

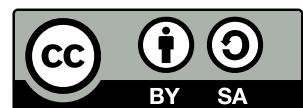
Building Cryptographic Proofs from Hash Functions

Alessandro Chiesa

Eylon Yogev

August 22, 2024

This work is licensed under a Creative Commons “Attribution-ShareAlike 4.0 International” license.



Contents

Foreword	v
Preface	vii
1 Mathematical background	1

I What are Cryptographic Proofs?

2 The random oracle model	14
3 Basic cryptographic properties	29
4 Arguments in the ROM	35
5 Additional security definitions	42
6 Basic observations about arguments	51
7 Arguments with multiple random oracles	66

II NARGs Based on SPs

8 Sigma protocols	74
9 Warmup: non-adaptive security	79
10 The Fiat–Shamir transformation for SPs	90
11 Additional security definitions	94
12 State restoration	99

III NARGs Based on IPs

13 Interactive proofs	111
14 The Fiat–Shamir transformation for IPs	121
15 Optimization: a faster variant	131
16 Additional security definitions	144

IV Commitment Schemes

17 Basic commitment scheme	153
18 Merkle commitment scheme	166

V SNARGs Based on PCPs

19 Probabilistically checkable proofs	199
20 Warmup: succinct interactive arguments from PCPs	207
21 The Micali transformation	216
22 Additional security definitions	225

VI SNARGs Based on IOPs

23 Interactive oracle proofs	233
24 Warmup: succinct interactive arguments from IOPs	239
25 The BCS transformation	248
26 Additional security definitions	261

VII Practical Considerations

27 Constructions of probabilistic proofs	275
28 Setting parameters	280
29 Merkle commitment scheme optimizations	291
30 Special soundness	308
31 Round-by-round soundness	351
 Error bounds	360
List of figures	363
List of tables	365
Glossary	366
Bibliographic notes	369
Bibliography	373

Foreword

Succinct proof systems are a remarkable tool. Let us briefly explain what they are with a simple example. Suppose that Alice has a public program P that does something interesting. Say it outputs the decimal digits of π one after another, first 3, then 1, then 4, and so on. Alice runs the program on her laptop, and the program outputs the millionth digit of π , which happens to be 5. She is so excited that she tells all her friends about it. One of Alice's friends, Bob, is a bit suspicious. Bob inspects the code of the program P and confirms that it indeed computes the digits of π correctly. But he worries that Alice simply guessed the millionth digit without actually running the program. Bob decides to check the computation for himself. He sets his laptop to run P and waits for it to reconfirm Alice's discovery.

While Bob waits for his laptop to rerun the entire computation from scratch, he starts to think that this makes no sense. What if their friend Carol does not trust either of them? She will also have to rerun the entire computation to convince herself that P ran correctly. What if David does not trust all three of them? He will also have to rerun the entire computation. This is a lot of wasted effort. All of them are redoing the same computation over and over.

Is there a better way? A succinct proof system gives a remarkable solution. It lets Alice compute a proof that the program P ran correctly and that its output is as claimed. The proof is *succinct*, meaning that it is short and fast to verify. In practice, a succinct proof is only a few kilobytes long and takes only a few milliseconds to verify. One should pause and marvel at this: whether Alice is computing the millionth or billionth digit of π , the proof that she computed the correct value is always only a few kilobytes and only takes a few milliseconds to verify.

Remarkably, this applies to any program P , even one that takes an auxiliary input from Alice. Alice can run the program and publish its output along with a succinct proof that it ran correctly. Then, with a few milliseconds of work, Bob, Carol, David, and anyone else can check the proof and be convinced that Alice ran the program correctly. There is no need for them to rerun the computation. The succinct proof can even be made zero knowledge, meaning that the proof reveals nothing about Alice's auxiliary input, and yet it convinces everyone that the program ran correctly on that input.

In recent years, the study of succinct proof systems has grown into a vast area of research with many beautiful ideas contributed by many researchers. One reason for this rapid progress is the commercial applications of these tools, for example, in scaling decentralized systems. Another reason is the fertile ground for new technical innovations. There are multiple approaches for constructing succinct proofs, giving researchers many directions to explore. There is also a strong need for engineers to build programming frameworks that make it easy for non-experts to use these tools. Overall, there is a vibrant community of researchers and developers that make this an exciting and fun area to work in.

This book covers a particular approach to succinct proof systems called hash-based proofs. These are conceptually the simplest proof systems available. They require no fancy mathematics, which makes them accessible to a broad set of readers. In addition, the security of these proof systems depends on relatively simple properties of the hash function. For standard cryptographic hash functions, these properties are believed to hold even against an adversary who has access

to a large-scale quantum computer. As such, these proof systems are post-quantum; they will remain secure even after a large-scale quantum computer is constructed.

The theory behind hash-based proof systems is well-understood and fairly mature. Yet, until now, no single reference has provided all the details. This is what this book is about. The book begins with precise definitions of the elements that make up a succinct proof system. Next, the book turns to building succinct proof systems from several abstract information theoretic objects. One example, that is widely used in practice, builds a succinct proof system from an important object called an Interactive Oracle Proof, or IOP. The construction relies purely on hash functions. The book slowly builds the necessary tools, and then gives a clear description of the construction and its proof of security.

This book is a significant contribution to the area of proof systems. It is meant for readers who enjoy a clear yet precise treatment of the material. I have no doubt that once you read this book you will want to learn more about the area. This book is a great starting point for a fun journey into the world of succinct proofs.

Dan Boneh
April 2024
Stanford University

Preface

“Proofs are fundamental to our lives, and as for all things fundamental we should expect that answering the question of what a proof is will always be an on going process.”

Silvio Micali, *Computationally Sound Proofs*

This book is about **cryptographic proofs**, which are protocols used to prove and to verify the correct execution of computations, succinctly and in zero knowledge (more about this soon). Cryptographic proofs are a fundamental object in computer science that lies at the intersection of cryptography and computational complexity. They have received tremendous interest from academia and industry, catalyzed by real-world applications. Cryptographic proofs, which draw upon some of the most beautiful ideas and powerful tools in the theory of computing, are a remarkable example of how deep mathematical ideas can play a key enabling role in new technologies.

This book provides a rigorous introduction to cryptographic proofs built from (ideal) hash functions, and serves as a self-study for students and a reference for practitioners. This includes notable constructions of succinct non-interactive arguments (SNARGs) from (ideal) hash functions. For example, STARKs (scalable transparent arguments of knowledge) are an example of such SNARGs.

Latest version The latest version of this book can be found at this website:

<https://hash-based-snargs-book.github.io/>

The website additionally links to the book’s source code in a Git repository, which makes explicit the latest corrections and additions. We would be delighted to receive comments (positive or negative!) on this book, as well as any corrections or suggestions. You can directly submit issues or pull requests to the repository on GitHub, or simply email us directly.

License The source code of the book (and the book itself) is licensed under the [Creative Commons Attribution-ShareAlike 4.0 International License](#) (CC BY-SA 4.0). Briefly, you are allowed to share and adapt the source code of this book, provided you give appropriate credit and indicate any changes; moreover, material derived from this book must carry the same license (or one compatible with it). See <https://creativecommons.org/licenses/by-sa/4.0/> for more on this license.

Motivation

The re-execution problem A basic goal in computer science is *computation integrity*:

How can one party check that another party executed a given computation correctly?

A natural approach to achieve this goal is to **re-execute the computation**, and check if this leads to the same claimed result.

Consider, for example, the illustrative computational task of counting the number of primes up to 1 billion. Alice determines via a prime-counting algorithm (e.g., the sieve of Eratosthenes) that the answer is 50,847,534, and publishes the computational claim “the number of primes up to 1 billion is 50,847,534”. How can Bob check that Alice’s computational claim is correct? One option is for Bob to re-run the algorithm that Alice used (or some other prime-counting algorithm) and check if the number of primes up to 1 billion really is 50,847,534.

However, in many settings re-execution is *not an option*.

