left[T \neq \bot \;\middle|\; \begin{array}{lcl} \nu & \leftarrow & \mathcal{N} \\ \boldsymbol{\rho} & \leftarrow & \mathcal{R} \\ b & \xleftarrow{\boldsymbol{\alpha},\boldsymbol{\rho}} & \langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}), \mathbf{V}_{\mathsf{IP}}(\mathbb{x}; \boldsymbol{\rho})\rangle_{\mathsf{IP}} \\ T & \leftarrow & \mathsf{TreeFinder}_0(\mathbb{x}, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \top; \nu) \end{array}\right] .
\end{aligned}$$

Proof. The proof follows directly from the definition of ϕ. □

Consider the two events $(\mathbb{x}, \mathbb{w}) \in R$ and $b = 0$. We claim that they must be mutually exclusive. Indeed, if $b = 0$, then the given transcript is not accepting, and thus, $\mathsf{TreeFinder}_0$ will return $\bot$. Therefore, taking the negation of the knowledge soundness part of Theorem 4.1 results in

$$\begin{aligned}
&\Pr\left[(\mathbb{x}, \mathbb{w}) \in R \;\middle|\; \begin{array}{lcl} b & \xleftarrow{(\boldsymbol{\alpha},\boldsymbol{\rho})} & \langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}), \mathbf{V}_{\mathsf{IP}}(\mathbb{x})\rangle_{\mathsf{IP}} \\ \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SKS}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \tilde{\mathbf{P}}_{\mathsf{IP}}) \end{array}\right] \\
&+ \Pr\left[b = 0 \;\middle|\; \begin{array}{lcl} b & \xleftarrow{(\boldsymbol{\alpha},\boldsymbol{\rho})} & \langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}), \mathbf{V}_{\mathsf{IP}}(\mathbb{x})\rangle_{\mathsf{IP}} \\ \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SKS}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \tilde{\mathbf{P}}_{\mathsf{IP}}) \end{array}\right] \geq \prod_{i=1}^{\mathsf{k}} \left(1 - \frac{a_i - 1}{N_i}\right) \ . \qquad (4)
\end{aligned}$$

Leveraging Lemma 4.3 we can rewrite the experimental part of the probability as

$$\begin{aligned}
&\Pr\left[T \neq \bot \;\middle|\; \begin{array}{lcl} \nu & \leftarrow & \mathcal{N} \\ \boldsymbol{\rho} & \leftarrow & \mathcal{R} \\ b & \xleftarrow{\boldsymbol{\alpha},\boldsymbol{\rho}} & \langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}), \mathbf{V}_{\mathsf{IP}}(\mathbb{x}; \boldsymbol{\rho})\rangle_{\mathsf{IP}} \\ T & \leftarrow & \mathsf{TreeFinder}_0(\mathbb{x}, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \top; \nu) \end{array}\right] \\
&\overbrace{+ \Pr\left[b = 0 \;\middle|\; \begin{array}{lcl} \nu & \leftarrow & \mathcal{N} \\ \boldsymbol{\rho} & \leftarrow & \mathcal{R} \\ b & \xleftarrow{\boldsymbol{\alpha},\boldsymbol{\rho}} & \langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}), \mathbf{V}_{\mathsf{IP}}(\mathbb{x}; \boldsymbol{\rho})\rangle_{\mathsf{IP}} \end{array}\right]}^{=1-\epsilon} \\
&= \Pr\left[T \neq \bot \;\middle|\; \begin{array}{lcl} \nu & \leftarrow & \mathcal{N} \\ \boldsymbol{\rho} & \leftarrow & \mathcal{R} \\ b & \xleftarrow{\boldsymbol{\alpha},\boldsymbol{\rho}} & \langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}), \mathbf{V}_{\mathsf{IP}}(\mathbb{x}; \boldsymbol{\rho})\rangle_{\mathsf{IP}} \\ T & \leftarrow & \mathsf{TreeFinder}_0(\mathbb{x}, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \top; \nu) \end{array}\right] + (1 - \epsilon) \qquad \text{Corollary 4.13}
\end{aligned}$$

$$
\begin{aligned}
&= \Pr\left[T \neq \perp \;\middle|\; \begin{array}{lcl} \nu' & \leftarrow & \mathcal{N}' \\ T & \leftarrow & \mathsf{TreeFinder}_0(\mathbb{x}, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \perp; \nu') \end{array}\right] + (1-\epsilon) && \text{Theorem 4.10} \\
&\geq \left(\prod_{i=1}^{\mathsf{k}} \frac{N_i}{N_i - a_i + 1}\right)\left(\epsilon - \left(1 - \prod_{i=1}^{\mathsf{k}}\left(1 - \frac{a_i - 1}{N_i}\right)\right)\right) + (1-\epsilon) \\
&\geq \epsilon - \left(1 - \prod_{i=1}^{\mathsf{k}}\left(1 - \frac{a_i - 1}{N_i}\right)\right) + (1-\epsilon) \\
&= \prod_{i=1}^{\mathsf{k}}\left(1 - \frac{a_i - 1}{N_i}\right) \ ,
\end{aligned}
$$

as desired.

Remark 4.14. The strong knowledge soundness error given in the theorem is tight: there is a malicious prover such that the strong knowledge soundness error is exactly κ_0. To see this, consider the following malicious prover $\tilde{\mathbf{P}}_{\mathsf{IP}}$. For every round i, it randomly guesses $a_i - 1$ challenges. If and only if one of the challenges picked by the verifier is one of the guessed challenges, then the prover convinces the verifier. Hence, $\epsilon = \kappa_0$. Moreover, $\mathbf{E}_{\mathsf{IP}}^{\mathsf{SKS}}$ cannot find a witness since the prover does not guess enough challenges. Therefore, Eq. (4) is tight, as is the strong knowledge soundness error.

4.3 Extraction time

We prove the following lemma.

Lemma 4.15. *For every $i \in \{0, \ldots, \mathsf{k}\}$, $\boldsymbol{\rho}' \in \mathcal{R}_{[:i+1]}$ and exractor randomness ν', we have*

$$
\mathbb{E}\big[I_i'(\nu', \boldsymbol{\rho}') \;\big|\; \nu' \leftarrow \mathcal{N}'\big] \leq \begin{cases} 1 & \text{if } i = \mathsf{k}, \\ \prod_{j=i+1}^{\mathsf{k}} a_j & \text{otherwise,} \end{cases}
$$

where $I_i'(\nu', \boldsymbol{\rho}')$ denotes the number of invocations to $\tilde{\mathbf{P}}_{\mathsf{IP}}$ done by

$$
\mathsf{TreeFinder}_i(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}', \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \perp; \nu')
$$

and $\boldsymbol{\alpha}' \leftarrow \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\rho}')$ are the first prover's messages.

Proof. Let us consider a slightly modified but equivalent version of algorithm $\mathsf{TreeFinder}_i$. For every challenge ρ_{i+1}' it picks, instead of calling $\tilde{\mathbf{P}}_{\mathsf{IP}}$ to get the respective response α_{i+1}', the corresponding call to $\mathsf{TreeFinder}_{i+1}$ is tasked to returns α_{i+1}' in addition to returning a tree of accepting transcript for $\mathbb{x}$. Hence, if we forget about the recursive calls, each execution of $\mathsf{TreeFinder}_i$, for $i \in \{0, \ldots, \mathsf{k}\}$, does at most one invocation of $\tilde{\mathbf{P}}_{\mathsf{IP}}$. This modified version of $\mathsf{TreeFinder}_i$ is more straightforward to analyze.

We prove this lemma by induction on i, going from k to 0. If $i = \mathsf{k}$, the algorithm needs exactly one invocation to obtain the last prover's message; this is a direct consequence of our modification since the original algorithm would have done zero invocation. Fix now $i \in \{0, \ldots, \mathsf{k} - 1\}$ and $\boldsymbol{\rho}' \in \mathcal{R}_{[:i+1]}$. Let $\boldsymbol{\alpha}' \leftarrow \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\rho}')$ be the first prover's messages. We want to upper bound the number of invocations to the oracle $\tilde{\mathbf{P}}_{\mathsf{IP}}$ done by

$$
\mathsf{TreeFinder}_i(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}', \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \perp) \ .
$$

Consider these two distinct scenarios.

Its first recursive call aborts. In that case, the algorithm will abort right after the first recursive call. Therefore, the number of invocations is just the expected number of invocations done by the recursive call; that is

$$\mathbb{E}\big[I'_{i+1}\big(\nu', \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu')\big) \;\big|\; \nu' \leftarrow \mathcal{N}'\big] \enspace ,$$

where $\rho^{\mathsf{ea}}_{i+1}(\nu')$ is the first challenges picked by

$$\mathsf{TreeFinder}_i(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}', \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot; \nu') \enspace ,$$

or, in other words, the challenge used for the early abort test. Note that we have intentionally omitted the dependency on $\boldsymbol{\rho}'$ in the symbol $\rho^{\mathsf{ea}}_{i+1}(\nu')$ to alleviate notation.

Its first recursive call does not abort. First, we must consider the number of invocations performed by the first recursive call, as in the previous case. Then, it remains to compute the number of invocations done within the loop. To do so, we compute the number of iterations performed in the loop and then multiply it by the number of invocations done by a single recursive call. Remark that this works because challenges are picked uniformly at random, independently of everything else. If $Y_i < a_{i+1} - 1$, $\mathsf{TreeFinder}_i$ will sample and check $N_{i+1} - 1$ challenges before realizing that there are not enough $(a_{i+2}, \ldots, a_{\mathsf{k}})$-tree of accepting transcripts for $\mathbb{x}$ with prefix $\boldsymbol{\rho}'$. On the other hand, if $Y_{i+1} \geq a_{i+1}$, the number of iterations will follow the NHG distribution, which has expectation $\mu_{N_{i+1}-1, Y_{i+1}-1, a_{i+1}} \geq N_{i+1} - 1$ (see Definition 2.14). Hence, in both cases, conditioned on $\ell = Y_{i+1}$ and the early abort test being true, the expected number of iterations done in the loop is upper bounded by

$$N_{i+1} \cdot \frac{a_{i+1} - 1}{\ell} \enspace .$$

For the second case, we want a more fine-grained analysis. We divide the iterations done in the loop into two categories: the ones that result in a succeeding recursive call, the *successful iterations*, and the ones that result in the recursive call returning $\bot$, the *unsuccessful iterations*. The number of successful iterations if $Y_i < a_{i+1}$ is $Y_i \leq a_{i+1} - 1$ by Y's definition. Otherwise, if $Y_i \geq a_{i+1}$ then the extractor will invoke exactly a_{i+1} good challenges for $(\nu', \boldsymbol{\rho}')$ before returing a $(a_{i+2}, \ldots, a_{\mathsf{k}})$-tree of accepting transcripts for $\mathbb{x}$ with prefix $\boldsymbol{\rho}'$, be construction of the algorithm. To get an upper bound for the number of unsuccessful iterations, we subtract the number of successful iterations from the total number of iterations, which we have computed above. For this sake, we let

$$p \coloneqq \Pr\left[\begin{array}{l|rcl} T \neq \bot & \nu' & \leftarrow & \mathcal{N}' \\ \text{conditioned on} & \alpha'_{i+1} & \leftarrow & \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu')) \\ Y_{i+1}(\nu', \boldsymbol{\rho}') = \ell & T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \boldsymbol{\alpha}' \parallel \alpha'_{i+1}, \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu'), \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot) \end{array}\right] \tag{5}$$

be Lemma 4.7's probability expression and $\bar{p}$ its complement. Table 1 summarizes the upper bounds resulting from this more precise analysis.

Now, we can give a recursive expression for the expected number of recursive calls conditioned on $Y_{i+1} = \ell$. Note that the probability p can be seen as the probability that the first challenge picked by the extractor is good. By conditioning on the first challenge picked by $\mathsf{TreeFinder}_i$ being good or not and by Lemma 4.7, we have for every $\boldsymbol{\rho}' \in \mathcal{R}_{[:i+1]}$,

$$\begin{aligned} &\mathbb{E}\big[I'_i(\nu', \boldsymbol{\rho}') \text{ conditioned on } Y_{i+1}(\nu', \boldsymbol{\rho}') = \ell \;\big|\; \nu' \leftarrow \mathcal{N}'\big] \\ &\leq \overbrace{\mathbb{E}\big[I'_{i+1}\big(\nu', \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu')\big) \text{ conditioned on } Y_{i+1}(\nu', \boldsymbol{\rho}') = \ell \;\big|\; \nu' \leftarrow \mathcal{N}'\big]}^{\text{First recursive call}} \end{aligned}$$

	$\ell \geq a_{i+1}$	$\ell < a_{i+1}$
Total calls	$\frac{(a_{i+1}-1)N_{i+1}}{\ell}$	$N_{i+1} - 1 \leq \frac{(a_{i+1}-1)N_{i+1}}{\ell}$
Successful calls	$a_{i+1} - 1$	$\ell - 1 < a_{i+1} - 1$
Unsuccessful calls	$\frac{(a_{i+1}-1)N_{i+1}}{\ell} - (a_{i+1} - 1)$ $= \bar{p} \cdot \frac{(a_{i+1}-1)N_{i+1}}{\ell}$	$N_{i+1} - \ell$ $= \bar{p} \cdot N_{i+1}$ $\leq \bar{p} \cdot \frac{(a_{i+1}-1)N_{i+1}}{\ell}$

Table 1: Expected number of iterations conditioned on $\ell = Y_{i+1}$ and the first recursive call not aborting. The quantity $\bar{p}$ is the complement of Eq. (5).

$$
\begin{aligned}
&+ \underbrace{p}_{\text{First call successful}} \cdot \Bigg\{ \overbrace{(a_{i+1}-1)}^{\substack{\text{\# of successful calls} \\ (\ell \geq a_{i+1} \,\vee\, \ell < a_{i+1})}} \\
&\quad \cdot \underbrace{\mathbb{E}\left[\begin{array}{l|rcl} I'_{i+1}\big(\nu', \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu')\big) & \nu' & \leftarrow & \mathcal{N}' \\ \text{conditioned on} & \alpha'_{i+1}(\nu') & \leftarrow & \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu')) \\ Y_{i+1}(\nu', \boldsymbol{\rho}') = \ell & T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \boldsymbol{\alpha}' \parallel \alpha'_{i+1}(\nu'), \\ \wedge\, T \neq \bot & & & \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu'), \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot) \end{array}\right]}_{\text{Recursive calls for successful cases}} \\
&+ \underbrace{\bar{p} \cdot \frac{(a_{i+1}-1)N_{i+1}}{\ell}}_{\text{\# of failed calls } (\ell \geq a_{i+1} \,\vee\, \ell < a_{i+1})} \\
&\quad \cdot \underbrace{\mathbb{E}\left[\begin{array}{l|rcl} I'_{i+1}\big(\nu', \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu')\big) & \nu' & \leftarrow & \mathcal{N}' \\ \text{conditioned on} & \alpha'_{i+1}(\nu') & \leftarrow & \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu')) \\ Y_{i+1}(\nu', \boldsymbol{\rho}') = \ell & T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \boldsymbol{\alpha}' \parallel \alpha'_{i+1}(\nu'), \\ \wedge\, T = \bot & & & \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu'), \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot) \end{array}\right]}_{\text{Recursive calls for failed cases}} \Bigg\} \\
&= \mathbb{E}\big[I'_{i+1}\big(\nu', \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu')\big) \text{ conditioned on } Y_{i+1}(\nu', \boldsymbol{\rho}') = \ell \;\big|\; \nu' \leftarrow \mathcal{N}'\big] + (a_{i+1} - 1) \\
&\quad \cdot \Bigg\{ p \cdot \mathbb{E}\left[\begin{array}{l|rcl} I'_{i+1}\big(\nu', \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu')\big) & \nu' & \leftarrow & \mathcal{N}' \\ \text{conditioned on} & \alpha'_{i+1}(\nu') & \leftarrow & \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu')) \\ Y_{i+1}(\nu', \boldsymbol{\rho}') = \ell & T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \boldsymbol{\alpha}' \parallel \alpha'_{i+1}(\nu'), \\ \wedge\, T \neq \bot & & & \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu'), \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot) \end{array}\right] \\
&\qquad + \bar{p} \cdot \mathbb{E}\left[\begin{array}{l|rcl} I'_{i+1}\big(\nu', \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu')\big) & \nu' & \leftarrow & \mathcal{N}' \\ \text{conditioned on} & \alpha'_{i+1}(\nu') & \leftarrow & \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu')) \\ Y_{i+1}(\nu', \boldsymbol{\rho}') = \ell & T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \boldsymbol{\alpha}' \parallel \alpha'_{i+1}(\nu'), \\ \wedge\, T = \bot & & & \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu'), \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot) \end{array}\right] \Bigg\} \\
&= a_{i+1} \cdot \mathbb{E}\big[I'_{i+1}\big(\nu', \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu')\big) \text{ conditioned on } Y_{i+1}(\nu', \boldsymbol{\rho}') = \ell \;\big|\; \nu' \leftarrow \mathcal{N}'\big] \;,
\end{aligned}
$$

which directly implies

$$
\mathbb{E}\big[I'_i(\nu', \boldsymbol{\rho}') \;\big|\; \nu' \leftarrow \mathcal{N}'\big] \leq a_{i+1}\, \mathbb{E}\big[I'_{i+1}\big(\nu', \boldsymbol{\rho}' \parallel \rho^{\mathsf{ea}}_{i+1}(\nu')\big) \;\big|\; \nu' \leftarrow \mathcal{N}'\big] \leq \prod_{j=i+1}^{\mathsf{k}} a_j \;,
$$

where the final step follows from the induction hypothesis. □

Let $I_i(\nu, \boldsymbol{\rho}', \boldsymbol{\rho})$ denotes the number of invocations done by

$$\mathsf{TreeFinder}_i(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\rho}', \tilde{\mathbf{P}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \top; \nu) \ ,$$

where $\boldsymbol{\alpha}' \leftarrow \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\rho}')$ and $\boldsymbol{\alpha} \leftarrow \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\rho})$. To get I_0 from I'_0, we use the same idea as in Corollary 4.13's proof; namely, we leverage ϕ to construct an expression relating I_0 and I'_0.