- A party may not be able to afford (or may not have enough resources) to re-execute the computation associated to a computational claim. For instance, computing the number of primes up to 10^{29} using a state-of-the-art prime-counting algorithm in 2022 took 90+ core years and a peak RAM usage of over 1 terabyte;¹ such a computation is not easily re-executed.
- Moreover, the computation may involve private inputs that cannot be shared with others so that they can re-execute the computation. Here is an example that involves RSA key pairs. Recall that an RSA key pair consists of a private key that contains two prime numbers (sampled from a suitable distribution) and a public key that contains their product; such key pairs are used for encryption schemes and digital signature schemes (whose hardness is based on the RSA assumption). Consider the claim “I know two primes whose product is the RSA public key N ”. Revealing the two primes enables others to cheaply verify the claim. However, doing so would give away the secret key corresponding to the public key.

We are then faced with the re-execution problem:

How to ensure computation integrity without re-execution?

This seems impossible at first glance. Hierarchy theorems in computational complexity state that, in general, there is no way to reduce the time or space used to determine the output of a computation.² Ensuring computation integrity without re-execution seems like an unattainable goal.

Avoiding re-execution Amazingly, it is possible to avoid re-execution, by requiring the party making a computational claim to also produce a *cryptographic proof* of computational integrity for the claim. (In particular, this circumvents the aforementioned limitations, and others that we do not mention here.) Informally, a party can certify a computation’s correctness by producing a cryptographic proof via a *prover algorithm*. Subsequently, anyone can check the cryptographic proof via a *verification algorithm*, instead of re-executing the original computation. Security holds against adversaries whose resources are suitably bounded (more on this later).

But how is verification of the cryptographic proof “better” than re-execution? This is due to remarkable properties that cryptographic proofs can achieve.

¹See <https://oeis.org/A006880> and <https://www.mersenneforum.org/showthread.php?t=20473>.

²For example, the hierarchy theorem for deterministic multi-tape Turing machines states that, for every time-constructible function $t(n)$, the complexity class $\text{DTIME}(t(n))$ is strictly larger than the complexity class $\text{DTIME}(o(\frac{t(n)}{\log t(n)}))$. In other words, there are computational tasks whose output can be determined in time $t(n)$ but not in time $o(\frac{t(n)}{\log t(n)})$. Other hierarchy theorems are known for other resources and models of computation.

- *Succinctness.* The verification algorithm can be *exponentially* faster than re-executing the original computation. This property, known as succinctness, enables computationally weak parties to verify the correctness of computationally expensive computations.
- *Zero knowledge.* The cryptographic proof reveals *no information* about any private inputs used in the computation. This property, known as zero knowledge, enables others to check the correctness of computations that involve secrets. (Secrets that were known only to the party that performed the computation and produced a cryptographic proof for it.)

A cryptographic proof that satisfies (some form of) succinctness is typically known as a *succinct argument*, or *SNARG* if it is non-interactive. A popular notion is that of a zkSNARK which is a *zero-knowledge SNARG of knowledge*. These notions will be introduced and defined in this book.

Applications Cryptographic proofs are commonly used in academic research, in the design of cryptographic primitives, protocols, and systems. Perhaps more exciting is the fact that cryptographic proofs have found applications well beyond academia: they have been widely deployed in the real world, in numerous settings where re-execution is not an option. At the time of writing they are one of the hottest topics in applied cryptography.

For example, cryptographic proofs play a key role in the design of secure distributed systems, such as a blockchains. They are used to achieve scalability and privacy goals, as sketched next.

- *Scalability.* The succinctness property enables increasing the transaction throughput in peer-to-peer networks, by moving expensive computation away from the network nodes.

Computation in smart contract systems (e.g., Ethereum) is slow and expensive: each computation step is re-executed by every node in the network, so the network is bottlenecked by the slowest node and users are charged a high price for each computation step.

A class of architectures known as “proof-based rollups” moves computation off-chain, in the sense that users send their computation requests to an aggregator who then periodically produces a cryptographic proof about batches of user computations; the smart contract system then verifies the cryptographic proof and authorizes a state transition reflecting all the computations in the batch. The succinctness property ensures that checking a cryptographic proof is exponentially cheaper than checking the computation that it attests to. Hence having every node in the network merely check the cryptographic proof is much cheaper, and enables the system to scale to more users and bigger computations.

- *Privacy.* The zero-knowledge property allows designing peer-to-peer networks with strong user privacy. Informally, the zero-knowledge property enables a party in the network to broadcast a cryptographic proof about a computation that involves its own private inputs. For instance, the computation may authorize the transfer of digital assets from one party to another party but without revealing the identity of these parties or the types and amounts of digital assets. Such privacy-preserving capabilities are essential to cryptocurrencies and other decentralized finance applications, because transactions often involve highly sensitive information that should not be disclosed to the whole network.

These and other applications have generated much interest in cryptographic proofs across several communities in academia, industry, and government. This broader interest demands material about cryptographic proofs that is more accessible and systematized, in order to effectively serve the needs of an audience beyond a few experts.

Scope

Cryptographic proofs are typically constructed from two building blocks: an information-theoretic probabilistic proof, and a cryptographic transformation that maps the probabilistic proof to a cryptographic proof. There are many choices for these building blocks, leading to many cryptographic proofs with different properties. This leads to a vast landscape of cryptographic proofs whose discussion is not our focus. Here we aim for depth rather than breadth: this book covers in detail a notable class of cryptographic proofs, omitting discussion of other cryptographic proofs.

Specifically, this book provides a comprehensive and rigorous treatment of **cryptographic proofs based on ideal hash functions**. We cover fundamental constructions, providing full security proofs for several relevant security properties and describing useful optimizations. Security reductions have explicit error bounds, which enables setting security parameters in practice. In most cases our analyses are essentially tight, and we improve upon the fragmented and incomplete treatment of this material that exists in the literature. We adopt uniform terminology and notation throughout the book to highlight the relationships between the different constructions that we cover.

The “pure” random oracle setting We consider a simple information-theoretic setting: constructing cryptographic proofs from a hash function that is modeled as *ideal*. This means that the hash function is modeled as a truly random function (different inputs lead to independent random outputs), and every party is granted query access to this same *random oracle*.³ This setting is known as the *random oracle model* (ROM), and is extensively used in cryptography.

In fact, we focus on the “pure” ROM, where cryptographic proofs are constructed given and only given a random oracle (no other cryptographic primitives or models are allowed). This setting can be viewed as one of information-theoretic security, as adversaries can be computationally unbounded, with the only restriction being that they can query the random oracle a bounded number of times.

All constructions in this book are *transformations* from some type of probabilistic proof to some type of (interactive or non-interactive) cryptographic proof in the ROM. The design and analysis of suitable probabilistic proofs is not a goal of this book (it demands considerable background in computational complexity and abstract algebra). Therefore, probabilistic proofs in this book are taken as a starting point towards the construction of cryptographic proofs in the ROM.

Why this scope? The selection of material for this book has several benefits.

- *Simplicity.* Hash functions are one of the most basic tools in cryptography, especially so when modeled as random oracles. Designing and analyzing constructions in the ROM typically involves only elementary definitions and discrete probability. This is in contrast to other tools often used to construct cryptographic proofs (such as linear-only encodings, pairings, scalar-product arguments, polynomial commitments, and others), which demand considerable background in cryptography, computational complexity, and abstract algebra. Thus *cryptographic proofs in the ROM are among the simplest known constructions*.

³This is in contrast to the case where the underlying hash function satisfies a concrete security property, such as collision resistance. Succinct non-interactive arguments are not known to exist when given only this cryptographic primitive.

- *Stability.* The landscape of cryptographic proofs is fast-changing due to much active research and remains, as a whole, not ready for a comprehensive systematization. On the other hand, cryptographic proofs in the ROM are a class of constructions that are very well understood and are unlikely to see changes that significantly diverge from current constructions.
- *Security.* Much of cryptography relies on hard problems borrowed from branches of mathematics such as number theory and algebraic groups. The structure of these problems, while useful for cryptography, may be exploited by future algorithms that make these problems less hard. In contrast, cryptography in the ROM relies only on a random function (which by definition has no structure) and nothing else. While the ROM is in a precise theoretical sense a strong model, in practice cryptography in the ROM seems to offer more robust long-term security. Indeed, in practice the random oracle is heuristically instantiated via a suitable hash function with no “useful” structure (e.g., SHA256). These considerations continue to hold even in the presence of quantum computers (for which many “hard” problems in cryptography are easy), and indeed cryptographic proofs in the ROM are at present one of the most efficient approaches to post-quantum cryptographic proofs. Overall, cryptographic proofs in the ROM are a technology that is likely to remain relevant for a long time.
- *Transparent setup.* The public parameters of a cryptographic proof based on a hash function (modeled as ideal for analysis) is the specification of the hash function itself. Since making the choice of hash function does not involve secret randomness, the cryptographic proof is said to have a *transparent setup* (or *public-coin setup*). Cryptographic proofs with a transparent setup are particularly convenient to deploy because there is no need for a trusted party (or multi-party cryptographic ceremony) to securely sample the public parameters — all that is needed is agreeing on the choice of hash function. For example, STARKs (scalable transparent arguments of knowledge) are a notable class of cryptographic proofs in the ROM, whose transparent setup has facilitated adoption.