Lemma 4.16. *For every instance $\mathbb{x}$ and deterministic malcious prover $\tilde{\mathbf{P}}_{\mathsf{IP}}$, the following equality holds:*

$$\mathbb{E}\left[I'_0(\nu', \emptyset) \mid \nu' \leftarrow \mathcal{N}'\right] = \mathbb{E}\left[I_0(\phi(\nu, \boldsymbol{\rho}), \emptyset, \boldsymbol{\rho}) \;\middle|\; \begin{array}{ccc} \nu & \leftarrow & \mathcal{N} \\ \boldsymbol{\rho} & \leftarrow & \mathcal{R} \end{array}\right] - 1 \ .$$

Proof. If $\tilde{\mathbf{P}}_{\mathsf{IP}}$ is deterministic and $\mathcal{T}(\nu, \boldsymbol{\rho}) = \mathcal{T}'(\nu')$, then the number of challenges picked by

$$\mathsf{TreeFinder}_0(\mathbb{x}, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \top; \nu)$$

and

$$\mathsf{TreeFinder}_0(\mathbb{x}, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\mathsf{IP}}, \cdot, \cdot, \bot; \nu') \ ,$$

where $\boldsymbol{\alpha} \leftarrow \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\rho})$, are identical. However, the responses of the left-most branch, *i.e.* $\boldsymbol{\alpha}$, are given for free in the former case (they are given as a parameter to $\mathsf{TreeFinder}_0$). Thus, one less invocation is needed (recall that an invocation corresponds to one complete protocol execution). Finally, the analysis above applies to the statement we are proving since by definition of ϕ, it holds that $\mathcal{T}(\nu, \boldsymbol{\rho}) = \mathcal{T}'(\phi(\nu, \boldsymbol{\rho}))$. □

We continue with a short sanity check.

Remark 4.17. In the SP case, *i.e.* $\mathsf{k} = 1$, we have by Lemma 4.15 that $I'_0 \leq a$ and thus by Lemma 4.16: $I_0 \leq a - 1$ as expected.

By combining Lemmas 4.15 and 4.16 it results that the number of invocations to $\tilde{\mathbf{P}}_{\mathsf{IP}}$ performed by our extractor is upper bounded by $\left(\prod_{i=1}^{\mathsf{k}} a_i\right) - 1$. Therefore, the complete extraction time of $\mathbf{E}_{\mathsf{IP}}^{\mathsf{SKS}}$ is smaller than

$$\left(\left(\prod_{i=1}^{\mathsf{k}} a_i\right) - 1\right) \cdot \left(\tau_{\tilde{\mathbf{P}}_{\mathsf{IP}}} + \mathsf{poly}(|\mathbb{x}|)\right) \ ,$$

as expected.

5 State-restoration knowledge soundness from special soundness for SPs

Let $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}$ be any state restoration malicious prover, also a t-query oracle algorithm. Generally, the rewinding state restoration knowledge soundness error of an interactive proof with round complexity k has a t^{k} multiplicative loss compared to the rewinding knowledge soundness error. This induces a similar loss in the Fiat–Shamir transformation. However, suppose the protocol is special sound. In that case, the loss reduces to $t + 1$, allowing us to prove the security of the non-interactive Fiat–Shamir transformation. This section deals with the formal statement of this claim and its proof in the case of SPs. We proceed to the more challenging case of IPs in Section 6.

In reality, as with PCPs, soundness is enough to ensure state-restoration knowledge soundness in the case of SPs. However, the analysis we do in this section gives the necessary intuition for the case of IP and, thus, serves as a stepping stone for this more challenging case.

As in Section 3, we first focus on the SPs case before turning in Section 4 to public-coin IPs with arbitrary many rounds. The first result we are interested in is the following.

Theorem 5.1. *Let* $\mathsf{SP} = (\mathbf{P}_{\mathsf{SP}}, \mathbf{V}_{\mathsf{SP}})$ *be a SP for a relation* R *with special soundness with arity* a *and verifier randomness complexity* r. *It follows that* SP *has rewinding state restoration knowledge soundness error*

$$\kappa^{\mathsf{sr}}_{\mathsf{SP}}(\mathbb{x}, t) \leq (t+1)\frac{a-1}{2^{\mathsf{r}}}$$

and extraction time

$$\mathsf{et}^{\mathsf{sr}}_{\mathsf{SP}}(n, t, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \leq (a-1)(t+1) \cdot (\tau_{\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}} + \mathsf{poly}(n)) \ .$$

Moreover, the number of times the knowledge soundness extractor invokes $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}$ *is* $(a-1)(t+1)$.

In [AFK22], a similar result is proven more directly: they directly prove that special soundness implies that the Fiat–Shamir transformation is knowledge sound under the ROM for SPs and IPs, without using the concept of state restoration. Note, however, that the result we want to prove is stronger than theirs since we guarantee a security property between special soundness and the security of the NIROP resulting from the Fiat–Shamir transformation.

We first construct an extractor and then prove respectively in Sections 5.2 and 5.3 that it has the correct knowledge soundness error and extraction time. Let SP be any SP with a-speical soundnes for a relation R. Let s be the salt's length, and finally, let $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}$ be any deterministic state restoration malicious prover, also a t-query oracle algorithm.

5.1 Construction of the extractor

Let $\mathbf{E}^{\mathsf{SS}}_{\mathsf{SP}}$ be the special sound extractor for SP. We construct an extractor $\mathbf{E}^{\mathsf{SRKS}}_{\mathsf{SP}}$ based on its special soundness extractor as in Section 3. Intuitively, the extractor, given $(\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2, \mathsf{tr}^{\mathsf{sr}}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$, runs $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}$ multiple times to find, if possible, an a-tree of accepting transcript for $\mathbb{x}$, similarly to Construction 3.3. Since $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}$ is a random-oracle algorithm, we must provide and simulate a fake random oracle to $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}$. For this purpose, we implement a lazy random oracle with the following properties:

- It returns a new challenge ρ' instead of the given one ρ.
- It is consistent with what has been already queried in the previous run of Γ^{sr}, that is, $\mathsf{tr}^{\mathsf{sr}}$; and
- It answers other queries at random;

To construct this lazy random oracle, we start by picking a new random oracle that is restricted to answer as in $\mathsf{tr}^{\mathsf{sr}}$. This way, the two last properties are satisfied. Then, we modify the random oracle so it answers ρ' when it receives $(\mathbb{x}, \alpha_1, \sigma)$, where ρ' is the newly picked challenge that we want to try. If $(\mathbb{x}, \alpha_1, \sigma)$ is appearing in $\mathsf{tr}^{\mathsf{sr}}$, we simply overwrite its designated output. For this purpose, we define the following notation.

Notation 11. *Let* $\mathsf{r} \in \mathbb{N}^*$ *and* $\mathsf{tr} = ((x_1, y_1), \ldots, (x_\ell, y_\ell))$*, where* $x_i \in \{0,1\}^*$ *and* $y_i \in \{0,1\}^{\mathsf{r}}$*, for* $i \in [\ell]$*. Then* $\mathcal{U}(\mathsf{r})|_{\mathsf{tr}}$ *denotes the set of random oracles* rnd *with output size* r *such that* $\mathsf{rnd}(x_i) = y_i$ *for every* $i \in [\ell]$*. Moreover, given any random oracle* rnd*, we denote by* $\mathsf{rnd}|_{\mathsf{tr}}$ *a new random oracle* rnd' *such that* $\mathsf{rnd}'(x_i) = y_i$ *for every* $i \in [\ell]$ *and such that* $\mathsf{rnd}'(x) = \mathsf{rnd}(x)$ *for every* $x \notin \{x_1, \ldots, x_\ell\}$*. The bar operator is called a* **restriction***.*

We now have all the required tools to introduce the extractor $\mathbf{E}^{\mathsf{SRKS}}_{\mathsf{SP}}$.

Construction 5.2. The rewinding state restoration knowledge soundness extractor is defined as follows.

$\mathbf{E}^{\mathsf{SRKS}}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2, \mathsf{tr}^{\mathsf{sr}}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$:
1. Get the success bit $b \leftarrow \mathbf{V}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho, \alpha_2)$.
2. If $b = 0$ or $|\mathbb{x}| > n$ return $\perp$.
3. Store the good challenge/prover's message pair $B \coloneqq \{(\alpha_1, \rho)\}$ and picked challenge $S \coloneqq \{\rho\}$.
4. Sample a lazy random oracle that matches the given trace $\dot{\mathsf{rnd}} \leftarrow \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}}$.
5. While it remains unpicked challenges $|\mathcal{R} \setminus S| > 0$ and we have not found enough good challenges $|B| \neq a$:
   (a) Get a new challenge unpicked, $\rho' \leftarrow \mathcal{R} \setminus S$..
   (b) Add the new challenge to a set of picked challenges $S \coloneqq S \cup \{\rho'\}$.
   (c) Update the random oracle so it returns the new challenge $\dot{\mathsf{rnd}}' \leftarrow \dot{\mathsf{rnd}}|_{((\mathbb{x},\alpha_1,\sigma),\rho')}$.
   (d) Run the state restoration game $(\mathbb{x}', \alpha_1', \sigma', \rho'', \alpha_2') \leftarrow \Gamma^{\mathsf{sr}}(s, \dot{\mathsf{rnd}}', \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$.
   (e) Compute the decision bit $b' \leftarrow \mathbf{V}_{\mathsf{SP}}(\mathbb{x}, \alpha_1', \rho'', \alpha_2')$.
   (f) If the transcript is accepting $b' = 1$ and the instance, prover's first message and salt and are the same $\mathbb{x}' = \mathbb{x} \wedge \alpha_1' = \alpha_1 \wedge \sigma' = \sigma$, add the good instance/message pair $B \coloneqq B \cup \{(\alpha_1', \rho')\}$.
6. If $|B| = a$, create an a-fork $\mathbf{F} \coloneqq (\alpha_1, B)$ and return $\mathbf{E}^{\mathsf{SS}}_{\mathsf{SP}}(\mathbb{x}, \mathbf{F})$, otherwise return $\perp$.

We represent $\dot{\mathsf{rnd}}$ using lazy sampling: the algorithm only keeps track of what has been queried by the extractor and picks at random for everything else. Once it enters Line 6's conditional statement, it will return a correct witness for $\mathbb{x}$ since we invoke $\mathbf{E}^{\mathsf{SS}}_{\mathsf{SP}}$ on an a-tree of accepting transcripts for $\mathbb{x}$.

Let

$$(\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$$

be the execution of the state restoration game done in the experiment of the state restoration knowledge soundness definition. We refer to this run as the *original run*.

At each iteration of Construction 5.2's loop, we pick a new distinct challenge ρ' we want to implement the new random oracle as

$$\dot{\mathsf{rnd}}' \leftarrow \dot{\mathsf{rnd}}|_{((\mathbb{x},\alpha_1,\sigma),\rho')} \ , \tag{6}$$

where $\dot{\mathsf{rnd}} \leftarrow \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}}$. Let

$$(\mathbb{x}', \alpha_1', \sigma', \rho'', \alpha_2') \leftarrow \Gamma^{\mathsf{sr}}(s, \dot{\mathsf{rnd}}', \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$$

be the call to the state restoration game done at Line 5d with the random oracle of Eq. (6).

We want the value returned by $\Gamma^{\mathsf{sr}}(s, \dot{\mathsf{rnd}}', \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$ to be $(\mathbb{x}, \alpha_1, \sigma, \rho', \alpha_2')$ for some $\alpha_2' \in \{0,1\}^{\mathsf{pv}_2}$, or formally,

$$\mathbb{x}' = \mathbb{x} \wedge \alpha_1' = \alpha_1 \wedge \sigma' = \sigma \wedge \rho'' = \rho \ . \tag{7}$$

Indeed, if Eq. (7) does not hold, we cannot ensure that the returned challenge ρ'' will be ρ', as desired. Moreover, even if the returned challenge is ρ', it does not imply that the returned instance is $\mathbb{x}$ and that the first prover message is the same as α_1, which are both necessary if we seek to construct a a-tree of accepting transcripts for $\mathbb{x}$ that from the branch $(\alpha_1, \rho, \alpha_2)$ resulting from the original run, similarly to what Construction 3.3 does. Notice that we only test for

$$\mathbb{x}' = \mathbb{x} \wedge \alpha_1' = \alpha_1 \wedge \sigma' = \sigma \tag{8}$$

in Line 5f of the extractor. These tests are sufficient because they are equivalent to Eq. (7)'s ones since $\mathbb{x}' = \mathbb{x}$, $\alpha_1' = \alpha$ and $\sigma' = \sigma$ implies $\rho'' = \rho'$, since ρ'' is the result of a deterministic function evaluated at $(\mathbb{x}', \alpha_1', \sigma')$ which is defined to return ρ' if Eq. (8) holds. We refer to these tests as the *consistency tests*. More formally, we define the following notion.

Definition 5.3. *A challenge $\rho \in \mathcal{R}$ is said to be* **consistent with** $(\mathbb{x}, \alpha_1, \sigma, \mathsf{rnd})$ *if*

- $|\mathbb{x}| \leq n$*; and*
- $(\mathbb{x}, \alpha_1, \sigma, \rho', \alpha_2') \leftarrow \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}|_{((\mathbb{x},\alpha_1,\sigma),\rho)}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$*, for some* $\alpha_2' \in \{0,1\}^{\mathsf{pv}_2}$.

We illustrate in Fig. 7 the implementation of the random oracle $\dot{\mathsf{rnd}}'$. This random oracle starts by acting like rnd and then, on the query $(\mathbb{x}, \alpha_1, \sigma)$, returns ρ' instead of the challenge ρ that would have been returned by $\Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$. After that, $\dot{\mathsf{rnd}}'$ answers the queries randomly. At this point, we do not have to check for consistency with what has been queried by $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}$ before $(\mathbb{x}, \alpha_1, \sigma)$ since we have assumed that all queries were distinct.

One may ask what if $(\mathbb{x}, \alpha_1, \sigma)$ is never queried by $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}$ in $\Gamma^{\mathsf{sr}}(s, \mathsf{rnd}|_{((\mathbb{x},\alpha_1,\sigma),\rho)}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$. This query would not have happened in the original run if this had been the case. However, this makes little sense for the malicious prover since it could not infer any information on the challenge picked by the state restoration game due to the nature of random oracles.

If the consistency tests are passed, we are back at the non-state restoration problem. Therefore, it remains to check that the challenge is an accepting challenge for $\mathbb{x}$. If so, we have found a new branch for the a-tree of accepting transcripts for $\mathbb{x}$. Challenges that are both consistent and accepting are said to be *good*. This is reflected formally in the following definition.

Definition 5.4. *A challenge $\rho \in \mathcal{R}$ is* **good for** $(\mathsf{rnd}, \mathbb{x}, \alpha_1, \sigma)$ *if there exists some $\alpha_2 \in \{0,1\}^{\mathsf{pv}_2}$ such that*

- $|\mathbb{x}| \leq n$*;*
- $\mathbf{V}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho, \alpha_2) = 1$*; and*
- *The output of* $\Gamma^{\mathsf{sr}}(s, \mathsf{rnd}|_{((\mathbb{x},\alpha_1,\sigma),\rho)}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$ *is* $(\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2)$.

Moreover, we denote by $Y^{\mathsf{sr}}(\mathsf{rnd}, \mathbb{x}, \alpha_1, \sigma)$ *the number of good challenges for* $(\mathsf{rnd}, \mathbb{x}, \alpha_1, \sigma)$ *within* $\mathcal{R}$.

Remark 5.5. For this extractor to succeed in returning a witness for $\mathbb{x}$, we need two conditions to hold. First, we want the early abort test (Line 2) to pass. Secondly, we need the number of challenges such that the Line 5f's condition holds to be larger or equal to the arity a. On the contrary, knowing that the early abort test passes, the only way for the extractor to not return a witness is that there are not enough challenges that satisfy Line 5f's condition.

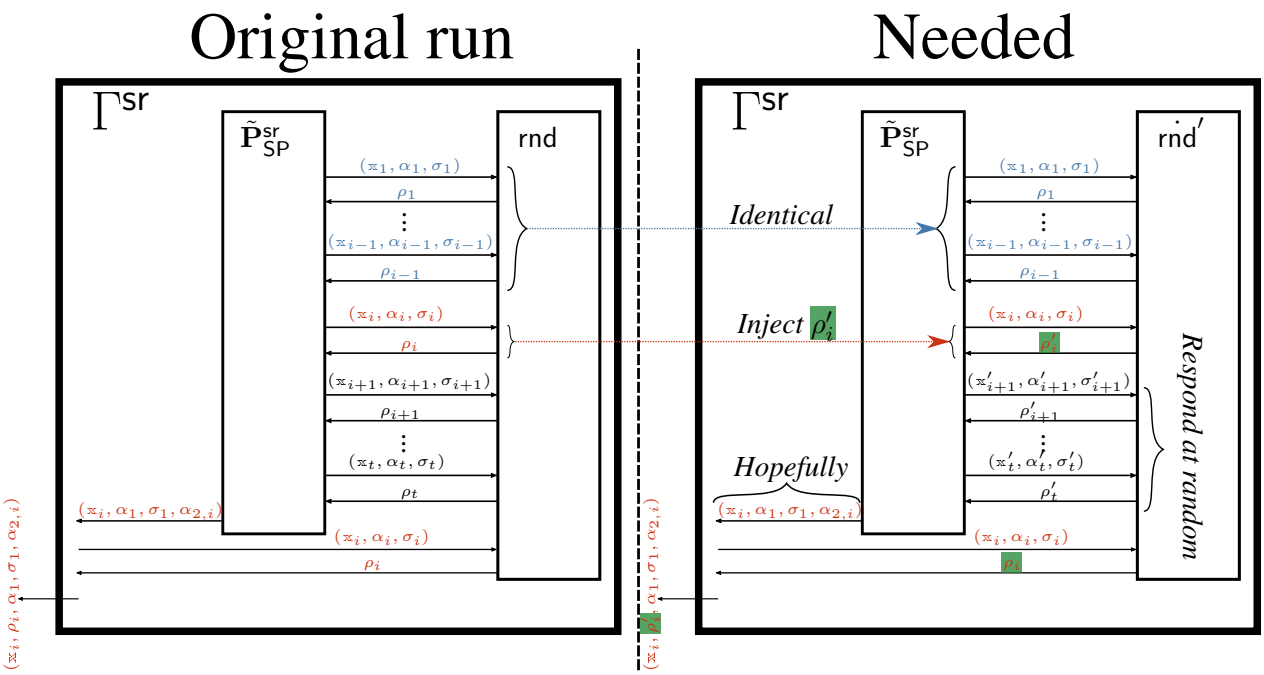

Figure 7: Comparison of the random oracle used in the original run and the one implemented at each iteration of the extractor.