Goal The current academic literature provides a fragmented and, regrettably, incomplete collection of constructions and results about cryptographic proofs in the ROM. The systematic and in-depth treatment of this book provides an auditable resource that details security definitions, security proofs, and optimizations that are relevant to practical use in real-world (and high-stakes) systems. This resource will enable the community to ensure the cryptographic security of implemented systems, and ultimately also inspire greater confidence in this new technology.

Complementary resources We stated that there is a vast landscape of material about cryptographic proofs, and that this book covers only a certain part of it. We mention several notable didactic resources that can help the reader learn about other parts of this landscape.

Alessandro Chiesa teaches a graduate course on probabilistic proofs; lectures and exercises based on his course are available on the websites of two summer schools [CG21; Chi23]. This complements the constructions described in this book, where probabilistic proofs are taken as a starting point.

The ZKProof Community Reference [ZKP22] surveys modern constructions of zero-knowledge proofs; this includes discussions of useful definitions about cryptographic proofs in various

settings and some constructions of succinct cryptographic proofs.

A book by Justin Thaler [Tha22] surveys different types of cryptographic proofs, commenting on their tradeoffs. An online course surveying cryptographic proofs, including material about practical concerns, is available at <https://zk-learning.org/>.

Structure of the book

Required background While understanding many constructions of succinct arguments requires background across several areas of computer science and mathematics (such as cryptography, computational complexity, coding theory, and abstract algebra), our choice to focus on succinct arguments constructed exclusively from random oracles significantly reduces the required background. *We assess that the background necessary to understand the material in this book consists of undergraduate-level discrete probability and algorithms.* In Chapter 1, which is a preliminary chapter on mathematical background, we provide all the basic facts used in this book. (If, while reading this book, you spot background material that is not included in Chapter 1, please kindly let us know!) In particular, a motivated reader who is familiar with the material in Chapter 1 should be able to fully understand the entire book without relying on additional references.

Structure Chapter 1 establishes basic notation and mathematical background for the rest of the book. We encourage the reader to skim through this chapter to determine if this basic material is familiar. All other chapters are divided into seven parts, which cover the following topics.

- *Part I: What are Cryptographic Proofs?*

This part introduces the random oracle model and the notion of cryptographic proofs, more formally known as *arguments*. Definitions for arguments include crucial properties such as adaptive security (for soundness, knowledge soundness, and zero knowledge), often ignored in academic papers. We recommend starting with this part, as all other parts depend on material in it.

- *Part II: NARGs Based on SPs*

This part describes how to transform a sigma protocol (a basic type of probabilistic proof) into a non-interactive argument. This relies on the **Fiat–Shamir transformation**, which uses the random oracle to “remove interaction” from the sigma protocol.

- *Part III: NARGs Based on IPs*

This part describes how to transform an interactive proof (a multi-round protocol between a prover and a verifier) into a non-interactive argument. This requires extending the ideas of Part II to multiple rounds, and should be read after understanding the material of Part II. A delicate point for this **multi-round Fiat–Shamir transformation** is to understand the soundness properties of the interactive proof that ensure security of the transformation.

- *Part IV: Commitment Schemes*

This part studies two commitment schemes in the random oracle model: a basic commitment scheme, and the **Merkle commitment scheme**. This part can be read independently:

these commitment schemes can be understood given only the basic definitions and properties of a random oracle. (In particular, no material on arguments is necessary.) Commitment schemes play a role in the construction of succinct arguments, which are studied in Parts V and VI.

- *Part V: SNARGs Based on PCPs*

This part describes how to construct succinct arguments from probabilistically checkable proofs, a type of probabilistic proof where a verifier reads few queries from a long proof string. This includes a construction that is interactive (known as the **Kilian transformation**), and then, building on this, a construction that is non-interactive (known as the **Micali transformation**). A key ingredient is the Merkle commitment scheme, studied in Part IV.

- *Part VI: SNARGs Based on IOPs*

This part describes how to construct succinct arguments from interactive oracle proofs, a type of probabilistic proof that simultaneously generalizes interactive proofs and probabilistically checkable proofs. Similarly to Part V, the part includes a simpler interactive argument and then, building on this, a non-interactive argument (known as the **Ben-Sasson–Chiesa–Spooncer transformation**, or **BCS transformation** for short). Merkle commitment schemes again play a key role. Interactive oracle proofs have notable applications in practice, via constructions of succinct arguments described in this part.

- *Part VII: Practical Considerations*

This part includes a selection of topics relevant to using arguments in practice. These include discussions on how to set parameters to ensure concrete security, optimizations, and strong soundness properties of probabilistic proofs. This part does not have to be read in order.

We summarize the dependencies between parts in Figure 1 and the transformations that we study in Figure 2.

How to use this book

Audience The book is aimed at people interested in understanding foundations and constructions of succinct arguments. The limited required background will make the material of this book accessible to different types of audiences.

- *Students.* An increasing number of undergraduate and graduate students wish to conduct research in cryptographic proofs, and SNARGs in particular. The material in this book enables students to gain familiarity with the main properties of argument systems in the simple ROM setting, and study how to write security reductions that transform a prover against an argument system into a prover against the underlying probabilistic proof.
- *Practitioners.* Explicit and detailed constructions of several cryptographic proofs, presented with uniform notation and language, helps practitioners audit implementations. Moreover, the concrete security bounds helps practitioners set security parameters of implementations. This book includes several examples of how to make such choices.

Educators will also benefit from this book, as we outline below via examples of courses where material of this book will be useful.

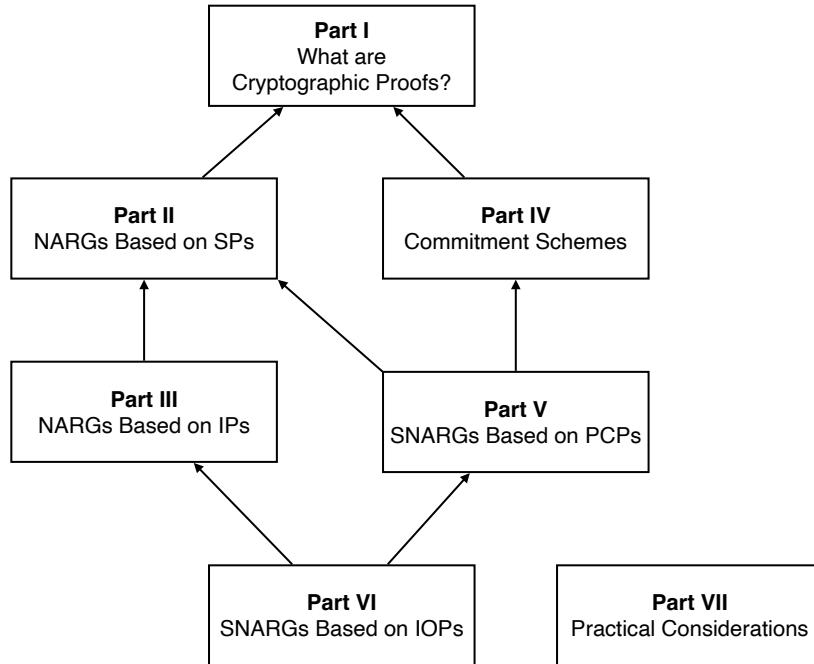


Figure 1: Dependencies between the different parts of this book. (NARG stands for *non-interactive argument*, and SNARG stands for *succinct non-interactive argument*.)

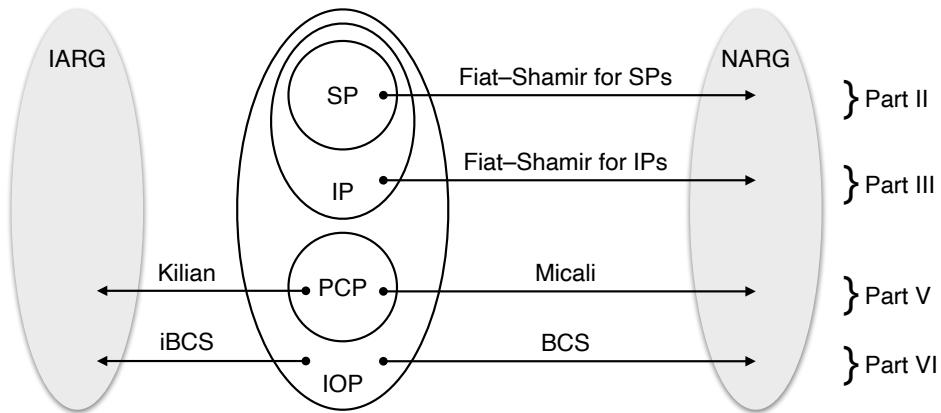


Figure 2: All constructions of cryptographic proofs in this book are transformations applied to some type of probabilistic proof (SP, IP, PCP, or IOP). This diagram lists the transformations that we study, showing which ones construct an interactive argument (IARG) and which ones construct a non-interactive argument (NARG).

Usage The book can be used as a self-contained reference that a self-learner (e.g., a student or practitioner interested in the area) can follow chapter by chapter. For this it is useful to keep in mind the dependencies between parts of the book shown in Figure 1.

Moreover, educators (e.g., a university professor) will find this book to be a useful didactic resource for certain courses. We give two examples.

- *Introductory cryptography.* An introductory cryptography course can benefit from discussions on the security definitions for argument systems in Part I, and from some of the simpler constructions of arguments. For example, the use of random oracles in Part II to remove interaction is a fundamental cryptographic paradigm known as the Fiat–Shamir transformation. Also, the succinct arguments in Part V are arguably among the simplest known, and have a strong pedagogical value (particularly so the Kilian transformation described there).
- *Graduate course on cryptographic proofs.* A graduate course on cryptographic proofs may include a selection of material about probabilistic proofs and transformations to cryptographic proofs. This book provides a useful reference for some of the simplest such transformations, those based solely on ideal hash functions.

Acknowledgements

Work on this book was generously funded by:

- the **Ethereum Foundation**;
- **Forte Labs**;
- **Protocol Labs**; and
- **Provable**.

We are sincerely grateful that these entities agreed to support our vision of this book when it was conceived in the summer of 2021. Moreover, work on this book was done in part while working at **StarkWare Industries** and we are grateful for its support.

Our knowledge and enthusiasm about cryptographic proofs has greatly benefited from interactions with many wonderful colleagues and mentors over many years of academic research, as well as from forays into industry. We express our deepest gratitude to all of these colleagues.

Several people have generously agreed to review early drafts of this book, contributing valuable ideas, feedback, and corrections. We thank them in alphabetic order: Gal Arnon, Shany Ben-David, Zijing Di, Giacomo Fenzi, Ziyi Guan, Shai Keidar, Mathias Marty, Amit Sharabi, Yuxi Zheng.

Those who reported bugs and provided suggestions are acknowledged in the corresponding pull requests on this book’s version history in the repository mentioned above.

Finally, we thank our families for tolerating our continuous work on this project for the past few years. Even if more prolonged and arduous than initially predicted, working on this project has been a joy, but would not have been possible without the generous support of our families.

1 Mathematical background

We introduce basic mathematical notation and facts that we use throughout this book.

1.1 Notation

Sets We denote by $|S|$ the cardinality of (i.e., number of elements in) a set S ; if S has an infinite number of elements then $|S| := \infty$. We denote by \mathbb{R} the set of real numbers, by \mathbb{N} the set of positive integers $\{1, 2, \dots\}$, and by $[n]$ the set of positive integers $\{1, 2, \dots, n\}$. For $k \in \{0, 1, \dots, n\}$, we denote by $\binom{[n]}{k}$ the set of all subsets of $[n]$ of cardinality k ; the number of such subsets is $\binom{n}{k} := \frac{n!}{k!(n-k)!}$. We denote by $\{x_1, \dots, x_n\}$ the set that contains the elements x_1, \dots, x_n ; its cardinality is the number of distinct elements among x_1, \dots, x_n (as duplicate elements are counted only once). Given a set S , we write $\{x_i\}_{i \in S}$ to denote the set that, for every $i \in S$, contains the element x_i ; for example, $\{x_i\}_{i \in [n]} = \{x_1, \dots, x_n\}$.

Tuples A tuple is an ordered list of elements. We denote by $(x_i)_{i \in [n]}$ the tuple (x_1, \dots, x_n) ; more generally, $(x_i)_{i \in S}$ denotes a tuple whose entries are indexed by elements of a given set S according to a canonical order. Duplicate elements in a tuple appear multiple times, at the appropriate locations. Tuples of tuples are obtained by nesting parentheses; for example, $((x_i, y_i, z_i))_{i \in [n]}$ is the tuple of tuples $((x_1, y_1, z_1), \dots, (x_n, y_n, z_n))$. If x is a tuple whose entries are indexed by a set S then, for every $i \in S$, $x[i]$ is the i -th entry of x ; more generally, for a subset $T \subseteq S$, $x[T]$ denotes the tuple obtained by considering the entries of x whose indices are in T . For example, if $x = (x_i)_{i \in \{1, 4, 7\}}$ then $x[7] = x_7$ and $x[\{1, 7\}] = (x_i)_{i \in \{1, 7\}}$.

Strings Let Σ be a finite alphabet (i.e., a finite set). A string over Σ is a function $x: D \rightarrow \Sigma$ over some ordered set D that takes values in Σ ; in particular, x is a tuple $(x_i)_{i \in D}$ where every entry x_i is an element of the alphabet Σ . Typically $D = [n]$ for some $n \in \mathbb{N}$, in which case we may write the string by listing the symbols in a sequence; for example, 001 represents the string $x: [3] \rightarrow \{0, 1\}$ where $x_1 = 0$, $x_2 = 0$, and $x_3 = 1$. We use \parallel to denote string concatenation; for example $00 \parallel 1$ is the string 001.

For every $n \in \mathbb{N}$, we denote by Σ^n the set of all strings over Σ of length n :

$$\Sigma^n := \{x: [n] \rightarrow \Sigma\} = \{(x_i)_{i \in [n]} : \forall i \in [n], x_i \in \Sigma\} .$$

We denote by $\Sigma^* := \cup_{n \in \mathbb{N}} \Sigma^n$ the set of all finite-length strings over Σ .

Given an ordered set D , we denote by Σ^D the set of strings over Σ of length $|D|$ whose entries are indexed by elements of D :

$$\Sigma^D := \{x: D \rightarrow \Sigma\} = \{(x_i)_{i \in D} : \forall i \in D, x_i \in \Sigma\} .$$

For example, if $\Sigma = \{0, 1\}$ and $D = \{1, 8, 9\}$, then a string x in $\Sigma^D = \{0, 1\}^{\{1, 8, 9\}}$ may have the following values: $x[1] = 0$, $x[8] = 1$, and $x[9] = 0$.

Bit lengths A binary string is a string over the alphabet $\Sigma = \{0, 1\}$. We denote by $\text{len}(x)$ the length of a binary string $x \in \{0, 1\}^*$; for example $\text{len}(001) = 3$. If x is not a binary string then $\text{len}(x)$ denotes the length of a suitable representation of x as a binary string. For example, if x is a string in Σ^n then $\text{len}(x) = n \cdot \lceil \log_2 |\Sigma| \rceil$ by representing each of the n symbols in x as a $\lceil \log_2 |\Sigma| \rceil$ -bit string (representing the appropriate symbol in Σ as a binary string). If x is a string in Σ^D then $\text{len}(x) = n \cdot (\lceil \log_2 |D| \rceil + \lceil \log_2 |\Sigma| \rceil)$ by representing x as a list of pairs: if $x = (x_i)_{i \in D}$ then x is encoded as a list of n pairs, each pair consisting of a $\lceil \log_2 |D| \rceil$ -bit string encoding an index i and a $\lceil \log_2 |\Sigma| \rceil$ -bit string encoding the symbol x_i .