5.2 Knowledge soundness error

We can characterize the extractor's success as follows.

Lemma 5.6. *Fix the instance* $\mathbb{x}$*, the prover's messages* (α_1, α_2)*, the challenge* ρ*, the state restoration game's trace* $\mathsf{tr}^{\mathsf{sr}}$ *and the deterministic malicious prover* $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}$ *arbitrarily. Then if* $|\mathbb{x}| \leq n$*,*

$$\mathbf{E}^{\mathsf{SRKS}}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2, \mathsf{tr}^{\mathsf{sr}}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$$

finds a witness for $\mathbb{x}$ *if and only if* $\mathbf{V}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho, \alpha_2) = 1$ *and* $Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}, \alpha_1, \sigma) \geq a$*, where* $\mathbf{E}^{\mathsf{SRKS}}_{\mathsf{SP}}$ *is defined as in Construction 5.2 and* $\dot{\mathsf{rnd}}$ *is the random oracle picked at Line 4 of the extractor.*

Proof. If the extractor can retrieve a witness for $\mathbb{x}$ it implies that it has passed the early abort, or in other words, $\mathbf{V}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho, \alpha_2) = 1$. Moreover, in that case, it must have found enough challenges ρ' such that

- $\mathbf{V}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho, \alpha_2) = 1$; and
- The output of $\Gamma^{\mathsf{sr}}(s, \dot{\mathsf{rnd}}|_{((\mathbb{x},\alpha_1,\sigma),\rho)}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$ is $(\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2)$ for some α_2.

In other words, we must have $Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}, \alpha_1, \sigma) \geq a$. The other direction follows directly from Remark 5.5. □

As we have pointed out, the value of $Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}, \alpha_1, \sigma)$ is important in the success probability's analysis of the extractor. Hence, we seek to find bounds related to this random variable. We start with the following lemma, which will be useful later.

Lemma 5.7. *Let* $T \coloneqq \{0,1\}^{\leq n} \times \{0,1\}^{\mathsf{pv}_1} \times \{0,1\}^s$*, then*

$$\sum_{(\mathbb{x}',\alpha_1',\sigma')\in T} \Pr\left[\begin{array}{l|ccc} Y^{\mathsf{sr}}(\dot{\mathsf{rnd}},\mathbb{x}',\alpha_1',\sigma') > 0 & \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ \textit{conditioned on} & (\mathbb{x},\alpha_1,\sigma,\rho,\alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathsf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ (\mathbb{x}',\alpha_1',\sigma') = (\mathbb{x},\alpha_1,\sigma) & \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array}\right] \leq t+1 \ .$$

Proof. We consider two distinct cases: the malicious prover $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}$ has queried $(\mathbb{x},\alpha_1,\sigma)$ in the game $\Gamma^{\mathsf{sr}}(s,\mathsf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$ or not. More formally, we can write the two cases as:

- $((\mathbb{x},\alpha_1,\sigma),\rho) \in \mathsf{tr}^{\mathsf{sr}}$; and
- $((\mathbb{x},\alpha_1,\sigma),\rho) \notin \mathsf{tr}^{\mathsf{sr}}$.

Moreover, we subdivide the former on the index $i \in [t]$ such that $\mathsf{tr}^{\mathsf{sr}}_i = ((\mathbb{x},\alpha_1,\sigma),\rho)$. Note that the index i is uniquely determined since we assume that queries are unique. Assume without loss of generality that the queries are well-formed, that is, there are of the form $(\mathbb{x},\alpha_1,\sigma)$ for some $\mathbb{x} \in \{0,1\}^{\leq n}$, $\alpha \in \{0,1\}^{\mathsf{pv}_1}$ and $\sigma \in \{0,1\}^s$. Then,

$$\begin{aligned}
&\sum_{(\mathbb{x}',\alpha_1',\sigma')\in T} \Pr\left[\begin{array}{l|ccc} Y^{\mathsf{sr}}(\dot{\mathsf{rnd}},\mathbb{x}',\alpha_1',\sigma') > 0 & \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ \text{conditioned on} & (\mathbb{x},\alpha_1,\sigma,\rho,\alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathsf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ (\mathbb{x}',\alpha_1',\sigma') = (\mathbb{x},\alpha_1,\sigma) & \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array}\right] \\
&= \sum_{(\mathbb{x}',\alpha_1',\sigma')\in T} \left(\sum_{i=1}^{t} \Pr\left[\begin{array}{l|ccc} Y^{\mathsf{sr}}(\dot{\mathsf{rnd}},\mathbb{x}',\alpha_1',\sigma') > 0 & \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ \text{conditioned on} & (\mathbb{x},\alpha_1,\sigma,\rho,\alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathsf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ (\mathbb{x}',\alpha_1',\sigma') = (\mathbb{x},\alpha_1,\sigma) & \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \\ \wedge\, \mathsf{tr}^{\mathsf{sr}}_i = ((\mathbb{x}',\alpha_1',\sigma'),\rho) \end{array}\right] \right. \\
&\quad \cdot \Pr\left[\begin{array}{l|ccc} \mathsf{tr}^{\mathsf{sr}}_i = ((\mathbb{x}',\alpha_1',\sigma'),\rho) & \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ \text{conditioned on} & (\mathbb{x},\alpha_1,\sigma,\rho,\alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathsf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ (\mathbb{x}',\alpha_1',\sigma') = (\mathbb{x},\alpha_1,\sigma) & & & \end{array}\right] \\
&\quad + \Pr\left[\begin{array}{l|ccc} Y^{\mathsf{sr}}(\dot{\mathsf{rnd}},\mathbb{x}',\alpha_1',\sigma') > 0 & \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ \text{conditioned on} & (\mathbb{x},\alpha_1,\sigma,\rho,\alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathsf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ (\mathbb{x}',\alpha_1',\sigma') = (\mathbb{x},\alpha_1,\sigma) & \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \\ \wedge\, ((\mathbb{x}',\alpha_1',\sigma'),\rho) \notin \mathsf{tr}^{\mathsf{sr}}_i \end{array}\right] \\
&\quad \left. \cdot \Pr\left[\begin{array}{l|ccc} ((\mathbb{x}',\alpha_1',\sigma'),\rho) \notin \mathsf{tr}^{\mathsf{sr}}_i & \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ \text{conditioned on} & (\mathbb{x},\alpha_1,\sigma,\rho,\alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathsf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ (\mathbb{x}',\alpha_1',\sigma') = (\mathbb{x},\alpha_1,\sigma) & & & \end{array}\right] \right) \\
&\leq \sum_{i=1}^{t} \underbrace{\sum_{(\mathbb{x}',\alpha_1',\sigma')\in T} \Pr\left[\begin{array}{l|ccc} \mathsf{tr}^{\mathsf{sr}}_i = ((\mathbb{x}',\alpha_1',\sigma'),\rho) & \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ \text{conditioned on} & (\mathbb{x},\alpha_1,\sigma,\rho,\alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathsf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ (\mathbb{x}',\alpha_1',\sigma') = (\mathbb{x},\alpha_1,\sigma) & & & \end{array}\right]}_{=1} \\
&\quad + \underbrace{\sum_{(\mathbb{x}',\alpha_1',\sigma')\in T} \Pr\left[\begin{array}{l|ccc} ((\mathbb{x}',\alpha_1',\sigma'),\rho) \notin \mathsf{tr}^{\mathsf{sr}}_i & \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ \text{conditioned on} & (\mathbb{x},\alpha_1,\sigma,\rho,\alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathsf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ (\mathbb{x}',\alpha_1',\sigma') = (\mathbb{x},\alpha_1,\sigma) & & & \end{array}\right]}_{=1} \\
&= t+1 \ .
\end{aligned}$$

□

Lemma 4.7 has a straightforward generalization to the state restoration setup.

Lemma 5.8. *For every $\ell \in [2^{\mathsf{r}}]$, instance $\mathbb{x}$, first prover message α_1 and salt σ,*

$$\Pr\left[\begin{array}{l} \rho \textit{ is good for } (\dot{\mathsf{rnd}}, \mathbb{x}, \alpha_1, \sigma) \\ \textit{conditioned on} \\ Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}, \alpha_1, \sigma) = \ell \end{array} \middle| \begin{array}{ccc} \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array}\right] = \frac{\ell}{2^{\mathsf{r}}} \ .$$

Proof. For every random oracle $\mathsf{rnd} \in \mathcal{U}(\mathsf{r})$, the value of $Y^{\mathsf{sr}}(\mathsf{rnd}, \mathbb{x}, \alpha_1, \sigma)$ is independent from $\rho := \mathsf{rnd}(\mathbb{x}, \alpha_1, \sigma)$ by definition of the former. Back to the experiment, we have $\rho = \mathsf{rnd}(\mathbb{x}, \alpha_1, \sigma)$ and since $((\mathbb{x}, \alpha_1, \sigma), \rho) \in \mathsf{tr}^{\mathsf{sr}}$ and both rnd and $\dot{\mathsf{rnd}}$ agree on $\mathsf{tr}^{\mathsf{sr}}$, it follows that $\rho = \dot{\mathsf{rnd}}(\mathbb{x}, \alpha_1, \sigma)$. Therefore ρ and $Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}, \alpha_1, \sigma)$ are independent. Finally, the challenge ρ is picked uniformly at random within a set of cardinality 2^{r}, and there are exactly ℓ challenges which are good for $(\mathsf{rnd}, \mathbb{x}, \alpha_1, \sigma)$ by assumption. □

We now prove that the extractor has the desired probability of success. By Lemma 5.6,

$$\begin{aligned}
&\Pr\left[\begin{array}{l} |\mathbb{x}| \le n \\ \wedge\, (\mathbb{x}, \mathbb{w}) \notin R \\ \wedge\, b = 1 \end{array} \middle| \begin{array}{ccc} \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ b & \leftarrow & \mathbf{V}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho, \alpha_2) \\ \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SRKS}}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2, \mathsf{tr}^{\mathsf{sr}}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \end{array}\right] \\
&= \Pr\left[\begin{array}{l} \rho \text{ is good for } (\dot{\mathsf{rnd}}, \mathbb{x}, \alpha_1, \sigma) \\ \wedge\, Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}, \alpha_1, \sigma) < a \end{array} \middle| \begin{array}{ccc} \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array}\right] \\
&\le \sum_{(\mathbb{x}', \alpha_1', \sigma') \in T} \Pr\left[\begin{array}{l} \rho \text{ is good for } (\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') \\ \wedge\, Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') < a \\ \text{conditioned on} \\ (\mathbb{x}', \alpha_1', \sigma') = (\mathbb{x}, \alpha_1, \sigma) \end{array} \middle| \begin{array}{ccc} \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array}\right] \\
&= \sum_{\ell=1}^{a-1} \sum_{(\mathbb{x}', \alpha_1', \sigma') \in T} \underbrace{\Pr\left[\begin{array}{l} \rho \text{ is good for } (\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') \\ \text{conditioned on} \\ (\mathbb{x}', \alpha_1', \sigma') = (\mathbb{x}, \alpha_1, \sigma) \\ \wedge\, Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') = \ell \end{array} \middle| \begin{array}{ccc} \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array}\right]}_{= \frac{\ell}{2^{\mathsf{r}}} \text{ by Lemma 5.8}} \\
&\quad \cdot \Pr\left[\begin{array}{l} Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') = \ell \\ \text{conditioned on} \\ (\mathbb{x}', \alpha_1', \sigma') = (\mathbb{x}, \alpha_1, \sigma) \end{array} \middle| \begin{array}{ccc} \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array}\right] \\
&\le \frac{a-1}{2^{\mathsf{r}}} \sum_{(\mathbb{x}', \alpha_1', \sigma') \in T} \sum_{\ell=1}^{a-1} \Pr\left[\begin{array}{l} Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') = \ell \\ \text{conditioned on} \\ (\mathbb{x}', \alpha_1', \sigma') = (\mathbb{x}, \alpha_1, \sigma) \end{array} \middle| \begin{array}{ccc} \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array}\right] \\
&\le \frac{a-1}{2^{\mathsf{r}}} \sum_{(\mathbb{x}', \alpha_1', \sigma') \in T} \Pr\left[\begin{array}{l} Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') > 0 \\ \text{conditioned on} \\ (\mathbb{x}', \alpha_1', \sigma') = (\mathbb{x}, \alpha_1, \sigma) \end{array} \middle| \begin{array}{ccc} \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array}\right] \\
&\le (t+1)\frac{a-1}{2^{\mathsf{r}}} \ ,
\end{aligned}$$

where the last inequality follows from Lemma 5.7.

5.3 Extraction time

The extraction time is directly related to the number of invocations done by the extractor to the malicious state restoration prover. Therefore, we formally capture the number of invocations the extractor does in a random variable.

Definition 5.9. *For every random oracle* rnd *and extractor's randomness* ν*, let* $I(\mathsf{rnd}, \nu)$ *be the number of invocations done by*

$$\mathbf{E}_{\mathsf{SP}}^{\mathsf{SRKS}}(\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2, \mathsf{tr}^{\mathsf{sr}}, \tilde{\mathbf{P}}_{\mathsf{SP}}^{\mathsf{sr}}; \nu)$$

to $\tilde{\mathbf{P}}_{\mathsf{SP}}^{\mathsf{sr}}$*, where* $(\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}_{\mathsf{SP}}^{\mathsf{sr}})$*. Running the state restoration game is considered as one invocation.*

We are interested in proving the following lemma.

Lemma 5.10. *It holds that*

$$\mathbb{E}\left[I(\mathsf{rnd}, \nu) \;\middle|\; \begin{array}{ccc} \nu & \leftarrow & \mathcal{N} \\ \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \end{array} \right] \leq (t+1) \cdot (a-1) \ .$$

Proof. We have

$$\begin{aligned}
&\mathbb{E}\left[I(\mathsf{rnd}, \nu) \;\middle|\; \begin{array}{ccc} \nu & \leftarrow & \mathcal{N} \\ \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \end{array} \right] \\
&\leq \sum_{(\mathbb{x}', \alpha_1', \sigma') \in T} \mathbb{E}\left[\begin{array}{l} I(\mathsf{rnd}, \nu) \\ \text{conditioned on} \\ (\mathbb{x}', \alpha_1', \sigma') = (\mathbb{x}, \alpha_1, \sigma) \end{array} \;\middle|\; \begin{array}{ccc} \nu & \leftarrow & \mathcal{N} \\ \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}_{\mathsf{SP}}^{\mathsf{sr}}) \\ \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array} \right] \\
&= \sum_{\ell=1}^{2^{\mathsf{r}}} \sum_{(\mathbb{x}', \alpha_1', \sigma') \in T} \mathbb{E}\left[\begin{array}{l} I(\mathsf{rnd}, \nu) \\ \text{conditioned on} \\ (\mathbb{x}', \alpha_1', \sigma') = (\mathbb{x}, \alpha_1, \sigma) \\ \wedge\, Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') = \ell \end{array} \;\middle|\; \begin{array}{ccc} \nu & \leftarrow & \mathcal{N} \\ \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}_{\mathsf{SP}}^{\mathsf{sr}}) \\ \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array} \right] \\
&\quad \cdot \Pr\left[\begin{array}{l} Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') = \ell \\ \text{conditioned on} \\ (\mathbb{x}', \alpha_1', \sigma') = (\mathbb{x}, \alpha_1, \sigma) \end{array} \;\middle|\; \begin{array}{ccc} \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}_{\mathsf{SP}}^{\mathsf{sr}}) \\ \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array} \right] \\
&= \sum_{\ell=1}^{2^{\mathsf{r}}} \sum_{(\mathbb{x}', \alpha_1', \sigma') \in T} \mathbb{E}\left[\begin{array}{l} I(\mathsf{rnd}, \nu) \\ \text{conditioned on} \\ (\mathbb{x}', \alpha_1', \sigma') = (\mathbb{x}, \alpha_1, \sigma) \\ \wedge\, Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') = \ell \\ \rho \text{ is good for } (\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') \end{array} \;\middle|\; \begin{array}{ccc} \nu & \leftarrow & \mathcal{N} \\ \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}_{\mathsf{SP}}^{\mathsf{sr}}) \\ \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array} \right] \\
&\quad \cdot \Pr\left[\begin{array}{l} Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') = \ell \\ \text{conditioned on} \\ (\mathbb{x}', \alpha_1', \sigma') = (\mathbb{x}, \alpha_1, \sigma) \end{array} \;\middle|\; \begin{array}{ccc} \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}_{\mathsf{SP}}^{\mathsf{sr}}) \\ \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array} \right] \\
&\quad \cdot \Pr\left[\begin{array}{l} \rho \text{ is good for } (\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') \\ \text{conditioned on} \\ (\mathbb{x}', \alpha_1', \sigma') = (\mathbb{x}, \alpha_1, \sigma) \end{array} \;\middle|\; \begin{array}{ccc} \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}_{\mathsf{SP}}^{\mathsf{sr}}) \\ \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array} \right] \ .
\end{aligned}$$

The first equality comes from the fact that the number of invocation is always zero if ρ is not good for $(\dot{\mathsf{rnd}}, \mathbb{x}, \alpha_1, \sigma)$. Indeed, in that case, we have

$$0 = \mathbf{V}_{\mathsf{SP}}(\mathbb{x}, \alpha_1, \rho, \alpha_2) \ \vee \ |\mathbb{x}| > n \ \vee \ (\mathbb{x}, \alpha_1, \sigma, \rho, \cdot) = \Gamma^{\mathsf{sr}}(s, \dot{\mathsf{rnd}}|_{((\mathbb{x},\alpha_1,\sigma),\rho)}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$$

and since

$$\Gamma^{\mathsf{sr}}(s, \dot{\mathsf{rnd}}|_{((\mathbb{x},\alpha_1,\sigma),\rho)}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) = \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \ ,$$

it must be that one of the first two conditions is not respected. In either case, the early abort test (Line 2 of the extractor) will not pass; therefore, the number of queries would be zero.

Let

$$e_{\ell,\mathbb{x}',\alpha_1',\sigma'} \coloneqq \mathbb{E}\left[\begin{array}{l} I(\mathsf{rnd}, \nu) \\ \text{conditioned on} \\ \wedge\, (\mathbb{x}', \alpha_1', \sigma') = (\mathbb{x}, \alpha, \sigma) \\ \wedge\, Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') = \ell \\ \rho \text{ is good for } (\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') \end{array} \middle| \begin{array}{ccc} \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array} \right] .$$

We show that for every choice of $\ell > 0$, $\mathbb{x}'$, α_1' and σ' we have $e_{\ell,\mathbb{x}',\alpha_1',\sigma'} \leq (a-1) \cdot \frac{2^{\mathsf{r}}}{\ell}$. First notice that since ρ is good for $(\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma')$ the extractor will not early abort. If $\ell < a$, the number of invocations will be $2^{\mathsf{r}} - 1 \leq (a-1)\frac{2^{\mathsf{r}}}{\ell}$ since the extractor will pick every possible challenge before realizing that there are not enough good challenges. On the other hand, if $\ell \geq a$, the number of iterations in the loop can be modelized, once again, by the NHG distribution. Hence in that case we have $e_{\ell,\mathbb{x}',\alpha_1',\sigma'} = \mu_{2^{\mathsf{r}}-1,\ell-1,a-1} = (a-1)\frac{2^{\mathsf{r}}}{\ell}$ where $\mu_{2^{\mathsf{r}}-1,\ell-1,a-1}$ is the mean of the NHG, which we have presented in Definition 2.14. Leveraging this upper bound, Lemma 5.7 and Lemma 5.8 we obtain

$$\begin{aligned}
&\mathbb{E}\left[I(\mathsf{rnd}, \nu) \,\middle|\, \begin{array}{ccc} \nu & \leftarrow & \mathcal{N} \\ \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \end{array} \right] \\
&= (a-1) \cdot \sum_{(\mathbb{x}',\alpha_1',\sigma') \in T} \Pr\left[\begin{array}{l} Y^{\mathsf{sr}}(\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') > 0 \\ \text{conditioned on} \\ \wedge\, (\mathbb{x}', \alpha_1', \sigma') = (\mathbb{x}, \alpha, \sigma) \\ \wedge\, \rho \text{ is good for } (\dot{\mathsf{rnd}}, \mathbb{x}', \alpha_1', \sigma') \end{array} \middle| \begin{array}{ccc} \mathsf{rnd} & \leftarrow & \mathcal{U}(\mathsf{r}) \\ (\mathbb{x}, \alpha_1, \sigma, \rho, \alpha_2) & \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \\ \dot{\mathsf{rnd}} & \leftarrow & \mathcal{U}(\mathsf{r})|_{\mathsf{tr}^{\mathsf{sr}}} \end{array} \right] \\
&\leq (a-1) \cdot (t+1)
\end{aligned}$$

□

Each call to $\Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$ consists of one invocation to $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}$, which runs in expected time $\tau_{\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}}$ by definition, one query to the random oracle rnd, which runs in constant time, and some operations that are efficient in terms of $|\mathbb{x}|$. Hence $\Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}})$'s running time is $\tau_{\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}} + \mathsf{poly}(|\mathbb{x}|)$. By the previous lemma, the expected number of invocations is $(a-1)\cdot(t+1)$. Finally, in case of success, we run $\mathbf{E}^{\mathsf{SS}}_{\mathsf{SP}}$, which is efficient. Overall, the extractor's extraction time is upper bounded by

$$(a-1)(t+1) \cdot (\tau_{\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}} + \mathsf{poly}(n)) \ ,$$

as desired.

6 State-restoration knowledge soundness from special soundness for IPs

In the general case of interactive proofs, the statement is the following.

Theorem 6.1. *Let* $\mathsf{IP} = (\mathbf{P}_{\mathsf{IP}}, \mathbf{V}_{\mathsf{IP}})$ *be a public-coin interactive proof for a relation* R *with* $(a_1, \ldots, a_{\mathsf{k}})$*-special soundness and verifier randomness complexity* $(\mathsf{r}_1, \ldots, \mathsf{r}_{\mathsf{k}})$*. It follows that* IP *has rewinding state restoration knowledge soundness error*

$$\kappa^{\mathsf{sr}}_{\mathsf{IP}}(\mathbb{x}, t) \leq (t+1)\left(1 - \prod_{i=1}^{\mathsf{k}}\left(1 - \frac{a_i - 1}{2^{\mathsf{r}_i}}\right)\right) \ ,$$

and extraction time

$$\mathsf{et}^{\mathsf{sr}}_{\mathsf{IP}}(n, t, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \leq \left(\left(\prod_{i=1}^{\mathsf{k}} a_i\right) + (t+1) \cdot \left(\left(\prod_{i=1}^{\mathsf{k}} a_i\right) - 1\right)\right) \cdot (\tau_{\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}} + \mathsf{poly}(n)) \ .$$

Moreover, the number of times the knowledge soundness extractor invokes $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}$ *is* $\left(\left(\prod_{i=1}^{\mathsf{k}} a_i\right) - 1\right)(t+1)$.

Let IP be an arbitrary interactive proof for a relation R with $(a_1, \ldots, a_{\mathsf{k}})$-special soundness. Let $(\mathsf{r}_1, \ldots, \mathsf{r}_{\mathsf{k}})$ be its randomness complexity and $N_i := 2^{\mathsf{r}_i}$ for every $i \in [\mathsf{r}]$. Finally, let $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}$ be any deterministic state restoration malicious prover, also a t-query oracle algorithm.

6.1 Construction of the extractor

Let $\mathbf{E}^{\mathsf{SS}}_{\mathsf{IP}}$ be IP's special sound extractor. We construct a state restoration knowledge sound extractor $\mathbf{E}^{\mathsf{SRKS}}_{\mathsf{IP}}$ that satisfies Theorem 6.1's bounds. As in the SP case, the extractor is first tasked to retrieve a $(a_1, \ldots, a_{\mathsf{k}})$-tree of accepting transcript for an instance $\mathbb{x}$ and then call $\mathbf{E}^{\mathsf{SS}}_{\mathsf{IP}}$ to retrieve a witness for $\mathbb{x}$. To do so, we take inspiration from Construction 4.2 and define a helper algorithm $\mathsf{TreeFinder}_0$ which is tasked by our extractor to find a tree of accepting transcripts for $\mathbb{x}$.

Construction 6.2. The rewinding state restoration knowledge soundness extractor is defined as follows.

$\mathbf{E}^{\mathsf{SRKS}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{tr}^{\mathsf{sr}}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}})$:
1. If $|\mathbb{x}| > n$ return $\perp$.
2. Pick random oracles that match the traces: $\dot{\mathsf{rnd}}_i \leftarrow \mathcal{U}(\mathsf{r}_i)|_{\mathsf{tr}^{\mathsf{sr}}_i}$ for every $i \in [\mathsf{k}]$.
3. $T \leftarrow \mathsf{TreeFinder}_0(\mathbb{x}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, (\dot{\mathsf{rnd}}_i)_{i\in[\mathsf{k}]})$.
4. If $T = \perp$ return $\perp$ otherwise return $\mathbf{E}^{\mathsf{SS}}_{\mathsf{IP}}(\mathbb{x}, T)$.

The random oracles in $(\dot{\mathsf{rnd}}_i)_{i\in[\mathsf{k}]}$ are represented using lazy sampling, which is why we use the dot notation here. The rewinding state restoration knowledge soundness extractor's helper algorithm is defined as follows.

Construction 6.3. Given $\mathbb{x}$, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{\mathsf{k}+1})$, $\boldsymbol{\alpha} = (\sigma_1, \ldots, \sigma_{\mathsf{k}+1})$, $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_{\mathsf{k}})$ and $\dot{\mathbf{rnd}} \in \mathcal{U}\big((\mathsf{r}_j)_{j\in[\mathsf{k}]}\big)$, the helper algorithm is

$\mathsf{TreeFinder}_i(\mathbb{x}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \dot{\mathbf{rnd}})$:
1. If we are treating the base case ($i = \mathsf{k}$):
   (a) Get the success bit $b \leftarrow \mathbf{V}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\rho})$.

(b) If $b = 1$ return 1 otherwise return $\perp$.

2. Early abort recursive call $T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \dot{\mathbf{rnd}})$.
3. If $T = \perp$ return $\perp$.
4. Add the challenge to a set of picked challenges $S \coloneqq \{\rho_{i+1}\}$, store the good challenge in a list $E \coloneqq (\rho_{i+1})$ and its answer $N \coloneqq (\alpha_{i+2})$.
5. While it remains unpicked challenges $|\mathcal{R} \setminus S| > 0$ and we have not found enough good challenges $|E| \neq a$:
   (a) Get a new challenge unpicked $\rho'_{i+1} \leftarrow \mathcal{R}_{i+1} \setminus S$.
   (b) Add the new challenge to a set of picked challenges $S \coloneqq S \cup \{\rho'_{i+1}\}$.
   (c) Adapt the random oracle: $\dot{\mathsf{rnd}}'_{i+1} \leftarrow \dot{\mathsf{rnd}}_{i+1}|_{((\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}), \rho'_{i+1})}$.
   (d) Set the new random oracles list: $\dot{\mathbf{rnd}}' \coloneqq \dot{\mathbf{rnd}}_{[:i+1]} \,\|\, \dot{\mathsf{rnd}}'_{i+1} \,\|\, \dot{\mathbf{rnd}}_{[i+2:]}$.
   (e) Execute the state restoration game: $(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \boldsymbol{\rho}') \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \dot{\mathbf{rnd}}', \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}})$.
   (f) Test whereas the partial transcripts are consistent: If $\boldsymbol{\alpha}'_{[:i+2]} \neq \boldsymbol{\alpha}_{[:i+2]} \vee \boldsymbol{\sigma}'_{[:i+2]} \neq \boldsymbol{\sigma}_{[:i+2]} \vee \boldsymbol{\rho}'_{[:i+2]} \neq \boldsymbol{\rho}_{[:i+1]} \,\|\, \rho'_{i+1} \vee \mathbb{x}' \neq \mathbb{x}$ go to Line 5.
   (g) Recursive call $T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}', \boldsymbol{\sigma}', \boldsymbol{\rho}', \dot{\mathbf{rnd}}')$.
   (h) If $T \neq \perp$, add the good challenge $E \coloneqq E \,\|\, \rho'_{i+1}$ and its response $N \coloneqq N \,\|\, \alpha'_{i+2}$.
6. If $|E| = a_{i+1}$, create a tree with α'_{i+1} as root and children N (verifier responses) connected by edges E (prover challenges) and return it, otherwise return $\perp$.

This algorithm is similar to Construction 4.4 but instead of invocating the malicious prover $\tilde{\mathbf{P}}_{\mathsf{IP}}$, which it does not have, to retrieve the next prover's message, it runs the whole state restoration game by tweaking the random oracle $\dot{\mathsf{rnd}}_{i+1}$ (Line 5c) in hope that the value $(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \boldsymbol{\rho}')$ returned by the state restoration game is such that

$$\boldsymbol{\alpha}'_{[:i+2]} = \boldsymbol{\alpha}_{[:i+2]} \wedge \boldsymbol{\sigma}'_{[:i+2]} = \boldsymbol{\sigma}_{[:i+2]} \wedge \boldsymbol{\rho}'_{[:i+2]} = \boldsymbol{\rho}_{[:i+1]} \,\|\, \rho'_{i+1} \wedge \mathbb{x}' = \mathbb{x} \ ,$$

where ρ'_{i+1} is the challenge picked by the algorithm. Analogously to Construction 5.2, we say these tests are the *consistency tests*. If the consistency tests are satisfied, then the state restoration game has returned a complete transcript $(\boldsymbol{\alpha}', \boldsymbol{\rho}')$ that matches the partial transcript $(\boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\rho}_{[:i+2]})$. The extractor can use this transcript to create a new path in the tree from the root node to a leaf node. In other words, instead of searching for edges, the algorithm directly searches for complete paths from the current location to a leaf.

To simplify the proof, we first forget about lazy sampling. Furthermore, we use the random oracles ($\mathbf{rnd}$) instead of lazy random oracles ($\dot{\mathbf{rnd}}$). Doing so will simplify our analysis. Eventually, we will show that, in our case, replacing a lazy random oracle with a classical random oracle does not change the probability expressions. Moreover, this change makes the notation lighter.

The proof of this statement is more involved than the other ones. Especially, we need a more fine-grained understanding of the randomness of

$$\mathsf{TreeFinder}_i(\mathbb{x}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu) \ .$$

This need motivates the following notation.

Definition 6.4. *For every $i \in \{0, \ldots, \mathsf{k}\}$, we denote by ν_i the randomness of*

$$\mathsf{TreeFinder}_i(\mathbb{x}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu_i) \ .$$

We say that the random space of this instantiation is $\mathcal{N}_i$, that is, $\nu_i \in \mathcal{N}_i$. Moreover, we denote by $\nu_{i+1}^{\rho'_{i+1}}(\nu_i)$ the randomness of its recursive call that results from using ρ'_{i+1} in the state restoration game, whether it was in the loop or for the early abort test. Finally, in the latter case, we write $\nu_{i+1}^{\mathsf{ea}}(\nu_i) \coloneqq \nu_{i+1}^{\rho_{i+1}}(\nu_i)$ to highlight that this is the randomness of the early abort recursive call.

During any call $\mathsf{TreeFinder}_i$, we would like to characterize the challenges ρ'_{i+1} that would pass the consistency tests and lead to a successful recursive call. As we have seen previously, we capture these two properties of a challenge in the two distinct properties combined to form the so-called *good* challenges. The challenges that pass the consistency tests are said to be *consistent*, while challenges that result in a successful recursive call are known as *successful* challenges.

Definition 6.5. *For every $i \in \{0, \ldots, \mathsf{k}-1\}$, a challenge $\rho_{i+1} \in \mathcal{R}_{i+1}$ is said to be* **consistent with**

$$(\mathbb{x}, \underbrace{(\alpha_1, \ldots, \alpha_{\mathsf{k}+1})}_{\coloneqq \boldsymbol{\alpha}}, \underbrace{(\sigma_1, \ldots, \sigma_{\mathsf{k}+1})}_{\coloneqq \boldsymbol{\sigma}}, \underbrace{(\mathsf{rnd}_1, \ldots, \mathsf{rnd}_{\mathsf{k}})}_{\coloneqq \mathbf{rnd}})$$

if

- $|\mathbb{x}| \leq n$*; and*
- *The output of*

$$\Gamma^{\mathsf{sr}}(s, (\mathsf{rnd}_1, \ldots, \mathsf{rnd}_i, \mathsf{rnd}_{i+1}|_{((\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}), \rho_{i+1})}, \mathsf{rnd}_{i+2}, \ldots, \mathsf{rnd}_{\mathsf{k}}), \tilde{\mathbf{P}}_{\mathsf{SP}}^{\mathsf{sr}})$$

is

$$(\mathbb{x}, (\alpha_1 \ldots, \alpha_{i+1}, \cdot, \ldots, \cdot), (\sigma_1, \ldots, \sigma_{i+1}, \cdot, \ldots, \cdot), (\rho_1, \ldots, \rho_{i+1}, \cdot, \ldots, \cdot)) \ ,$$

where $\rho_j \coloneqq \mathsf{rnd}_j(\mathbb{x}, (\alpha_1, \ldots, \alpha_j), (\sigma_1, \ldots, \sigma_j))$ for every $j \in [i]$.

Successful challenges are formally defined as follows.

Definition 6.6. *For every $i \in \{0, \ldots, \mathsf{k}-1\}$, a challenge $\rho_{i+1} \in \mathcal{R}_{i+1}$ is said to be* **successful for**

$$(\mathbb{x}, \nu_{i+1}, \underbrace{(\alpha_1, \ldots, \alpha_{\mathsf{k}+1})}_{\coloneqq \boldsymbol{\alpha}}, \underbrace{(\sigma_1, \ldots, \sigma_{\mathsf{k}+1})}_{\coloneqq \boldsymbol{\sigma}}, \underbrace{(\mathsf{rnd}_1, \ldots, \mathsf{rnd}_{\mathsf{k}})}_{\coloneqq \mathbf{rnd}})$$

if

- $|\mathbb{x}| \leq n$*; and*
- *The output of*

$$\mathsf{TreeFinder}_{i+1}(\mathbb{x}, \tilde{\mathbf{P}}_{\mathsf{IP}}^{\mathsf{sr}}, \boldsymbol{\alpha}', \boldsymbol{\sigma}', \boldsymbol{\rho}', \mathbf{rnd}; \nu_{i+1})$$

is not $\perp$, where

$$(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \boldsymbol{\rho}') \leftarrow \Gamma^{\mathsf{sr}}(s, (\mathsf{rnd}_1, \ldots, \mathsf{rnd}_i, \mathsf{rnd}_{i+1}|_{((\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}), \rho_{i+1})}, \mathsf{rnd}_{i+2}, \ldots, \mathsf{rnd}_{\mathsf{k}}), \tilde{\mathbf{P}}_{\mathsf{SP}}^{\mathsf{sr}}) \ .$$

In other words, the output is not an $(a_{i+1}, \ldots, a_{\mathsf{k}})$-tree of accepting transcripts for $\mathbb{x}$ with prefix

$$(\mathsf{rnd}_j(\mathbb{x}, (\alpha_1, \ldots, \alpha_j), (\sigma_1, \ldots, \sigma_j)))_{j \in [i]} \ .$$

When a challenge is consistent and successful, we say it is a *good challenge*. Good challenges are the ones that the extractor could use to construct the trees of accepting transcripts for $\mathbb{x}$ with the corresponding prefix. This definition is similar to how good challenges are defined in Section 4.