Sampling We denote by $a \leftarrow D$ the process of sampling a according to the distribution D . Similarly, given a finite set S , we denote by $a \leftarrow S$ the process of sampling a according to the uniform distribution over S .

Asymptotics We use standard asymptotic notation. For example, given two functions $f, g: \mathbb{N} \rightarrow \mathbb{R}$, we write $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ to respectively denote the fact that there exists a constant $c > 0$ such that, for all large enough n , it holds that $f(n) \leq c \cdot g(n)$ and $f(n) \geq c \cdot g(n)$. Moreover, we write $f(n) = o(g(n))$ and $f(n) = \omega(g(n))$ to respectively denote the fact that for every constant $c > 0$, for all large enough n , it holds that $f(n) \leq c \cdot g(n)$ and $f(n) \geq c \cdot g(n)$. We additionally use notation to refer to some unspecified polynomially-bounded function: $\text{poly}(n)$ denotes $n^{O(1)}$ and, more generally, $\text{poly}(n_1, n_2, \dots)$ denotes $(n_1 + n_2 + \dots)^{O(1)}$.

Languages and relations A relation \mathcal{R} is a set consisting of instance-witness pairs (\mathbf{x}, \mathbf{w}) . The corresponding language $\mathcal{L}(\mathcal{R})$ is the set of instances \mathbf{x} for which there exists a witness \mathbf{w} such that (\mathbf{x}, \mathbf{w}) is in the relation \mathcal{R} . We write $|\mathbf{x}|$ to denote the size of an instance; this could be the number of bits to describe \mathbf{x} (namely, $\text{len}(\mathbf{x})$) or some other measure suitable for instances of the relation \mathcal{R} .

Algorithms We informally refer to algorithms as procedures running on an appropriate computation model. If we are not interested in the time or space complexity of an algorithm (such as when we consider unbounded adversaries), then the underlying computation model does not matter. (It does not matter whether it is a single-tape Turing machine, multi-tape Turing machine, register machine, or other.) If we discuss the time or space complexity of an algorithm then we assume that the algorithm is running on a register machine that resembles a modern computer (a finite set of instructions with read/write random-access to memory with appropriate word length). We consider both deterministic algorithms and probabilistic algorithms (which additionally have access to a string of random bits of a certain length).

An algorithm runs in polynomial time if its running time is upper bounded by a polynomial in the size of its input. (If the algorithm has multiple inputs, then we consider the sum of their sizes.)

Interactive algorithms An algorithm A is *interactive* if it maintains state across multiple input-to-output invocations. The outputs of A given sequence of inputs a_1, a_2, a_3, \dots are as follows:

$$b_1 \leftarrow A(a_1), b_2 \leftarrow A(a_2), b_3 \leftarrow A(a_3), \dots.$$

Sometimes, it is helpful to make explicit the state passed from one execution to the next, in which case we view the algorithm A as stateless:

$$(b_1, \text{aux}_1) \leftarrow A(a_1), (b_2, \text{aux}_2) \leftarrow A(\text{aux}_1, a_2), (b_3, \text{aux}_3) \leftarrow A(\text{aux}_2, a_3), \dots.$$

Black boxes Sometimes we describe algorithms that receive other algorithms as input and rely only on their input-output functionality (by running them on certain inputs and obtaining corresponding outputs), rather than also using their description. We emphasize when this is the case by saying that they are used as *black boxes*.

Let A and B be algorithms. We say that A uses B as a black-box, denoted $A(B)$, if A invokes B some number of times (on inputs of its choice) and otherwise does not “look” at B ’s description. More precisely, for every two algorithms B_1 and B_2 that represent the same function, $A(B_1) = A(B_2)$.

If B is an interactive algorithm, then we define B to be the black box for the *stateless* representation of B , namely, the black box for the function that given an internal state of B and an input for B returns the corresponding output of B and resulting internal state. In particular, $A(B)$ can interact with B on a sequence of inputs a_1, a_2, a_3, \dots as follows:

$$(b_1, \text{aux}_1) \leftarrow B(a_1), (b_2, \text{aux}_2) \leftarrow B(\text{aux}_1, a_2), (b_3, \text{aux}_3) \leftarrow B(\text{aux}_2, a_3), \dots.$$

There is a subtle technicality in the above sequence of black-box invocations. The (arbitrary) internal states $(\text{aux}_i)_i$ received by A from (the stateless representation of) B may contain *non-black-box* information about the algorithm B (e.g., the code of B); moreover, the internal states $(\text{aux}_i)_i$ may be large (e.g., of exponential size if B is inefficient), impacting the efficiency of A . Yet A is only interested in the input-output behavior of B , and we wish to ensure that a black-box invocation of B does not burden A with B ’s inefficiencies. However, A needs B ’s internal states in order to facilitate multiple invocations of B from the same execution point on different inputs.

This technicality is resolved by regarding the internal states $(\text{aux}_i)_i$ as merely *handles* to the actual (possibly large) internal states. These handles allow the algorithm A to invoke B as needed by specifying the handle of an internal state rather than the internal state itself (which is not exposed to A). If A performs at most t invocations, then at most t handles are needed, so each handle can be expressed as a $\lceil \log t \rceil$ -bit string. For simplicity, we slightly abuse notation and use the same notation $(b_i, \text{aux}_i) \leftarrow B(\text{aux}_{i-1}, a_i)$ as above.

Malicious parties We often (though not always) put a tilde over symbols that refer to malicious parties: if P is the symbol that we would use for an honest party, then \tilde{P} denotes a malicious party. (A malicious party is one that does not necessarily follow the prescribed instructions of a protocol.)

1.2 Probability

Events We denote by $\Pr[E]$ the probability of an event E , which is a real number in $[0, 1]$. For example, if b is a random bit then $\Pr[b = 0] = \frac{1}{2}$. We use logical operators on events: \bar{E} is the negation of E ; $E_1 \wedge E_2$ is the event where E_1 and E_2 hold; $E_1 \vee E_2$ is the event where E_1 or E_2 holds; and so on. For example, $\Pr[E \wedge \bar{E}] = 0$ and $\Pr[E] + \Pr[\bar{E}] = 1$.

Two events E_1 and E_2 are independent if (and only if) $\Pr[E_1 \wedge E_2] = \Pr[E_1] \cdot \Pr[E_2]$. In other words, the probability that both events hold simultaneously equals the product of the two probabilities where each event holds individually.

The probability of the event E_1 conditioned on the event E_2 is $\Pr[E_1 | E_2] := \frac{\Pr[E_1 \wedge E_2]}{\Pr[E_2]}$. This probability is defined only when $\Pr[E_2] > 0$. If E_1 and E_2 are independent then $\Pr[E_1 | E_2] = \Pr[E_1]$. (Conversely, if $\Pr[E_2] > 0$ and $\Pr[E_1 | E_2] = \Pr[E_1]$ then E_1 and E_2 are independent.)

The law of total probability states that the probability of an event can be split across a collection of events that partition the probability space.

Lemma 1.2.1 (law of total probability). *Let E_1 be an event.*

- For every E_2 ,

$$\Pr[E_1] = \Pr[E_1 \wedge E_2] + \Pr[E_1 \wedge \overline{E_2}].$$

- More generally, for every collection of events $E_{2,1}, \dots, E_{2,n}$ that partition the probability space (such that $\Pr[\bigvee_{i \in [n]} E_{2,i}] = 1$ and $\Pr[E_{2,i} \wedge E_{2,j}] = 0$ for every distinct $i, j \in [n]$),

$$\Pr[E_1] = \sum_{i \in [n]} \Pr[E_1 \wedge E_{2,i}].$$

If all the relevant conditional probabilities are well-defined, the equalities above can be rewritten as

$$\begin{aligned} \Pr[E_1] &= \Pr[E_1 | E_2] \cdot \Pr[E_2] + \Pr[E_1 | \overline{E}_2] \cdot \Pr[\overline{E}_2] \text{ and} \\ \Pr[E_1] &= \sum_{i \in [n]} \Pr[E_1 | E_{2,i}] \cdot \Pr[E_{2,i}]. \end{aligned}$$

The addition rule provides a way to express the probability of the union of two events:

$$\Pr[E_1 \vee E_2] = \Pr[E_1] + \Pr[E_2] - \Pr[E_1 \wedge E_2].$$

Intuitively, $\Pr[E_1] + \Pr[E_2]$ double counts where E_1 and E_2 intersect, and so subtracting $\Pr[E_1 \wedge E_2]$ corrects for this. In particular, we learn that $\Pr[E_1 \vee E_2] \leq \Pr[E_1] + \Pr[E_2]$.