Definition 6.7. *For every $i \in \{0, \ldots, \mathsf{k}-1\}$, a challenge $\rho_{i+1} \in \mathcal{R}_{i+1}$ is said to be* **good for**

$$(\mathbb{x}, \nu_i, \underbrace{(\alpha_1, \ldots, \alpha_{\mathsf{k}+1})}_{:=\boldsymbol{\alpha}}, \underbrace{(\sigma_1, \ldots, \sigma_{\mathsf{k}+1})}_{:=\boldsymbol{\sigma}}, \underbrace{(\mathsf{rnd}_1, \ldots, \mathsf{rnd}_{\mathsf{k}})}_{:=\mathbf{rnd}})$$

if is consistent $(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \mathbf{rnd})$ and successful for $(\mathbb{x}, \nu_{i+1}^{\rho_{i+1}}(\nu_i), \boldsymbol{\alpha}, \boldsymbol{\sigma}, \mathbf{rnd})$. Moreover, we denote by

$$Y_{i+1}^{\mathsf{sr}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \mathbf{rnd}, \nu_i)$$

the number of good challenges for $(\mathbb{x}, \nu_i, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \mathbf{rnd})$ within $\mathcal{R}_{i+1}$.

Figs. 8 to 10 illustrate the steps of the tree's construction. For the sake of simplicity, we change the way the algorithm works. Instead of working recursively, the current node, say at level i, tries to find a_i accepting transcripts for $\mathbb{x}$ (instead of a tree of accepting transcripts for $\mathbb{x}$). This reflects that the node is searching for complete transcripts instead of just a new unique challenge ρ_i. This change makes the algorithm easier to understand graphically yet harder to express algorithmically. This is the reason why we use the recursive definition for the analysis. In this example, the helper algorithm is tasked by $\mathbf{E}_{\mathsf{IP}}^{\mathsf{SRKS}}$ to return a $(3, 2, 2)$-tree of accepting transcripts for $\mathbb{x}$. After the root node job has finished running, the current tree is a $(3, 1, 1)$-tree of accepting transcripts for $\mathbb{x}$, depicted in Fig. 8. The edges highlighted in red correspond to rejecting transcripts, and the edges in blue are early abort challenges, meaning that if this challenge leads to a rejecting transcript, the node will also reject. Then, when all layer 2 nodes are carried out, the resulting tree is a $(3, 2, 1)$-tree of accepting transcripts for $\mathbb{x}$, exhibited in Fig. 9. Finally, when all nodes have finished executing, the resulting tree is a $(3, 2, 2)$-tree of accepting transcripts for $\mathbb{x}$. This tree is shown in Fig. 10.

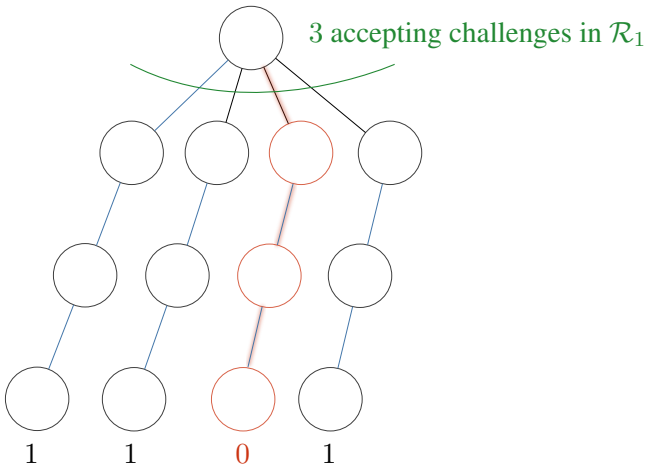

Figure 8: Example of a tree picked by the extractor. At this point, the root node has finished, and we are left with a $(3, 1, 1)$-tree of accepting transcripts for $\mathbb{x}$.

Consider an execution of

$$\mathsf{TreeFinder}_0(\mathbb{x}, \tilde{\mathbf{P}}_{\mathsf{IP}}^{\mathsf{sr}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd})$$

run by the extractor presented in Construction 6.2. Then, any recursive call to the helper algorithm resulting from this instantiation has the following property.

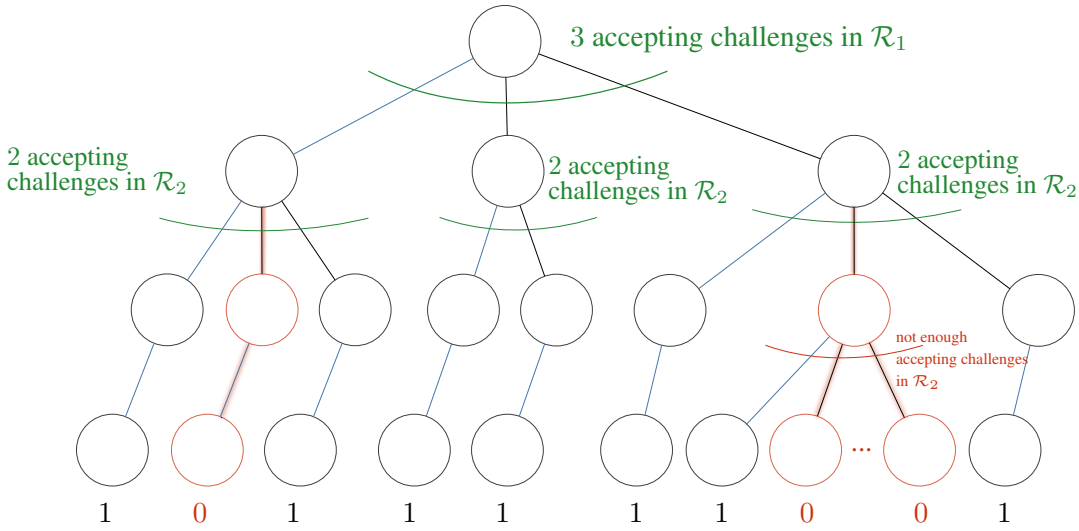

Figure 9: Example of a tree picked by the extractor. At this point, all layer 1 nodes have finished, and we are left with a $(3, 2, 1)$-tree of accepting transcripts for $\mathbb{x}$.

Lemma 6.8. *For any $i \in \{0, \ldots, \mathsf{k}\}$, let $\mathsf{TreeFinder}_i(\mathbb{x}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd})$ be any instantiation that follows from running the Construction 6.2's extractor in Definition 2.38's experiment. We have that*

$$\rho_{i+1} = \mathsf{rnd}_{i+1}(\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) \ .$$

Proof. This statement follows directly from the definition of the state restoration game. Indeed, the challenges $\boldsymbol{\rho}$ are always computed as $\boldsymbol{\rho} = (\mathsf{rnd}_i(\mathbb{x}, \boldsymbol{\alpha}_{[:i+1]}, \boldsymbol{\sigma}_{[:i+1]}))_{i \in [\mathsf{k}]}$, which implies the desired result. □

6.2 Knowledge soundness error

First notice that for every $i \in \{0, \ldots, \mathsf{k} - 1\}$, it holds that

$$\begin{aligned}
1 - \kappa_i &= \prod_{j=i+1}^{\mathsf{k}} \left(1 - \frac{a_j - 1}{N_j}\right) \\
&= \left(1 - \frac{a_{i+1} - 1}{N_{i+1}}\right) \prod_{j=i+2}^{\mathsf{k}} \left(1 - \frac{a_j - 1}{N_j}\right) \\
&= \frac{N_{i+1} - a_{i+1} - 1}{N_{i+1}} \prod_{j=i+2}^{\mathsf{k}} \left(1 - \frac{a_j - 1}{N_j}\right) \\
&= \frac{N_{i+1} - a_{i+1} - 1}{N_{i+1}} (1 - \kappa_{i+1}) \ . \qquad (9)
\end{aligned}$$

We now present Lemma 5.8's generalization to the state restoration setup.

Lemma 6.9. *Recall that, for every $i \in [\mathsf{k} + 1]$, pv_i is the length of all possible i-th messages sent by the prover. Let*

$$T \coloneqq \{0,1\}^{\leq n} \times \left(\bigtimes_{j=1}^{\mathsf{k}+1} \{0,1\}^{\mathsf{pv}_j}\right) \times (\{0,1\}^s)^{\mathsf{k}+1}$$

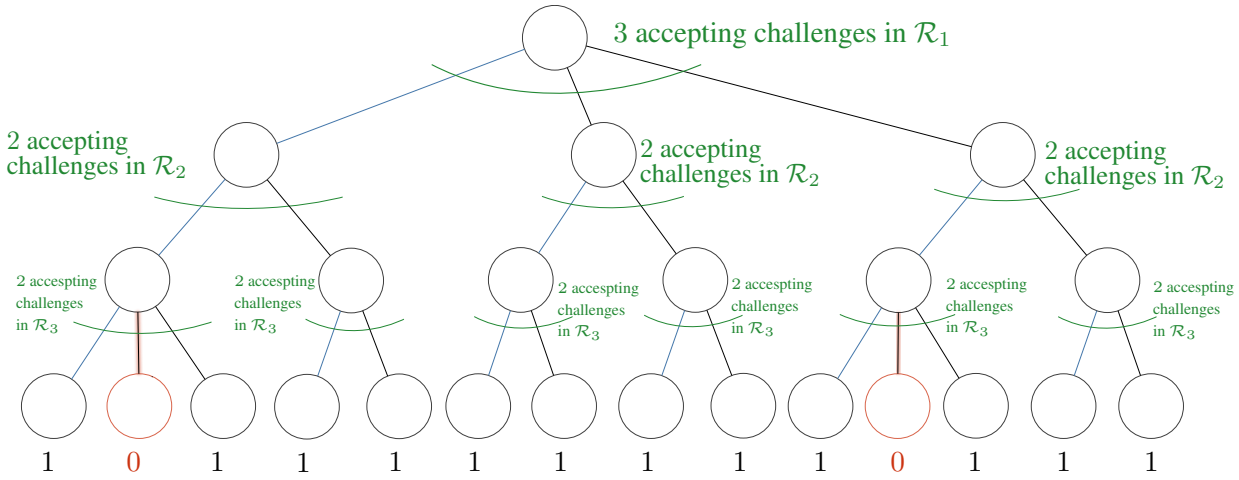

Figure 10: Example of a tree picked by the extractor. At this point, all layer 2 nodes have finished, and we are left with a $(3, 2, 2)$-tree of accepting transcripts for $\mathbb{x}$.

be the set of all instances, prover messages, and salt strings triplet. Then, for every $i \in \{0, \ldots, \mathsf{k}-1\}$, $\ell \in \{0, \ldots, N_i\}$, $(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T$ *and* $\nu_i \in \mathcal{N}_i$, *we have*

$$\Pr\left[\begin{array}{l} T \neq \bot \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) \\ \textit{conditioned on} \\ Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) = \ell \end{array} \middle| \begin{array}{rcl} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu^{\mathsf{ea}}_{i+1}(\nu_i)) \end{array}\right] = \frac{\ell}{N_{i+1}} \ .$$

Proof. We know by Lemma 6.8 that $\rho_{i+1} = \mathsf{rnd}(\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]})$, which implies that ρ_{i+1} is uniformly distributed. Morover, by definition of $Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i)$, the value of ρ_{i+1} is independent of ℓ. Therefore, this equality holds. □

This result reflects the probability that ρ_{i+1} is a good chalenge for $(\mathbb{x}, \nu_{i+1}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \mathbf{rnd})$. Indeed, the first condition ensures that the challenge is successful, while the second, combined with Lemma 6.9, ensures that it is consistent. The following result follows directly from the definition of a conditional probability.

Corollary 6.10. *For every* $i \in \{0, \ldots, \mathsf{k}-1\}$, $\ell \in \{0, \ldots, N_i\}$, $(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T$ *and* $\nu_i \in \mathcal{N}_i$, *we have*

$$\Pr\left[\begin{array}{l} T \neq \bot \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) \\ \wedge\, Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) = \ell \end{array} \middle| \begin{array}{rcl} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu^{\mathsf{ea}}_{i+1}(\nu_i)) \end{array}\right]$$
$$= \frac{\ell}{N_{i+1}} \Pr\left[Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) = \ell \;\middle|\; \mathbf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big)\right] \ .$$

Lemma 5.7 also has a natural generalization to the state restoration setup.

Lemma 6.11. *For every* $i \in \{0, \ldots, \mathsf{k}-1\}$, *it holds that*

$$\sum_{(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T} \Pr\left[Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) > 0 \;\middle|\; \begin{array}{rcl} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \end{array}\right] \leq t+1 \ .$$

Proof. Given any execution of the state restoration game

$$(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \ ,$$

we define the *extended trace* to be the tuple $(\overline{\mathsf{tr}^{\mathsf{sr}}_1}, \ldots, \overline{\mathsf{tr}^{\mathsf{sr}}_{\mathsf{k}}})$ where $\overline{\mathsf{tr}^{\mathsf{sr}}_i} = \mathsf{tr}^{\mathsf{sr}}_i \parallel (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}, \boldsymbol{\rho}_i)$ for every $i \in [\mathsf{k}]$. This ensures that all partial transcripts $\{(\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}, \boldsymbol{\rho}_i)\}_{i\in[\mathsf{k}]}$ resulting from the output of the game lies in the new traces. Note that for every $i \in [\mathsf{k}]$, the length of $\overline{\mathsf{tr}}_i$ is at most $t+1$ whereas the length of $\mathsf{tr}^{\mathsf{sr}}_i$ is at most t. Then,

$$\begin{aligned}
&\sum_{(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T} \Pr\left[Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) > 0 \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \end{array} \right] \\
&= \sum_{(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T} \sum_{j=1}^{t+1} \Pr\left[\begin{array}{l} Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) > 0 \\ \text{conditioned on} \\ \overline{\mathsf{tr}^{\mathsf{sr}}_{i,j}} = (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \end{array} \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \right] \\
&\quad \cdot \Pr\left[\overline{\mathsf{tr}^{\mathsf{sr}}_{i,j}} = (\mathbb{x}, \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \;\middle|\; \begin{array}{ccc} \mathsf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \end{array} \right] \\
&\leq \sum_{j=1}^{t+1} \sum_{(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T} \Pr\left[\overline{\mathsf{tr}^{\mathsf{sr}}_{i,j}} = (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \;\middle|\; \begin{array}{ccc} \mathsf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{SP}}) \end{array} \right] \\
&\leq t+1 \ .
\end{aligned}$$

□

Similarly to the fractions of accepting challenges $\epsilon(\mathbb{x})$, we are interested in counting the fraction of random oracles list **rnd** that would make the state restoration accepts.

Definition 6.12. *We say that a list of random oracles* **rnd** *is accepting if*

$$(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}})$$

is such that $\mathbf{V}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\rho}) = 1$. *Moreover, let* ϵ^{sr} *be the fraction of accepting* **rnd** *within* $\mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big)$*'s support.*

The following theorem is the core proof of Theorem 6.1's success probability statement.

Theorem 6.13. *It holds that*

$$\begin{aligned}
&\Pr\left[T \neq \perp \;\middle|\; \begin{array}{ccc} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_0(\mathbb{x}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}) \end{array} \right] \\
&\geq (t+1) \cdot (\epsilon^{\mathsf{sr}} - \kappa_i) \ ,
\end{aligned}$$

where κ_i *is defined in Eq.* (1).

Proof. We first show that

$$\Pr\left[T \neq \bot \;\middle|\; \begin{array}{ccc} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_i(\mathbb{x}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}) \end{array}\right] \geq \frac{\epsilon^{\mathsf{sr}} - (t+1)\kappa_i}{1-\kappa_i} \tag{10}$$

holds for every $i \in \{0, \ldots, \mathsf{k}\}$. Instantiating this result with $i = 0$ implies the result we are looking for since

$$\frac{\epsilon^{\mathsf{sr}} - (t+1)\kappa_0}{1-\kappa_0} \geq (t+1) \cdot (\epsilon^{\mathsf{sr}} - \kappa_i) \ .$$

We prove Eq. (10) by induction on i going from k to 0. The base case is straightforward forward since the right-hand side reduces to ϵ^{sr}, which is, by definition of the latter, exactly what the left-hand side is.

For the inductive step, fix $i \in \{0, \ldots, \mathsf{k}\}$ and assume the statement holds for every $j \in \{\mathsf{k}, \ldots, i+1\}$. It is straightforward to see that we need the following conditions for an instantiation of $\mathsf{TreeFinder}_i$ to succeed in finding a tree of accepting transcripts. First, the instantiation must pass the early abort tests, and second, the number of good challenges should be large enough. Hence,

$$\begin{aligned}
&\Pr\left[T \neq \bot \;\middle|\; \begin{array}{ccc} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_i(\mathbb{x}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}) \end{array}\right] \\
&= \Pr\left[\begin{array}{l} T \neq \bot \\ \wedge\, Y^{\mathsf{sr}}_{i+1}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \mathbf{rnd}, \nu_i) \geq a_{i+1} \end{array} \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu^{\mathsf{ea}}_{i+1}(\nu_i)) \end{array}\right] \\
&= \sum_{(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T} \sum_{\ell=a_{i+1}}^{N_{i+1}} \Pr\left[\begin{array}{l} T \neq \bot \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) \\ \text{conditioned on} \\ \wedge\, Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) = \ell \end{array} \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu^{\mathsf{ea}}_{i+1}(\nu_i)) \end{array}\right] \\
&\quad \cdot \Pr\left[Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) = \ell \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \end{array}\right] ,
\end{aligned}$$

which by Lemma 6.9 is

$$\begin{aligned}
&\geq \sum_{(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T} \sum_{\ell=a_{i+1}}^{N_{i+1}} \frac{\ell}{N_{i+1}} \Pr\left[Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) = \ell \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \end{array}\right] \\
&\geq \sum_{(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T} \sum_{\ell=a_{i+1}}^{N_{i+1}} \frac{N_{i+1}}{N_{i+1} - a_{i+1} + 1} \left(\frac{\ell}{N_{i+1}} - \frac{a_{i+1}-1}{N_{i+1}}\right) \\
&\quad \cdot \Pr\left[Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) = \ell \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \end{array}\right] \\
&\geq \sum_{(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T} \sum_{\ell=1}^{N_{i+1}} \frac{N_{i+1}}{N_{i+1} - a_{i+1} + 1} \left(\frac{\ell}{N_{i+1}} - \frac{a_{i+1}-1}{N_{i+1}}\right)
\end{aligned}$$

$$\cdot \Pr\left[Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) = \ell \;\middle|\; \begin{array}{rcl} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \end{array} \right] ,$$

and by using Corollary 6.10 we get

$$\begin{aligned}
&\geq \frac{N_{i+1}}{N_{i+1}-a_{i+1}+1} \sum_{\ell=1}^{N_{i+1}} \sum_{(\mathbb{x}',\boldsymbol{\alpha}',\boldsymbol{\sigma}')\in T} \Bigg(\\
&\quad \Pr\left[\begin{array}{l} T \neq \bot \\ \wedge (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) \\ \wedge Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) = \ell \end{array} \;\middle|\; \begin{array}{rcl} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu^{\mathsf{ea}}_{i+1}(\nu_i)) \end{array} \right] \\
&\quad - \frac{a_{i+1}-1}{N_{i+1}} \Pr\left[Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) > 0 \;\middle|\; \begin{array}{rcl} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \end{array} \right] \Bigg) \\
&\geq \frac{N_{i+1}}{N_{i+1}-a_{i+1}+1} \\
&\quad \Bigg(\Pr\left[T \neq \bot \;\middle|\; \begin{array}{rcl} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu^{\mathsf{ea}}_{i+1}(\nu_i)) \end{array} \right] \\
&\quad - \frac{a_{i+1}-1}{N_{i+1}} \sum_{(\mathbb{x}',\boldsymbol{\alpha}',\boldsymbol{\sigma}')\in T} \Pr\left[Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) > 0 \;\middle|\; \begin{array}{rcl} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \end{array} \right] \Bigg) .
\end{aligned}$$

Note that $\nu^{\mathsf{ea}}_{i+1}(\nu_i)$ is uniformly distributed and not dependent on the algorithm's inputs; therefore, we can apply the induction hypothesis. Combing the induction hypothesis with Lemma 6.11, we obtain

$$\begin{aligned}
&\Pr\left[T \neq \bot \;\middle|\; \begin{array}{rcl} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i}(\mathbb{x}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}) \end{array} \right] \\
&\geq \frac{N_{i+1}}{N_{i+1}-a_{i+1}+1} \cdot \left(\frac{\epsilon^{\mathsf{sr}} - (t+1)\kappa_{i+1}}{1-\kappa_{i+1}} - \frac{a_{i+1}-1}{N_{i+1}}(t+1) \right) && \text{Eq. (9)} \\
&= \frac{\epsilon^{\mathsf{sr}} - (t+1)\kappa_{i+1}}{1-\kappa_i} - \frac{a_{i+1}-1}{N_{i+1}-a_{i+1}+1}(t+1) \\
&= \frac{1}{1-\kappa_i} \left(\epsilon^{\mathsf{sr}} - (t+1) \left(\kappa_{i+1} - (1-\kappa_i) \frac{a_{i+1}-1}{N_{i+1}-a_{i+1}+1} \right) \right) && \text{Eq. (9)} \\
&= \frac{\epsilon^{\mathsf{sr}} - (t+1)\kappa_i}{1-\kappa_i} ,
\end{aligned}$$

which concludes Eq. (10)'s proof by induction and, therefore, the proof of theorem's statement. □

We assert that Construction 6.2 satisfies Theorem 6.1's bound on the state restoration knowledge error.

Indeed, assuming $|\mathbb{x}| \leq n$, we have by Theorem 6.13

$$
\begin{aligned}
&\Pr\left[\begin{array}{l|rcl} & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\mathbb{w}) \notin R & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge\, b = 1 & b & \leftarrow & \mathbf{V}_{\mathsf{IP}}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\rho}) \\ & \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SRKS}}_{\mathsf{IP}}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{tr}^{\mathsf{sr}},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array}\right] \\
&= \Pr\left[\begin{array}{l|rcl} & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ & b & \leftarrow & \mathbf{V}_{\mathsf{IP}}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\rho}) \\ T = \bot & \mathbb{w} & \leftarrow & \mathbf{E}^{\mathsf{SRKS}}_{\mathsf{IP}}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{tr}^{\mathsf{sr}},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge\, b = 1 & \mathsf{rnd}_1 & \leftarrow & \mathcal{U}(\mathsf{r}_1)|_{\mathsf{tr}^{\mathsf{sr}}_1} \\ & \vdots & & \\ & \mathsf{rnd}_{\mathsf{k}} & \leftarrow & \mathcal{U}(\mathsf{r}_{\mathsf{k}})|_{\mathsf{tr}^{\mathsf{sr}}_{\mathsf{k}}} \\ & T & \leftarrow & \mathsf{TreeFinder}_0(\mathbb{x},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},(\mathsf{rnd}_i)_{i\in[\mathsf{k}]}) \end{array}\right] \\
&= \Pr\left[\begin{array}{l|rcl} & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T = \bot & b & \leftarrow & \mathbf{V}_{\mathsf{IP}}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\rho}) \\ \wedge\, b = 1 & T & \leftarrow & \mathsf{TreeFinder}_0(\mathbb{x},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd}) \end{array}\right] \\
&= 1 - \Pr\left[T \neq \bot \;\middle|\; \begin{array}{rcl} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_0(\mathbb{x},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd}) \end{array}\right] \\
&\quad \underbrace{- \Pr\left[b = 0 \;\middle|\; \begin{array}{rcl} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ b & \leftarrow & \mathbf{V}_{\mathsf{IP}}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\rho}) \end{array}\right]}_{=1-\epsilon^{\mathsf{sr}}} \\
&\geq (t+1)\kappa_0 \ ,
\end{aligned}
$$

since the two events $b = 0$ and $(\mathbb{x},\mathbb{w}) \in R$ are disjoint. The first equality comes from their two experiments being indistinguishable since the random oracles of the right-hand side are restricted to match the traces of the others.

6.3 Extraction time

Proving that Construction 6.2 satisfies Theorem 6.1 claimed extraction time is more involved than proving the bounds of the success probability. To do so, we start by defining some useful random variables.

Definition 6.14. *For every $i \in \{0,\ldots,\mathsf{k}\}$, $\mathbb{x}$, $\boldsymbol{\alpha}$, $\boldsymbol{\sigma}$, $\boldsymbol{\rho}$, $\mathbf{rnd}$ and ν_i, let*

$$\Lambda'_i(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i)$$

be the number of challenges picked by

$$\mathsf{TreeFinder}_i(\mathbb{x},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd};\nu_i)$$

without considering the first one tested, that is ρ_{i+1}. Moreover, let

$$\Lambda_i(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i)$$

be the number of challenges picked by this instantiation, including ρ_{i+1}.

The second random variable depicts the cardinality of the set S used in $\mathsf{TreeFinder}_i$ at the end of the algorithm execution. Note that

$$\Lambda_i(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) \leq \Lambda'_i(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) + 1 \ . \tag{11}$$

If the consistency tests (Line 5f) are unsatisfied, the current instantiation executes no recursive calls. Hence, challenges that pass these tests induce an additional execution time. For the analysis, we subdivide these challenges into two categories: successful and others.

Definition 6.15. *For every $i \in \{0, \ldots, \mathsf{k}-1\}$, $\mathbb{x}$, $\boldsymbol{\alpha}$, $\boldsymbol{\sigma}$, $\boldsymbol{\rho}$, $\mathbf{rnd}$ and ν_i, let*

$$X_i^{\top}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i)$$

be the number of challenges ρ'_{i+1} picked by

$$\mathsf{TreeFinder}_i(\mathbb{x}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu_i) \ ,$$

including ρ_{i+1}, that are good for $(\mathbb{x}, \nu_i, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \mathbf{rnd})$. Conversely, let

$$X_i^{\perp}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i)$$

denotes the number of challenges ρ'_{i+1} picked by $\mathsf{TreeFinder}_i$, including ρ_{i+1}, that are consistent with $(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \mathbf{rnd})$ but not successful for $(\mathbb{x}, \nu_{i+1}^{\rho'_{i+1}}(\nu_i), \boldsymbol{\alpha}, \boldsymbol{\sigma}, \mathbf{rnd})$. Moreover, we define $X_{\mathsf{k}}^{\top}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_{\mathsf{k}}) \coloneqq 0$ and $X_{\mathsf{k}}^{\perp}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_{\mathsf{k}}) \coloneqq 0$.

Focusing on the consistent challenges picked by $\mathsf{TreeFinder}_i$, we are also interested in the number of invocations to the state restoration game done by recursive calls resulting from these consistent challenges.

Definition 6.16. *For every $i \in \{0, \ldots, \mathsf{k}-1\}$, $\mathbb{x}$, $\boldsymbol{\alpha}$, $\boldsymbol{\sigma}$, $\boldsymbol{\rho}$, $\mathbf{rnd}$ and ν_i, let*

$$I_i(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i)$$

counts the number of invocations performed by

$$\mathsf{TreeFinder}_i(\mathbb{x}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu_i) \ .$$

Moreover, let

$$I_i^{\top}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i)$$

counts the number of invocations done by the recursive calls that result from challenges that are good for $(\mathbb{x}, \nu_i, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \mathbf{rnd})$. Conversely, we denote by

$$I_i^{\perp}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i)$$

the number of invocations done by the recursive calls that result from challenges ρ'_{i+1} that are consistent with $(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \mathbf{rnd})$ but not successful for $(\mathbb{x}, \nu_{i+1}^{\rho'_{i+1}}(\nu_i), \boldsymbol{\alpha}, \boldsymbol{\sigma}, \mathbf{rnd})$. Finally, let $I_{\mathsf{k}}^{\top}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_{\mathsf{k}}) \coloneqq 0$ and $I_{\mathsf{k}}^{\perp}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_{\mathsf{k}}) \coloneqq 0$.

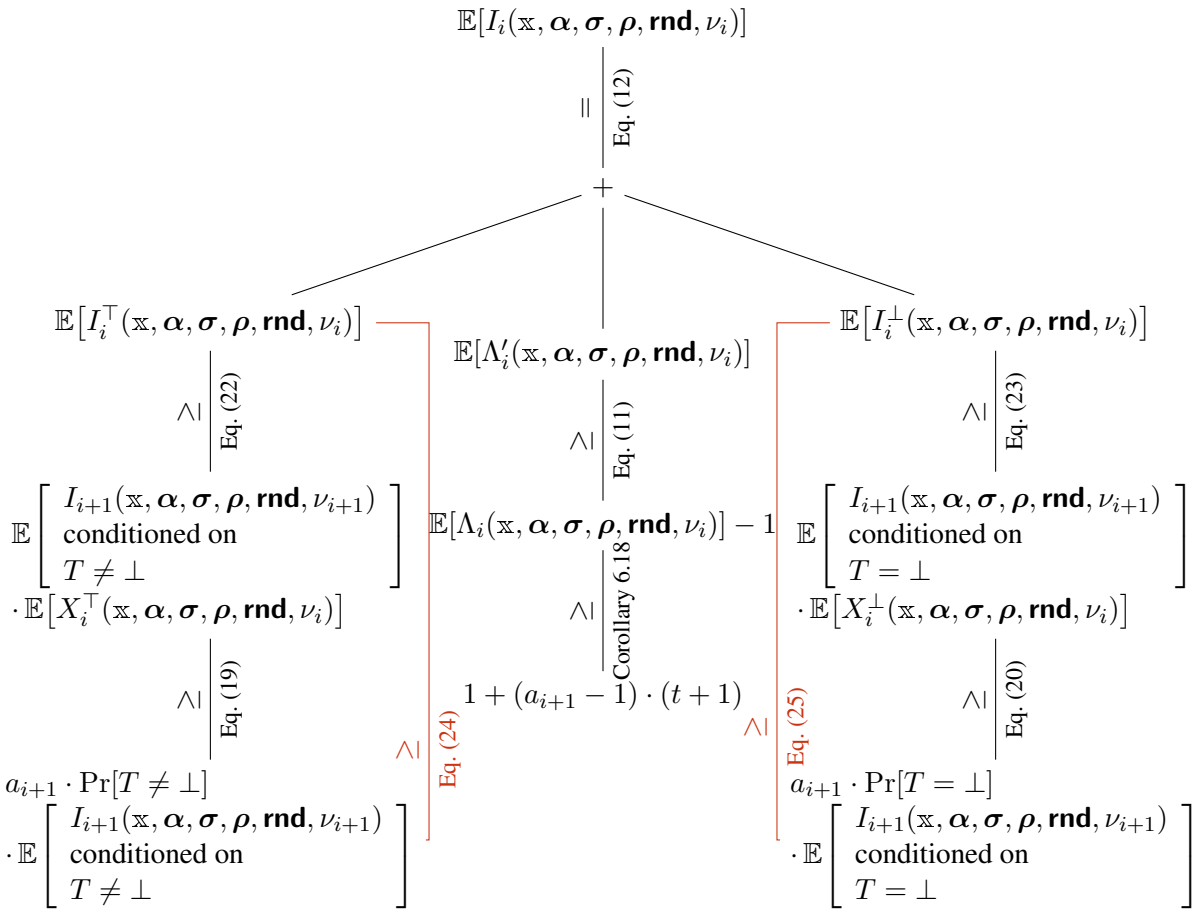

Figure 11: Lemma 6.19's proof strategy for expressing I_i recursively.

By the previous definitions, it follows that for every $i \in \{0, \ldots, \mathsf{k}\}$, $\boldsymbol{\alpha}$, $\boldsymbol{\sigma}$, $\boldsymbol{\rho}$, $\mathbf{rnd}$ and ν'_i

$$I_i(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) = I_i^{\top}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) + I_i^{\perp}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) + \Lambda'_i(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) \ . \quad (12)$$

This is the starting point of the proof; we use this decomposition to find an upper bound I_i in terms of I_{i+1} by upper bounding the values $I_i^{\top}$, $I_i^{\perp}$ and Λ_i with an expression including I_{i+1}. This recursive statement is captured in Lemma 6.19. Then, we inject this recursive bound in a proof by induction to show Theorem 6.20. The proof strategy for Lemma 6.19 and the intermediate results necessary are depicted in Fig. 11. For simplicity, we have removed the experiments and conditioning on random variables that are universally quantified. We now continue by proving every result present in this figure.

It is straightforward to see that the random variable Λ'_i corresponds to the number of the invocations done to $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}$ by $\mathsf{TreeFinder}_i$ without considering invocations performed within the recursive calls. By the

early abort test, it follows that

$$\Pr\left[\begin{array}{l|rcl} \Lambda'_i(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i)=0 & \nu_i & \leftarrow & \mathcal{N}_i \\ \text{conditioned on} & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ T\neq\bot & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ & T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd};\nu^{\mathsf{ea}}_{i+1}(\nu_i)) \end{array}\right] = 1 \ . \tag{13}$$

By fixing the recursive part of the randomness and conditioning on the early abort test passing, we claim that the number of challenges picked by an instantiation of $\mathsf{TreeFinder}_i$ follows a NHG distribution. For every $\mathbb{x}'$, $\boldsymbol{\alpha}'$, $\boldsymbol{\sigma}'$ and $\boldsymbol{\nu}'_{i+1}$, the random variable

$$\left\{\begin{array}{l|rcl} \Lambda'_i(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i) & \nu_{i+1} & \leftarrow & \mathcal{N} \\ \text{conditioned on} & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ T\neq\bot & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge\,\boldsymbol{\nu}'_{i+1}=\boldsymbol{\nu}_{i+1}(\rho_i) & T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \wedge\,Y^{\mathsf{sr}}_{i+1}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\mathbf{rnd},\nu_i)=\ell & & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd};\nu^{\mathsf{ea}}_{i+1}(\nu_i)) \\ \wedge\,(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & & & \\ \quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) & & & \end{array}\right\}$$

follows a NHG with $2^{\mathsf{r}_{i+1}}-1$ balls of which $\ell-1$ are success balls, and it has to find $a_{i+1}-1$ success balls (there are only $2^{\mathsf{r}_{i+1}}-1$ total balls and $\ell-1$ success balls because we already use ρ_{i+1} for the early abort test). This result follows directly from Definition 2.14. Moreover, this implies

$$\begin{aligned} &\mathbb{E}\left[\begin{array}{l|rcl} \Lambda'_i(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i) & \nu_i & \leftarrow & \mathcal{N}_i \\ \text{conditioned on} & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ T\neq\bot & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge\,\boldsymbol{\nu}'_{i+1}=\boldsymbol{\nu}_{i+1}(\rho_i) & T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \wedge\,Y^{\mathsf{sr}}_{i+1}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\mathbf{rnd},\nu_i)=\ell & & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd};\nu^{\mathsf{ea}}_{i+1}(\nu_i)) \\ \wedge\,(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & & & \\ \quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) & & & \end{array}\right] \\ &=(a_{i+1}-1)\frac{2^{\mathsf{r}i+1}}{\ell} \ . \end{aligned} \tag{14}$$

We need the following result to find an upper bound on Λ'_i.