More generally, if there are more than two events, there can be intersections among any number of the events, leading to more complicated combinatorics of corrections. For example, for three events (see Figure 1.1 for an illustration):

$$\begin{aligned} \Pr[E_1 \vee E_2 \vee E_3] &= \Pr[E_1] + \Pr[E_2] + \Pr[E_3] \\ &\quad - \Pr[E_1 \wedge E_2] - \Pr[E_2 \wedge E_3] - \Pr[E_3 \wedge E_1] \\ &\quad + \Pr[E_1 \wedge E_2 \wedge E_3]. \end{aligned}$$

Since $\Pr[E_1 \wedge E_2 \wedge E_3] \geq 0$, this equality implies a simple lower bound:

$$\Pr[E_1 \vee E_2 \vee E_3] \geq \sum_{i \in [3]} \Pr[E_i] - \sum_{\substack{i,j \in [3] \\ \text{with } i \neq j}} \Pr[E_i \wedge E_j].$$

Moreover, since $\Pr[E_1 \wedge E_2 \wedge E_3] \leq \Pr[E_1 \wedge E_2]$, this equality also implies a simple upper bound:

$$\Pr[E_1 \vee E_2 \vee E_3] \leq \sum_{i \in [3]} \Pr[E_i].$$

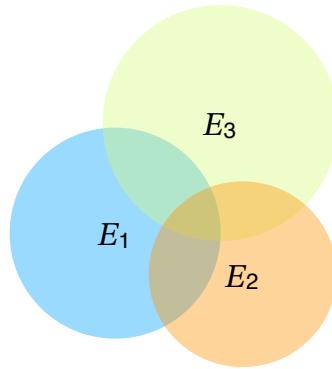


Figure 1.1: Diagram of the pairwise and three-wise intersections of events \$E_1, E_2, E_3\$.

The general case is called the *inclusion-exclusion principle*, which provides a way to express the probability of the union of \$n\$ events \$E_1, \dots, E_n\$ as a (signed) sum of the probabilities of all their intersections.

Lemma 1.2.2 (inclusion-exclusion principle). *For every \$n\$ events \$E_1, \dots, E_n\$ it holds that*

$$\Pr [\vee_{i \in [n]} E_i] = \sum_{k \in [n]} \left((-1)^{k+1} \cdot \sum_{\substack{I \subseteq [n] \\ \text{with } |I|=k}} \Pr [\wedge_{i \in I} E_i] \right).$$

The equality in the above lemma can be inconvenient to use because it involves many terms. The statement below about the probability of \$\vee_{i \in [n]} E_i\$ is simpler to use: it provides a lower bound and an upper bound in terms of the individual probabilities of the single events \$E_i\$ and of their pairwise intersections \$E_i \wedge E_j\$.

Lemma 1.2.3 (simple inclusion-exclusion principle). *For every \$n\$ events \$E_1, \dots, E_n\$,*

$$\sum_{i \in [n]} \Pr[E_i] - \sum_{\substack{i,j \in [n] \\ \text{with } i \neq j}} \Pr[E_i \wedge E_j] \leq \Pr [\vee_{i \in [n]} E_i] \leq \sum_{i \in [n]} \Pr[E_i].$$

Proof. With give a proof by induction on the number of events \$n\$.

Base case: \$n \in \{1, 2, 3\}\$. The case for \$n = 1\$ holds because all expressions in the lemma are \$\Pr[E_1]\$. Separately, in the discussion before the lemma we prove the cases where \$n = 2\$ and \$n = 3\$.

Inductive case: \$n > 1\$. We prove the lemma for \$n\$ assuming that the lemma holds for every \$n' \in [n-1]\$. Define the event \$E^* := \vee_{i \in [n-1]} E_i\$. By the induction hypothesis for \$n' = n-1\$,

$$\sum_{i \in [n-1]} \Pr[E_i] - \sum_{\substack{i,j \in [n-1] \\ \text{with } i \neq j}} \Pr[E_i \wedge E_j] \leq \Pr [E^*] \leq \sum_{i \in [n-1]} \Pr[E_i]. \quad (1.1)$$

First we prove the upper bound. By the induction hypothesis for \$n' = 2\$ (on the events \$E^*\$ and \$E_n\$), and by Equation 1.1, we have that

$$\Pr [E^* \vee E_n] \leq \Pr [E^*] + \Pr [E_n] \leq \sum_{i \in [n-1]} \Pr [E_i] + \Pr [E_n] = \sum_{i \in [n]} \Pr [E_i].$$

Next we prove the lower bound. Applying the (now proved) upper bound to the events $\{E_i \wedge E_n\}_{i \in [n]}$, we obtain

$$\Pr[\vee_{i \in [n]} (E_i \wedge E_n)] \leq \sum_{i \in [n]} \Pr[E_i \wedge E_n]. \quad (1.2)$$

By the induction hypothesis for $n' = 2$ (on the events E^* and E_n), and by Equations 1.1 and 1.2, we have that

$$\begin{aligned} & \Pr[E^* \vee E_n] \\ & \geq \Pr[E^*] + \Pr[E_n] - \Pr[E^* \wedge E_n] \\ & \geq \left(\sum_{i \in [n-1]} \Pr[E_i] - \sum_{\substack{i,j \in [n-1] \\ \text{with } i \neq j}} \Pr[E_i \wedge E_j] \right) + \Pr[E_n] - \Pr[E^* \wedge E_n] \\ & = \sum_{i \in [n]} \Pr[E_i] - \sum_{\substack{i,j \in [n-1] \\ \text{with } i \neq j}} \Pr[E_i \wedge E_j] - \Pr[\vee_{i \in [n]} (E_i \wedge E_n)] \\ & \geq \sum_{i \in [n]} \Pr[E_i] - \sum_{\substack{i,j \in [n] \\ \text{with } i \neq j}} \Pr[E_i \wedge E_j]. \end{aligned}$$

□

We describe events via expressions that involve random variables. In this book random variables take values that are binary strings or real values. In the latter case, a binary random variable is one that takes only values that are zero or one.

Random variables If X is a random variable that takes values in a finite set $S \subseteq \mathbb{R}$ then we can define its *expectation* as follows:

$$\mathbb{E}[X] := \sum_{x \in S} x \cdot \Pr[X = x].$$

Two random variables X and Y are independent if (and only if) for every x and y it holds that $\Pr[X = x \wedge Y = y] = \Pr[X = x] \cdot \Pr[Y = y]$. More generally, the random variables X_1, \dots, X_n are independent if (and only if) for every x_1, \dots, x_n it holds that $\Pr[\wedge_{i \in [n]} X_i = x_i] = \prod_{i \in [n]} \Pr[X_i = x_i]$.

The law of large numbers states that, given an infinite sequence of random variables $\{X_i\}_{i \in \mathbb{N}}$ that are independently and identically distributed, $\mathbb{E}[X]$ is the limit of $\frac{1}{n} \sum_{i \in [n]} X_i$ as n tends to infinity. The distance of $\frac{1}{n} \sum_{i \in [n]} X_i$ from $\mathbb{E}[X]$ as a function of n is captured by *concentration bounds*. One example, is the following (additive) *Chernoff bound*.

Lemma 1.2.4 (Chernoff bound). *Let X_1, \dots, X_n be independent binary random variables such that $\Pr[X_i = 1] = p_i$ for every $i \in [n]$. Let $X := \sum_{i \in [n]} X_i$ and $\mu := \mathbb{E}[X] = \sum_{i \in [n]} p_i$. For every $\delta \in (0, 1)$,*

$$\Pr[|X - \mu| \geq \delta \cdot \mu] \leq 2 \cdot e^{-\frac{\mu \delta^2}{3}}.$$

Statistical distance We measure the statistical distance between two random variables in terms of their total variation distance, as in the definition below.

Definition 1.2.5. *The statistical distance between two random variables X and Y taking values in a finite set S is defined as*

$$\Delta(X, Y) := \frac{1}{2} \sum_{a \in S} |\Pr[X = a] - \Pr[Y = a]|. \quad (1.3)$$

Equivalently, the statistical distance can also be defined as

$$\Delta(X, Y) := \max_{S' \subseteq S} |\Pr[X \in S'] - \Pr[Y \in S']|. \quad (1.4)$$

We show that the two definitions given above (Equations 1.3 and 1.4) are equivalent.

Claim 1.2.6. *The definition using Equation 1.3 is equivalent to the definition using Equation 1.4.*

Proof. Define the subset $S_0 \subseteq S$ as

$$S_0 := \{a \in S : \Pr[X = a] \geq \Pr[Y = a]\}.$$

Additionally, let S_1 be the complement set: $S_1 := S \setminus S_0$.

Observe that $\Pr[X \in S_0] + \Pr[X \in S_1] = 1$ and $\Pr[Y \in S_0] + \Pr[Y \in S_1] = 1$. Hence

$$\Pr[X \in S_0] - \Pr[Y \in S_0] = \Pr[Y \in S_1] - \Pr[X \in S_1]. \quad (1.5)$$

One can argue that S_0 or S_1 maximizes the expression $\max_{S' \subseteq S} |\Pr[X \in S'] - \Pr[Y \in S']|$. Then, using Equation 1.5, we get that

$$\Pr[Y \in S_1] - \Pr[X \in S_1] = \max_{S' \subseteq S} |\Pr[X \in S'] - \Pr[Y \in S']| = \Pr[X \in S_0] - \Pr[Y \in S_0].$$

Finally, we write

$$\begin{aligned} \Delta(X, Y) &:= \frac{1}{2} \sum_{a \in S} |\Pr[X = a] - \Pr[Y = a]| \\ &= \frac{1}{2} \left(\sum_{a \in S_0} (\Pr[X = a] - \Pr[Y = a]) + \sum_{a \in S_1} (\Pr[Y = a] - \Pr[X = a]) \right) \\ &= \frac{1}{2} (\Pr[X \in S_0] - \Pr[Y \in S_0] + \Pr[Y \in S_1] - \Pr[X \in S_1]) \\ &= \frac{1}{2} (\Pr[X \in S_0] - \Pr[Y \in S_0] + (1 - \Pr[Y \in S_0]) - (1 - \Pr[X \in S_0])) \\ &= \Pr[X \in S_0] - \Pr[Y \in S_0] \\ &= \max_{S' \subseteq S} |\Pr[X \in S'] - \Pr[Y \in S']|. \end{aligned}$$

□

The following claim states basic properties about statistical distance that we use.

Claim 1.2.7. *The statistical distance function satisfies the following properties.*

1. For every random variable X , $\Delta(X, X) = 0$.
2. For every random variables X, Y , $0 \leq \Delta(X, Y) \leq 1$.
3. For every random variables X, Y, Z , $\Delta(X, Z) \leq \Delta(X, Y) + \Delta(Y, Z)$.

Proof. The first two properties follow directly from the definition. We prove the third property as follows:

$$\begin{aligned}
\Delta(X, Z) &= \frac{1}{2} \sum_a |\Pr[X = a] - \Pr[Z = a]| \\
&= \frac{1}{2} \sum_a |(\Pr[X = a] - \Pr[Y = a]) + (\Pr[Y = a] - \Pr[Z = a])| \\
&\leq \frac{1}{2} \sum_a |\Pr[X = a] - \Pr[Y = a]| + \frac{1}{2} \sum_a |(\Pr[Y = a] - \Pr[Z = a])| \\
&= \Delta(X, Y) + \Delta(Y, Z).
\end{aligned}$$

□

The following claim states that manipulating two random variables X and Y by applying the same function f does not increase their statistical distance.

Claim 1.2.8. *For every two random variables X, Y over a domain S and function f defined over S ,*

$$\Delta(f(X), f(Y)) \leq \Delta(X, Y).$$

Proof.

$$\begin{aligned}
\Delta(f(X), f(Y)) &= \frac{1}{2} \sum_{b \in f(S)} |\Pr[f(X) = b] - \Pr[f(Y) = b]| \\
&= \frac{1}{2} \sum_{b \in f(S)} \left| \sum_{a \in f^{-1}(b)} \Pr[X = a] - \Pr[Y = a] \right| \\
&\leq \frac{1}{2} \sum_{b \in f(S)} \sum_{a \in f^{-1}(b)} |\Pr[X = a] - \Pr[Y = a]| \\
&\leq \frac{1}{2} \sum_{a \in S} |\Pr[X = a] - \Pr[Y = a]| \\
&\leq \Delta(X, Y).
\end{aligned}$$

□

Suppose that we have a random variable X and two random variables Y and Y' that depend on X . The following claim states that if Y and Y' are statistically close for many values of X then the joint distributions (X, Y) and (X, Y') are statistically close.

Claim 1.2.9. *Let X be a random variable, and let $\{Y(x)\}_x$ and $\{Y'(x)\}_x$ be families of random variables. Define Y to be the random variable obtained by sampling x from X , and then sampling y from $Y(x)$; similarly define Y' . If $\Pr_{x \leftarrow X}[\Delta(Y(x), Y'(x)) \geq \delta] \leq \epsilon$ then $\Delta((X, Y), (X, Y')) \leq \delta + \epsilon$.*

Proof. Define $S := \{x : \Delta(Y(x), Y'(x)) \geq \delta\}$. Then, we have that

$$\begin{aligned}
& \Delta((X, Y), (X, Y')) \\
&= \frac{1}{2} \cdot \sum_{x,y} |\Pr[(X, Y) = (x, y)] - \Pr[(X, Y') = (x, y)]| \\
&= \frac{1}{2} \cdot \sum_{x,y} |\Pr[X = x] \cdot \Pr[Y(x) = y] - \Pr[X = x] \cdot \Pr[Y'(x) = y]| \\
&= \sum_x \Pr[X = x] \cdot \frac{1}{2} \cdot \sum_y |\Pr[Y(x) = y] - \Pr[Y'(x) = y]| \\
&= \sum_x \Pr[X = x] \cdot \Delta(Y(x), Y'(x)) \\
&= \sum_{x \in S} \Pr[X = x] \cdot \Delta(Y(x), Y'(x)) + \sum_{x \notin S} \Pr[X = x] \cdot \Delta(Y(x), Y'(x)) \\
&\leq \sum_{x \in S} \Pr[X = x] \cdot 1 + \sum_{x \notin S} \Pr[X = x] \cdot \delta \\
&\leq \epsilon + \delta.
\end{aligned}$$

□

Claim 1.2.10. Let X and Y be two random variables over the same probability space. Let E_1 , E_2 be two events with $\Pr[E_1] = \Pr[E_2]$. Then,

$$\Delta(X, Y) \leq \Delta((X | E_1), (Y | E_2)) + \Pr[\overline{E_1}]. \quad (1.6)$$

Proof. Let S be the support of the random variables. For every subset $S' \subseteq S$,

$$\begin{aligned}
& |\Pr[X \in S'] - \Pr[Y \in S']| \\
&= |\Pr[X \in S' | E_1] \cdot \Pr[E_1] + \Pr[X \in S' | \overline{E_1}] \cdot \Pr[\overline{E_1}] \\
&\quad - \Pr[Y \in S' | E_2] \cdot \Pr[E_2] - \Pr[Y \in S' | \overline{E_2}] \cdot \Pr[\overline{E_2}]| \quad (\text{by total probability}) \\
&= |\Pr[E_1] \cdot (\Pr[X \in S' | E_1] - \Pr[Y \in S' | E_2]) \\
&\quad + \Pr[\overline{E_1}] \cdot (\Pr[X \in S' | \overline{E_1}] - \Pr[Y \in S' | \overline{E_2}])| \quad (\text{since } \Pr[E_1] = \Pr[E_2]) \\
&\leq \Pr[E_1] \cdot |\Pr[X \in S' | E_1] - \Pr[Y \in S' | E_2]| \\
&\quad + \Pr[\overline{E_1}] \cdot |\Pr[X \in S' | \overline{E_1}] - \Pr[Y \in S' | \overline{E_2}]| \\
&\leq |\Pr[X \in S' | E_1] - \Pr[Y \in S' | E_2]| + \Pr[\overline{E_1}].
\end{aligned}$$

Then, by the equivalent definition of statistical distance (Equation 1.4),

$$\begin{aligned}
\Delta(X, Y) &:= \max_{S' \subseteq S} |\Pr[X \in S'] - \Pr[Y \in S']| \\
&\leq \max_{S' \subseteq S} |\Pr[X \in S' | E_1] - \Pr[Y \in S' | E_2]| + \Pr[\overline{E_2}] \\
&= \Delta((X | E_1), (Y | E_2)) + \Pr[\overline{E_1}].
\end{aligned}$$

□

1.3 Probabilistic experiments

We introduce notation for defining random variables and events over these.