Lemma 6.17. *For every $i\in\{0,\ldots,\mathsf{k}\}$, it holds that*

$$\begin{aligned} &\mathbb{E}\left[\begin{array}{l|rcl} \Lambda'_i(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i) & \nu_i & \leftarrow & \mathcal{N}_i \\ \textit{conditioned on} & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ T\neq\bot & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ & T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd};\nu^{\mathsf{ea}}_{i+1}(\nu_i)) \end{array}\right] \\ &\quad\cdot\Pr\left[\begin{array}{l|rcl} T\neq\bot & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ & T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd}) \end{array}\right] \\ &\leq(a_{i+1}-1)\cdot(t+1) \ . \end{aligned}$$

Proof. Note that for every $(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T$ and $\boldsymbol{\nu}'_{i+1}$, we have

$$\Pr\left[\begin{array}{l|l} T \neq \bot & \nu_i \leftarrow \mathcal{N}_i \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & \mathbf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge\, Y^{\mathsf{sr}}_{i+1}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \mathbf{rnd}, \nu_i) = 0 & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \text{conditioned on} & \qquad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu^{\mathsf{ea}}_{i+1}(\nu_i)) \\ \wedge\, \boldsymbol{\nu}'_{i+1} = \boldsymbol{\nu}_{i+1}(\rho_i)(\rho_i) & \end{array}\right] = 0 \tag{15}$$

since if the recursive call return succeeds and the consistency tests pass, there is, by definition, at least one good challenge for $(\mathbb{x}, \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd})$. The following result holds for any choice of $\boldsymbol{\nu}'_{i+1}$, which implies the desired result. By Corollary 6.10 and Eq. (14)

$$\begin{aligned}
&\mathbb{E}\left[\begin{array}{l|l} \Lambda'_i(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) & \nu_i \leftarrow \mathcal{N}_i \\ \text{conditioned on} & \mathbf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ T \neq \bot & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge\, \boldsymbol{\nu}'_{i+1} = \boldsymbol{\nu}_{i+1}(\rho_i) & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & \qquad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu^{\mathsf{ea}}_{i+1}(\nu_i)) \end{array}\right] \\
&\quad \cdot \Pr\left[\begin{array}{l|l} T \neq \bot & \mathbf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \text{conditioned on} & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge\, \boldsymbol{\nu}'_{i+1} = \boldsymbol{\nu}_{i+1}(\rho_i) & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & \qquad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}) \end{array}\right] \\
&= \sum_{(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T} \sum_{\ell=0}^{2^{\mathsf{r}_{i+1}}} \\
&\quad \mathbb{E}\left[\begin{array}{l|l} \Lambda'_i(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) & \\ \text{conditioned on} & \nu_i \leftarrow \mathcal{N}_i \\ T \neq \bot & \mathbf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \wedge\, \boldsymbol{\nu}'_{i+1} = \boldsymbol{\nu}_{i+1}(\rho_i) & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & \qquad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu^{\mathsf{ea}}_{i+1}(\nu_i)) \\ \wedge\, Y^{\mathsf{sr}}_{i+1}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \mathbf{rnd}, \nu_i) = \ell & \end{array}\right] \\
&\quad \underbrace{\cdot \Pr\left[\begin{array}{l|l} T \neq \bot & \nu_i \leftarrow \mathcal{N}_i \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & \mathbf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge\, Y^{\mathsf{sr}}_{i+1}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \mathbf{rnd}, \nu_i) = \ell & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \text{conditioned on} & \qquad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu^{\mathsf{ea}}_{i+1}(\nu_i)) \\ \wedge\, \boldsymbol{\nu}'_{i+1} = \boldsymbol{\nu}_{i+1}(\rho_i) & \end{array}\right]}_{= 0 \text{ when } \ell = 0 \text{ by Eq. (15)}} \\
&\leq \sum_{(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T} \sum_{\ell=1}^{2^{\mathsf{r}_{i+1}}} (a_{i+1} - 1) \cdot \Pr\left[Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) = \ell \;\middle|\; \begin{array}{lcl} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \end{array}\right] \\
&= (a_{i+1} - 1) \sum_{(\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \in T} \cdot \Pr\left[Y^{\mathsf{sr}}_{i+1}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \mathbf{rnd}, \nu_i) > 0 \;\middle|\; \begin{array}{lcl} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \end{array}\right],
\end{aligned}$$

which by Lemma 6.11 is upper bounded by $(a_{i+1}-1)\cdot(t+1)$. □

Now, we can show the following result.

Corollary 6.18. *For every $i \in \{0,\dots,\mathsf{k}\}$, the following holds*

$$\mathbb{E}\left[\Lambda_i(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i) \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array}\right] \leq 1+(a_{i+1}-1)\cdot(t+1)\ .$$

Proof. We have

$$\begin{aligned}
&\mathbb{E}\left[\Lambda_i(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i) \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array}\right] && \text{Eq. (11)}\\
&\leq 1+\mathbb{E}\left[\Lambda'_i(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i) \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array}\right] \\
&= 1+\underbrace{\mathbb{E}\left[\begin{array}{l}\Lambda'_i(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i)\\ \text{conditioned on}\\ T=\bot\end{array} \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N} \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd};\nu^{\mathsf{ea}}_{i+1}(\nu_i)) \end{array}\right]}_{=0 \text{ by Eq. (13)}} \\
&\quad\cdot\Pr\left[T=\bot \;\middle|\; \begin{array}{ccc} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd}) \end{array}\right] \\
&\quad+\mathbb{E}\left[\begin{array}{l}\Lambda'_i(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i)\\ \text{conditioned on}\\ T\neq\bot\end{array} \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd};\nu^{\mathsf{ea}}_{i+1}(\nu_i)) \end{array}\right] && \text{Lemma 6.17}\\
&\quad\cdot\Pr\left[T\neq\bot \;\middle|\; \begin{array}{ccc} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd}) \end{array}\right] \\
&= (a_{i+1}-1)\cdot(t+1)\ ,
\end{aligned}$$

which concludes the proof of this corollary. □

Consider the two following quantities. For every $(\mathbb{x}',\boldsymbol{\alpha}',\boldsymbol{\sigma}')\in T$, let

$$u \coloneqq \left|\left\{\rho'_{i+1}\in\{0,1\}^{2^{\mathsf{k}_{i+1}}} \;\middle|\; \begin{array}{l} T\neq\bot \\ \wedge\,(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) \\ \quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) \end{array}\right\}\right|\ ,$$

and

$$v \coloneqq \left| \left\{ \rho'_{i+1} \in \{0,1\}^{2^{\mathsf{k}_{i+1}}} \;\middle|\; \begin{array}{l} T = \bot \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) \end{array} \right\} \right| ,$$

where for every ρ'_{i+1}

$$(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \leftarrow \Gamma^{\mathsf{sr}}(s, (\mathsf{rnd}_1, \ldots, \mathsf{rnd}_i, \mathsf{rnd}_{i+1}|_{((\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}), \rho'_{i+1})}, \mathsf{rnd}_{i+2}, \ldots, \mathsf{rnd}_{\mathsf{k}}), \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}})$$

and

$$T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}) \ .$$

Given these notations, it is straightforward to see that

$$\Pr\left[\begin{array}{l} T \neq \bot \\ \text{conditioned on} \\ \mathbf{rnd}' = \mathbf{rnd} \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) \end{array} \middle| \begin{array}{ccl} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}) \end{array} \right] = \frac{u}{u+v} \tag{16}$$

and

$$\Pr\left[\begin{array}{l} T = \bot \\ \text{conditioned on} \\ \mathbf{rnd}' = \mathbf{rnd} \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) \end{array} \middle| \begin{array}{ccl} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}) \end{array} \right] = \frac{v}{u+v} \ , \tag{17}$$

for all $\mathbf{rnd}' \in \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big)$ and $(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T$.

Next, note that the random variable

$$\left\{ \begin{array}{l} X_i^{\bot}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) \\ \quad +(a_{i+1} - 1) \\ \text{conditioned on} \\ \mathbf{rnd}' = \mathbf{rnd} \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) \\ T \neq \bot \end{array} \middle| \begin{array}{ccl} \nu_i & \leftarrow & \mathcal{N} \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu^{\mathsf{ea}}_{i+1}(\rho_i)) \end{array} \right\}$$

follows an NHG distribution with $u-1$ good balls, $u+v-1$ total balls and where we are looking for $a_{i+1}-1$ good balls. The minus one comes from the fact that the first good ball has already been picked for the early abort branch. Therefore, it follows from Definition 2.14 that

$$\mathbb{E}\left[\begin{array}{l} X_i^{\bot}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) \\ \text{conditioned on} \\ \mathbf{rnd}' = \mathbf{rnd} \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) \\ T \neq \bot \end{array} \middle| \begin{array}{ccl} \nu_i & \leftarrow & \mathcal{N} \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu^{\mathsf{ea}}_{i+1}(\rho_i)) \end{array} \right] \leq (a_{i+1}-1)\frac{v}{u} \ . \tag{18}$$

We now give upper bounds on $X_i^\top$ and $X_i^\perp$'s expectations. We begin with the former.

Upper bound on $X_i^\top$'s expectation. By conditioning on T being $\perp$ or not, we have

$$
\begin{aligned}
&\mathbb{E}\left[\begin{array}{l|l}
X_i^\top(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i) & \\
\text{conditioned on} & \\
\mathbf{rnd}'=\mathbf{rnd} & \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \\
\wedge(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & \\
\quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) &
\end{array}\right] \\
&=\mathbb{E}\left[\begin{array}{l|l}
X_i^\top(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i) & \\
\text{conditioned on} & \\
\mathbf{rnd}'=\mathbf{rnd} & \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd};\nu^{\mathsf{ea}}_{i+1}(\rho_i)) \end{array} \\
\wedge(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & \\
\quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) & \\
\wedge T\neq\perp &
\end{array}\right] \\
&\quad\cdot\Pr\left[\begin{array}{l|l}
T\neq\perp & \\
\text{conditioned on} & \\
\mathbf{rnd}'=\mathbf{rnd} & \begin{array}{ccc} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd}) \end{array} \\
\wedge(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & \\
\quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) &
\end{array}\right] \\
&\quad+\underbrace{\mathbb{E}\left[\begin{array}{l|l}
X_i^\top(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i) & \\
\text{conditioned on} & \\
\mathbf{rnd}'=\mathbf{rnd} & \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd};\nu^{\mathsf{ea}}_{i+1}(\rho_i)) \end{array} \\
\wedge(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & \\
\quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) & \\
\wedge T=\perp &
\end{array}\right]}_{=0} \\
&\quad\cdot\Pr\left[\begin{array}{l|l}
T=\perp & \\
\text{conditioned on} & \\
\mathbf{rnd}'=\mathbf{rnd} & \begin{array}{ccc} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd}) \end{array} \\
\wedge(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & \\
\quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) &
\end{array}\right] \\
&\leq a_{i+1}\cdot\Pr\left[\begin{array}{l|l}
T\neq\perp & \\
\text{conditioned on} & \\
\mathbf{rnd}'=\mathbf{rnd} & \begin{array}{ccc} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd}) \end{array} \\
\wedge(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & \\
\quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) &
\end{array}\right], \qquad (19)
\end{aligned}
$$

since the algorithm stops after finding enough good challenges.

Upper bound on $X_i^\perp$'s expectation. We have

$$
\mathbb{E}\left[\begin{array}{l|l}
X_i^\perp(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i) & \\
\text{conditioned on} & \\
\mathbf{rnd}'=\mathbf{rnd} & \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \\
\wedge(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & \\
\quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) &
\end{array}\right]
$$

$$= \mathbb{E}\left[\begin{array}{l|l} X_i^{\perp}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) & \nu_i \leftarrow \mathcal{N}_i \\ \text{conditioned on} & \mathbf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \mathbf{rnd}' = \mathbf{rnd} & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & \quad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu^{\mathsf{ea}}_{i+1}(\rho_i)) \\ \wedge T \neq \perp & \end{array}\right] \qquad \text{Eq. (18)}$$

$$\cdot \Pr\left[\begin{array}{l|l} T \neq \perp & \mathbf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \text{conditioned on} & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \mathbf{rnd}' = \mathbf{rnd} & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \wedge (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & \quad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & \end{array}\right]$$

$$+ \underbrace{\mathbb{E}\left[\begin{array}{l|l} X_i^{\perp}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) & \nu_i \leftarrow \mathcal{N}_i \\ \text{conditioned on} & \mathbf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \mathbf{rnd}' = \mathbf{rnd} & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & \quad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu^{\mathsf{ea}}_{i+1}(\rho_i)) \\ \wedge T = \perp & \end{array}\right]}_{=1}$$

$$\cdot \Pr\left[\begin{array}{l|l} T = \perp & \mathbf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \text{conditioned on} & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \mathbf{rnd}' = \mathbf{rnd} & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \wedge (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & \quad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & \end{array}\right]$$

$$\leq \Pr\left[\begin{array}{l|l} T = \perp & \mathbf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \text{conditioned on} & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \mathbf{rnd}' = \mathbf{rnd} & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \wedge (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & \quad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & \end{array}\right] \qquad \text{Eq. (17)}$$

$$+ \Pr\left[\begin{array}{l|l} T \neq \perp & \mathbf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \text{conditioned on} & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \mathbf{rnd}' = \mathbf{rnd} & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \wedge (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & \quad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & \end{array}\right] \qquad \text{Eq. (16)}$$

$$\cdot (a_{i+1} - 1)\frac{v}{u}$$

$$= a_{i+1} \cdot \Pr\left[\begin{array}{l|l} T = \perp & \mathbf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \text{conditioned on} & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \mathbf{rnd}' = \mathbf{rnd} & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \wedge (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & \quad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & \end{array}\right]. \qquad (20)$$

Finally, we show the following recursive inequality for I holds.

Lemma 6.19. *For every $i \in \{0, \ldots, \mathsf{k}-1\}$, it holds that*

$$\mathbb{E}\left[I_i(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \right]$$
$$\leq a_{i+1} \cdot \mathbb{E}\left[I_{i+1}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_{i+1}) \;\middle|\; \begin{array}{ccc} \nu_{i+1} & \leftarrow & \mathcal{N}_{i+1} \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \right]$$
$$+ (t+1) \cdot (a_{i+1} - 1) \ .$$

Proof. The invocations to the state restoration game come from two sources: those done within the loop of $\mathsf{TreeFinder}_i$ and within the recursive calls. The former is straightforward to analyze. Indeed, the extractor executes the state restoration game every time we pick a new challenge, except for the early abort test. Hence, the number of times we invoke the state restoration game within the loop is

$$\Lambda'_i(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i)$$

and an upper bound on the expectation of $\Lambda_i = \Lambda'_i + 1$ is given in Corollary 6.18. It remains to show how many invocations are done within the recursive calls.

For every $\ell \in \{0, \ldots, 2^{\mathsf{r}_{i+1}}\}$, $\mathbf{rnd}' \in \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big)$ and $(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T$, we have

$$\mathbb{E}\left[\begin{array}{l} I_i^{\top}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) \\ \text{conditioned on} \\ \mathbf{rnd}' = \mathbf{rnd} \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) \\ \wedge\, X_i^{\top}(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}', \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) = \ell \end{array} \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \right]$$
$$= \ell \cdot \mathbb{E}\left[\begin{array}{l} I_{i+1}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_{i+1}) \\ \text{conditioned on} \\ \mathbf{rnd}' = \mathbf{rnd} \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) \\ \wedge\, T \neq \bot \end{array} \;\middle|\; \begin{array}{ccc} \nu_{i+1} & \leftarrow & \mathcal{N}_{i+1} \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu_{i+1} \end{array} \right] \tag{21}$$

since any subset of $\{0,1\}^{\mathsf{r}_{i+1}}$ with cardinality ℓ of good challenges is equally likely to be sampled.

It follows from Eq. (21) that for every $\mathbf{rnd}' \in \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big)$ and $(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T$,

$$\mathbb{E}\left[\begin{array}{l} I_i^{\top}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) \\ \text{conditioned on} \\ \mathbf{rnd}' = \mathbf{rnd} \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) \end{array} \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \right]$$
$$= \sum_{\ell=0}^{2^{\mathsf{r}_{i+1}}} \mathbb{E}\left[\begin{array}{l} I_i^{\top}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) \\ \text{conditioned on} \\ \mathbf{rnd}' = \mathbf{rnd} \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) = (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) \\ \wedge\, X_i^{\top}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) = \ell \end{array} \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \right]$$

$$
\cdot \Pr\left[\begin{array}{l|rcl} X_i^{\top}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i)=\ell & \nu_i & \leftarrow & \mathcal{N}_i \\ \text{conditioned on} & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \mathbf{rnd}'=\mathbf{rnd} & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]})=(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & & & \end{array}\right]
$$

$$
=\mathbb{E}\left[\begin{array}{l|rcl} I_{i+1}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_{i+1}) & \nu_{i+1} & \leftarrow & \mathcal{N}_{i+1} \\ \text{conditioned on} & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \mathbf{rnd}'=\mathbf{rnd} & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]})=(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ T\neq\perp & & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd};\nu_{i+1}) \end{array}\right]
$$

$$
\cdot\left(\sum_{\ell=0}^{2^{\mathsf{r}}i+1}\ell\Pr\left[\begin{array}{l|rcl} X_i^{\top}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i)=\ell & \nu_i & \leftarrow & \mathcal{N}_i \\ \text{conditioned on} & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \mathbf{rnd}'=\mathbf{rnd} & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & & & \\ \quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) & & & \end{array}\right]\right)
$$

$$
=\mathbb{E}\left[\begin{array}{l|rcl} I_{i+1}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_{i+1}) & \nu_{i+1} & \leftarrow & \mathcal{N}_{i+1} \\ \text{conditioned on} & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \mathbf{rnd}'=\mathbf{rnd} & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) & & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd};\nu_{i+1}) \\ T\neq\perp & & & \end{array}\right]
$$

$$
\cdot\,\mathbb{E}\left[\begin{array}{l|rcl} X_i^{\top}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i) & \nu_i & \leftarrow & \mathcal{N}_i \\ \text{conditioned on} & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \mathbf{rnd}'=\mathbf{rnd} & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & & & \\ \quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) & & & \end{array}\right]. \tag{22}
$$

Likewise,

$$
\mathbb{E}\left[\begin{array}{l|rcl} I_i^{\perp}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i) & \nu_i & \leftarrow & \mathcal{N}_i \\ \text{conditioned on} & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \mathbf{rnd}'=\mathbf{rnd} & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & & & \\ \quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) & & & \end{array}\right]
$$

$$
=\mathbb{E}\left[\begin{array}{l|rcl} I_{i+1}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_{i+1}) & \nu_{i+1} & \leftarrow & \mathcal{N}_{i+1} \\ \text{conditioned on} & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \mathbf{rnd}'=\mathbf{rnd} & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) & & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd};\nu_{i+1}) \\ T=\perp & & & \end{array}\right]
$$

$$
\cdot\,\mathbb{E}\left[\begin{array}{l|rcl} X_i^{\perp}(\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho},\mathbf{rnd},\nu_i) & \nu_i & \leftarrow & \mathcal{N}_i \\ \text{conditioned on} & \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \mathbf{rnd}'=\mathbf{rnd} & (\mathbb{x},\boldsymbol{\alpha},\boldsymbol{\sigma},\boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s,\mathbf{rnd},\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge(\mathbb{x}',\boldsymbol{\alpha}'_{[:i+2]},\boldsymbol{\sigma}'_{[:i+2]}) & & & \\ \quad=(\mathbb{x},\boldsymbol{\alpha}_{[:i+2]},\boldsymbol{\sigma}_{[:i+2]}) & & & \end{array}\right]. \tag{23}
$$

The next step is to develop Eqs. (22) and (23) by using the upper bounds on the $X_i^{\top}$'s expected value and $X_i^{\perp}$'s expected value, introduced in Eqs. (19) and (20).

Combining Eq. (22) with Eq. (19), we have

$$\mathbb{E}\left[\begin{array}{l|l} I_i^{\top}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathsf{rnd}, \nu_i) & \\ \text{conditioned on} & \nu_i \leftarrow \mathcal{N}_i \\ \mathsf{rnd}' = \mathsf{rnd} & \mathsf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & \end{array}\right]$$

$$\leq a_{i+1} \cdot \Pr\left[\begin{array}{l|l} T \neq \bot & \\ \text{conditioned on} & \mathsf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \mathsf{rnd}' = \mathsf{rnd} & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & \qquad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathsf{rnd}) \end{array}\right]$$

$$\cdot\, \mathbb{E}\left[\begin{array}{l|l} I_{i+1}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathsf{rnd}, \nu_{i+1}) & \\ \text{conditioned on} & \nu_{i+1} \leftarrow \mathcal{N}_{i+1} \\ \mathsf{rnd}' = \mathsf{rnd} & \mathsf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ T \neq \bot & \qquad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathsf{rnd}; \nu_{i+1}) \end{array}\right]. \tag{24}$$

Similarly, combining Eq. (22) with Eq. (20), we have

$$\mathbb{E}\left[\begin{array}{l|l} I_i^{\perp}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathsf{rnd}, \nu) & \\ \text{conditioned on} & \nu_i \leftarrow \mathcal{N}_i \\ \mathsf{rnd}' = \mathsf{rnd} & \mathsf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & \end{array}\right]$$

$$\leq a_{i+1} \cdot \Pr\left[\begin{array}{l|l} T = \bot & \\ \text{conditioned on} & \mathsf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \mathsf{rnd}' = \mathsf{rnd} & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & \qquad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathsf{rnd}) \end{array}\right]$$

$$\cdot\, \mathbb{E}\left[\begin{array}{l|l} I_{i+1}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathsf{rnd}, \nu_{i+1}) & \\ \text{conditioned on} & \nu_{i+1} \leftarrow \mathcal{N}_{i+1} \\ \mathsf{rnd}' = \mathsf{rnd} & \mathsf{rnd} \leftarrow \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ \wedge\, (\mathbb{x}', \boldsymbol{\alpha}'_{[:i+2]}, \boldsymbol{\sigma}'_{[:i+2]}) & (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) \xleftarrow{\mathsf{tr}^{\mathsf{sr}}} \Gamma^{\mathsf{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \quad = (\mathbb{x}, \boldsymbol{\alpha}_{[:i+2]}, \boldsymbol{\sigma}_{[:i+2]}) & T \leftarrow \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ \wedge\, T = \bot & \qquad \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathsf{rnd}; \nu_{i+1}) \end{array}\right]. \tag{25}$$

Consider the decomposition done in Eq. (12). The first two summands of the right-hand side can be respectively upper bounded by Eqs. (24) and (25) since these bounds hold for all $\mathsf{rnd}' \in \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big)$ and $(\mathbb{x}', \boldsymbol{\alpha}', \boldsymbol{\sigma}') \in T$. Moreover, as explained above, an upper bound for the last summand is indirectly given in

Corollary 6.18. Therefore, it follows that

$$
\begin{aligned}
&\mathbb{E}\left[I_i(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \right] \\
&= \mathbb{E}\left[I_i^{\top}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \right] \\
&\quad + \mathbb{E}\left[I_i^{\perp}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \right] \\
&\quad + \mathbb{E}\left[\Lambda_i'(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \right] \\
&\leq a_{i+1} \cdot \Pr\left[T \neq \perp \;\middle|\; \begin{array}{ccc} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}) \end{array} \right] \\
&\quad \cdot \mathbb{E}\left[\begin{array}{l} I_{i+1}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_{i+1}) \\ \text{conditioned on} \\ T \neq \perp \end{array} \;\middle|\; \begin{array}{ccc} \nu_{i+1} & \leftarrow & \mathcal{N}_{i+1} \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu_{i+1}) \end{array} \right] \\
&\quad + a_{i+1} \cdot \Pr\left[T = \perp \;\middle|\; \begin{array}{ccc} \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}) \end{array} \right] \\
&\quad \cdot \mathbb{E}\left[\begin{array}{l} I_{i+1}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_{i+1}) \\ \text{conditioned on} \\ T = \perp \end{array} \;\middle|\; \begin{array}{ccc} \nu_{i+1} & \leftarrow & \mathcal{N}_{i+1} \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ T & \leftarrow & \mathsf{TreeFinder}_{i+1}(\mathbb{x}, \\ & & \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}; \nu_{i+1}) \end{array} \right] \\
&\quad + (a_{i+1} - 1) \cdot (t+1) \\
&= a_{i+1} \cdot \mathbb{E}\left[I_{i+1}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_{i+1}) \;\middle|\; \begin{array}{ccc} \nu_{i+1} & \leftarrow & \mathcal{N}_{i+1} \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \right] \\
&\quad + (a_{i+1} - 1) \cdot (t+1) \;\; ,
\end{aligned}
$$

which concludes the proof. □

We can prove the following theorem now that we have a recursive bound for I_i.

Theorem 6.20. *For every $i \in \{0, \dots, \mathsf{k}-1\}$, the following holds*

$$\mathbb{E}\left[I_i(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \right]$$
$$\leq \left(\prod_{j=i+1}^{\mathsf{k}} a_j\right) + (t+1)\cdot\left(\left(\prod_{j=i+1}^{\mathsf{k}} a_j\right) - 1\right) \ .$$

Proof. We prove this statement by induction on i going from k to 0. The base case follows from the fact that $\mathsf{TreeFinder}_{\mathsf{k}}$ does zero invocation of the state restoration game. Indeed, its only job is to run the verifier to obtain the success bit. Therefore, the number of invocations is smaller than one, which is the value of the Theorem 6.20's right-hand side when $i = \mathsf{k}$.

For the inductive step, we have by Lemma 6.19 and then by the induction hypothesis

$$\mathbb{E}\left[I_i(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_i) \;\middle|\; \begin{array}{ccc} \nu_i & \leftarrow & \mathcal{N}_i \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \right]$$
$$\leq a_{i+1}\cdot\mathbb{E}\left[I_{i+1}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_{i+1}) \;\middle|\; \begin{array}{ccc} \nu_{i+1} & \leftarrow & \mathcal{N}_{i+1} \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \right]$$
$$+ (t+1)\cdot(a_{i+1}-1)$$
$$\leq a_{i+1}\left(\left(\prod_{j=i+2}^{\mathsf{k}} a_j\right) + (t+1)\cdot\left(\left(\prod_{j=i+2}^{\mathsf{k}} a_j\right) - 1\right)\right) + (t+1)\cdot(a_{i+1}-1)$$
$$= \left(\prod_{j=i+1}^{\mathsf{k}} a_j\right) + (t+1)\cdot\left(\left(\prod_{j=i+1}^{\mathsf{k}} a_j\right) - 1\right) \ ,$$

which concludes the proof of this theorem. □

Inserting Construction 6.2 in Definition 2.38 probability experiment, we have

$$\mathbb{E}\left[I_0(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, (\dot{\mathsf{rnd}}_i)_{i\in[\mathsf{k}]}, \nu_0) \;\middle|\; \begin{array}{ccc} \nu_0 & \leftarrow & \mathcal{N}_0 \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \\ \dot{\mathsf{rnd}}_1 & \leftarrow & \mathcal{U}(\mathsf{r}_1)|_{\mathsf{tr}_1^{\mathsf{sr}}} \\ \vdots & & \\ \dot{\mathsf{rnd}}_{\mathsf{k}} & \leftarrow & \mathcal{U}(\mathsf{r}_{\mathsf{k}})|_{\mathsf{tr}_{\mathsf{k}}^{\mathsf{sr}}} \end{array} \right]$$
$$= \Pr\left[I_0(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}, \mathbf{rnd}, \nu_0) \;\middle|\; \begin{array}{ccc} \nu_0 & \leftarrow & \mathcal{N}_0 \\ \mathbf{rnd} & \leftarrow & \mathcal{U}\big((\mathsf{r}_i)_{i\in[\mathsf{k}]}\big) \\ (\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\sigma}, \boldsymbol{\rho}) & \xleftarrow{\mathbf{tr}^{\mathsf{sr}}} & \Gamma^{\mathsf{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}) \end{array} \right]$$
$$\leq \left(\prod_{j=i+1}^{\mathsf{k}} a_j\right) + (t+1)\cdot\left(\left(\prod_{j=i+1}^{\mathsf{k}} a_j\right) - 1\right) \ ,$$

by Theorem 6.20. The first equality comes from their two experiments being indistinguishable since the random oracles of the right-hand side are restricted to match the traces of the others. Finally, every time the extractor executes the state restoration game with $\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}$, it also runs some polynomial time computation. Therefore, its total extraction time is

$$\left(\left(\prod_{j=1}^{\mathsf{k}} a_j\right) + (t+1)\cdot\left(\left(\prod_{j=1}^{\mathsf{k}} a_j\right) - 1\right)\right)\cdot(\tau_{\tilde{\mathbf{P}}^{\mathsf{sr}}_{\mathsf{IP}}} + \mathsf{poly}(n)) \ .$$

References

[ACK21] Thomas Attema, Ronald Cramer, and Lisa Kohl. "A Compressed Sigma-Protocol Theory for Lattices". In: *Advances in Cryptology – CRYPTO 2021*. Ed. by Tal Malkin and Chris Peikert. Cham: Springer International Publishing, 2021, pp. 549–579. ISBN: 978-3-030-84245-1.

[AFK22] Thomas Attema, Serge Fehr, and Michael Klooß. "Fiat-Shamir Transformation of Multi-round Interactive Proofs". In: *Theory of Cryptography*. Ed. by Eike Kiltz and Vinod Vaikuntanathan. Cham: Springer Nature Switzerland, 2022, pp. 113–142.

[AFR23] Thomas Attema, Serge Fehr, and Nicolas Resch. "Generalized Special-Sound Interactive Proofs andăTheir Knowledge Soundness". In: *Theory of Cryptography*. Ed. by Guy Rothblum and Hoeteck Wee. Cham: Springer Nature Switzerland, 2023, pp. 424–454. ISBN: 978-3-031-48621-0.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive Oracle Proofs". In: *Proceedings of the 14th Theory of Cryptography Conference*. TCC '16-B. 2016, pp. 31–60.

[BDFLSZ11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. "Random Oracles in a Quantum World". In: *Proceedings of the 17th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '11. 2011, pp. 41–69.

[CMS19] Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. "Succinct Arguments in the Quantum Random Oracle Model". In: *Proceedings of the 17th Theory of Cryptography Conference*. TCC '19. 2019, pp. 1–29.

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

[DFMS19] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. *Security of the Fiat-Shamir Transformation in the Quantum Random-Oracle Model*. Cryptology ePrint Archive, Report 2019/190. 2019.

[DH76] W. Diffie and M.E. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

[EG85] Taher El Gamal. "A public key cryptosystem and a signature scheme based on discrete logarithms". In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472.

[FS86] Amos Fiat and Adi Shamir. "How to prove yourself: practical solutions to identification and signature problems". In: *Proceedings of the 6th Annual International Cryptology Conference*. CRYPTO '86. 1986, pp. 186–194.

[LZ19] Qipeng Liu and Mark Zhandry. *Revisiting Post-Quantum Fiat-Shamir*. Cryptology ePrint Archive, Report 2019/262. 2019.

[Mic00] Silvio Micali. "Computationally Sound Proofs". In: *SIAM Journal on Computing* 30.4 (2000). Preliminary version appeared in FOCS '94., pp. 1253–1298.

[Sch90] C. P. Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Ed. by Gilles Brassard. New York, NY: Springer New York, 1990, pp. 239–252.

[YZ22] Takashi Yamakawa and Mark Zhandry. "Verifiable Quantum Advantage without Structure". In: *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. 2022, pp. 69–74. DOI: 10.1109/FOCS54457.2022.00014.

A Relation between strong and weak knowledge soundness

We prove Theorem 2.32 in this appendix.

Proof. Let $\mathsf{IP} = (\mathbf{P}_{\mathsf{IP}}, \mathbf{V}_{\mathsf{IP}})$ be a public-coin interactive with strong knowledge soundness error κ_{IP} and extraction time $\mathsf{et}_{\mathsf{IP}}$. Let $\mathbf{E}_{\mathsf{IP}}^{\mathsf{SKS}}$ be a strong knowledge soundness extractor for IP. We construct a weak knowledge soundness extractor $\mathbf{E}_{\mathsf{IP}}^{\mathsf{KS}}$, which achieves the desired properties as follows:

$\mathbf{E}_{\mathsf{IP}}^{\mathsf{KS}}(\mathbb{x}, \tilde{\mathbf{P}}_{\mathsf{IP}})$:
1. Pick verifier randomness $\boldsymbol{\rho} \leftarrow \mathcal{R}$.
2. Run $\tilde{\mathbf{P}}_{\mathsf{IP}}$: $\boldsymbol{\alpha} \leftarrow \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\rho})$.
3. Return $\mathbf{E}_{\mathsf{IP}}^{\mathsf{SKS}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \tilde{\mathbf{P}}_{\mathsf{IP}})$.

Let

$$E(\mathbb{x}, \boldsymbol{\rho}, \tilde{\mathbf{P}}_{\mathsf{IP}}) \coloneqq \left\{ b = 1 \;\middle|\; b \xleftarrow{\boldsymbol{\alpha}, \boldsymbol{\rho}} \langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}), \mathbf{V}_{\mathsf{IP}}(\mathbb{x}; \boldsymbol{\rho}) \rangle_{\mathsf{IP}} \right\}$$

be the event that the $\mathbf{V}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\rho})$ accepts against $\tilde{\mathbf{P}}_{\mathsf{IP}}$ on the instance $\mathbb{x}$ and $\epsilon(\mathbb{x}, \tilde{\mathbf{P}}_{\mathsf{IP}})$ be the fraction of $\boldsymbol{\rho}$ in $\boldsymbol{\mathcal{R}}$ such that $\langle \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}), \mathbf{V}_{\mathsf{IP}}(\mathbb{x}; \boldsymbol{\rho}) \rangle_{\mathsf{IP}} = 1$. The weak knowledge soundness error $\mathbf{E}_{\mathsf{IP}}^{\mathsf{KS}}$ is

$$\begin{aligned}
&\Pr\left[(\mathbb{x}, \mathbb{w}) \in R \;\middle|\; \mathbb{w} \leftarrow \mathbf{E}_{\mathsf{IP}}^{\mathsf{KS}}(\mathbb{x}, \tilde{\mathbf{P}}_{\mathsf{IP}})\right] \\
&= \Pr\left[(\mathbb{x}, \mathbb{w}) \in R \;\middle|\; \begin{array}{rcl} \boldsymbol{\rho} & \leftarrow & \mathcal{R} \\ \boldsymbol{\alpha} & \leftarrow & \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\rho}) \\ \mathbb{w} & \leftarrow & \mathbf{E}_{\mathsf{IP}}^{\mathsf{SKS}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \tilde{\mathbf{P}}_{\mathsf{SP}}) \end{array}\right] \\
&= 1 - \Pr\left[(\mathbb{x}, \mathbb{w}) \notin R \;\middle|\; \begin{array}{rcl} \boldsymbol{\rho} & \leftarrow & \mathcal{R} \\ \boldsymbol{\alpha} & \leftarrow & \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\rho}) \\ \mathbb{w} & \leftarrow & \mathbf{E}_{\mathsf{IP}}^{\mathsf{SKS}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \tilde{\mathbf{P}}_{\mathsf{SP}}) \end{array}\right] \\
&= 1 - \underbrace{\Pr\left[\begin{array}{l} (\mathbb{x}, \mathbb{w}) \notin R \\ \wedge\, b = 1 \end{array} \;\middle|\; \begin{array}{rcl} \boldsymbol{\rho} & \leftarrow & \mathcal{R} \\ \boldsymbol{\alpha} & \leftarrow & \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\rho}) \\ b & \leftarrow & \mathbf{V}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\rho}) \\ \mathbb{w} & \leftarrow & \mathbf{E}_{\mathsf{IP}}^{\mathsf{SKS}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \tilde{\mathbf{P}}_{\mathsf{SP}}) \end{array}\right]}_{\leq \kappa_{\mathsf{IP}}} \\
&\quad - \underbrace{\Pr\left[\begin{array}{l} (\mathbb{x}, \mathbb{w}) \notin R \\ \wedge\, b = 0 \end{array} \;\middle|\; \begin{array}{rcl} \boldsymbol{\rho} & \leftarrow & \mathcal{R} \\ \boldsymbol{\alpha} & \leftarrow & \tilde{\mathbf{P}}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\rho}) \\ b & \leftarrow & \mathbf{V}_{\mathsf{IP}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\rho}) \\ \mathbb{w} & \leftarrow & \mathbf{E}_{\mathsf{IP}}^{\mathsf{SKS}}(\mathbb{x}, \boldsymbol{\alpha}, \boldsymbol{\rho}, \tilde{\mathbf{P}}_{\mathsf{SP}}) \end{array}\right]}_{\leq \Pr\left[\neg E(\mathbb{x}, \boldsymbol{\rho}, \tilde{\mathbf{P}}_{\mathsf{IP}})\right] = 1 - \epsilon(\mathbb{x}, \tilde{\mathbf{P}}_{\mathsf{IP}})} \\
&\geq 1 - \kappa_{\mathsf{IP}} - (1 - \epsilon(\mathbb{x}, \tilde{\mathbf{P}}_{\mathsf{IP}})) \\
&\geq \epsilon(\mathbb{x}, \tilde{\mathbf{P}}_{\mathsf{IP}}) - \kappa_{\mathsf{IP}}
\end{aligned}$$

as desired. The additional running of $\mathbf{E}_{\mathsf{SP}}^{\mathsf{KS}}$ compared to $\mathbf{E}_{\mathsf{IP}}^{\mathsf{SKS}}$ is $\Theta(\mathsf{k})$ for sampling the challenges and $\Theta(\tau_{\tilde{\mathbf{P}}_{\mathsf{IP}}})$ for collecting the responses. Therefore, the slight increase in time complexity leads to

$$\mathsf{et}'_{\mathsf{IP}}(\mathbb{x}, \tilde{\mathbf{P}}_{\mathsf{IP}}) = \Theta(\mathsf{k} + \tau_{\tilde{\mathbf{P}}_{\mathsf{IP}}}) + \mathsf{et}_{\mathsf{IP}}(\mathbb{x}, \tilde{\mathbf{P}}_{\mathsf{IP}}) \ .$$

□