- We write $\{x \mid x \leftarrow A\}$ to denote the random variable that consists of the string x sampled according to the probabilistic algorithm A .
- We write $\Pr[f(x) = 1 \mid x \leftarrow A]$ to denote the probability of the event that $f(x) = 1$ when x is sampled according to the probabilistic algorithm A , for a given boolean function f .
- We write $\Pr[D(x) = 1 \mid x \leftarrow A]$ to denote the probability of the event that $D(x) = 1$ when x is sampled according to the probabilistic algorithm A , for a given probabilistic algorithm D . In this case the probability is over the randomness of both A and D .

Sometimes we implicitly define the algorithm A or the boolean function *within the expression itself*, by using pseudocode and logical expressions. For example, we can write

$$\left\{ (y, z) \mid \begin{array}{l} y \leftarrow B \\ z \leftarrow C(y) \end{array} \right\}$$

to denote the random variable $x := (y, z)$ that is obtained by the algorithm A that samples y according to B , samples z according to C on input y , and outputs the pair (y, z) . We can also write

$$\Pr \left[y = z \mid \begin{array}{l} y \leftarrow B \\ z \leftarrow C(y) \end{array} \right]$$

to denote the probability that the random variable $x = (y, z)$ sampled as above is such that $f(x) = 1$ where f is the boolean function that given a pair $x = (y, z)$ outputs 1 if and only if $y = z$.

The main purpose of probabilistic experiments in this book is to specify security properties of cryptographic schemes, by setting out the actions of an adversary and then providing “winning conditions” for its output.

1.4 Auxiliary (in)equalities

We state several equalities and inequalities that we use throughout this book.

Lemma 1.4.1 (bounds on binomial coefficients). *For every $n, k \in \mathbb{N}$ with $1 \leq k \leq n$,*

$$\left(\frac{n}{k} \right)^k \leq \binom{n}{k} \leq \left(\frac{e \cdot n}{k} \right)^k.$$

Lemma 1.4.2 (arithmetic progression). *For every $a, d \in \mathbb{R}$,*

$$\sum_{i=1}^n (a + (i-1)d) = na + \frac{n(n-1)}{2}d.$$

Lemma 1.4.3 (geometric progression). *For every $r \in (0, 1)$ and $a \in \mathbb{R}$,*

$$\sum_{i=1}^n ar^{i-1} = a \frac{r^n - 1}{r - 1}.$$

Lemma 1.4.4. Let $\delta_1, \dots, \delta_n \in [0, 1]$ be such that $\sum_{i=1}^n \delta_i = 1$. Then,

$$\sum_{i=2}^n \delta_{i-1} \cdot \delta_i \leq 1/4.$$

Proof. We first observe that getting an upper bound of 1 is relatively easy. Indeed, since each $\delta_i \leq 1$ we can bound the expression by $\sum_{i=2}^n \delta_{i-1} \cdot \delta_i \leq \sum_{i=2}^n \delta_i \leq 1$. The rest of the proof is dedicated to give the tight bound of 1/4.

Our first step is to show that the expression is maximized when all but three of the variables are zero. Let $i \in [n]$ be the minimal index to maximize the sum $\delta_{i-1} + \delta_{i+1}$. Fix any $j \in [n] \setminus \{i-1, i, i+1\}$ for which $\delta_j > 0$. We show that one can increase the sum by moving all the weight of δ_j to δ_i . Define

$$\delta'_k := \begin{cases} 0 & \text{if } k = j \\ \delta_i + \delta_j & \text{if } k = i \\ \delta_k & \text{otherwise} \end{cases}.$$

Then, we show that the expression with $\delta'_1, \dots, \delta'_n$ is greater (or equal) to the expression with $\delta_1, \dots, \delta_n$. Namely, we show that

$$\sum_{i=1}^n \delta_{i-1} \cdot \delta_i \leq \sum_{i=1}^n \delta'_{i-1} \cdot \delta'_i.$$

Most of the terms are the same on both sides. The terms that are different are those that include i or j . Thus, the expression can be rewritten as

$$\delta_{j-1}\delta_j + \delta_j\delta_{j+1} + \delta_{i-1}\delta_i + \delta_i\delta_{i+1} \leq \delta'_{j-1}\delta'_j + \delta'_j\delta'_{j+1} + \delta'_{i-1}\delta'_i + \delta'_i\delta'_{i+1}.$$

We expand the right side. Since $\delta'_j = 0$, $\delta'_i = \delta_i + \delta_j$, we get that it suffices to show that

$$\delta_{j-1}\delta_j + \delta_j\delta_{j+1} + \delta_{i-1}\delta_i + \delta_i\delta_{i+1} \leq \delta_{i-1}(\delta_i + \delta_j) + (\delta_i + \delta_j)\delta_{i+1}.$$

Since $\delta_j \neq 0$, this can be rewritten as

$$\delta_{j-1} + \delta_{j+1} \leq \delta_{i-1} + \delta_{i+1},$$

which is true by our assumption on i . We conclude that $\delta_k = 0$ for all $k \in [n] \setminus \{i-1, i, i+1\}$.

Given this, we rewrite the expression of the lemma as

$$\delta_{i-1} \cdot \delta_i + \delta_i \cdot \delta_{i+1} = \delta_i \cdot (\delta_{i-1} + \delta_{i+1}) = \delta_i \cdot (\delta_{i-1} + 1 - \delta_{i-1} - \delta_i) = \delta_i \cdot (1 - \delta_i) \leq 1/4.$$

□

Part I

What are Cryptographic Proofs?

Overview

We introduce the random oracle model and discuss basic cryptographic properties of random oracles. Then we introduce the main object of study in this book: arguments in the random oracle model. We build intuition by proving several basic observations about arguments in the random oracle model. We conclude by describing the setting of multiple random oracles, which can be realized via a single random oracle and we often use to simplify discussions.

2	The random oracle model	14
2.1	Definition	14
2.2	Simulation by lazy sampling	16
2.3	Pseudorandom functions vs. random oracles	17
2.4	Realizations	19
2.5	Benefits and drawbacks	23
2.6	The setting of multiple random oracles	24
3	Basic cryptographic properties	29
3.1	Unpredictability	29
3.2	Inversion resistance	30
3.3	Collision resistance	32
4	Arguments in the ROM	35
4.1	Non-interactive arguments	35
4.2	Interactive arguments	37
4.3	Efficiency measures	39
4.4	Succinct arguments	40
5	Additional security definitions	42
5.1	Knowledge soundness	42
5.2	Zero knowledge	47
6	Basic observations about arguments	51
6.1	Error reduction	51
6.2	Non-adaptive vs. adaptive security	54
6.3	Construction with an inefficient prover	56
6.4	Lower bound on argument size	59
6.5	Limitation on public randomness	61
6.6	Limitation on zero knowledge	62
6.7	Simulation on instances not in the language	64
7	Arguments with multiple random oracles	66
7.1	Definitions	66
7.2	Multiple oracles based on a single oracle	68

2 The random oracle model

Hash functions are a fundamental cryptographic primitive that enables mapping any input into a fixed-size output that “looks random”. A popular tool to study cryptographic protocols based on hash functions is the *random oracle model* (abbreviated *ROM*). The hash function in this model is assumed to be *ideal* in the sense that every input is mapped to a random output, independently of any other inputs. This is modeled by granting to everyone (honest parties and malicious parties) query access to a random function, known as a *random oracle*.

All constructions of cryptographic proofs studied in this book are in the ROM. In this chapter we introduce the basic notations and definitions of the ROM, and discuss the benefits and drawbacks of cryptography in the ROM.

2.1 Definition

We formalize the ROM.

Random oracles For $\sigma \in \mathbb{N}$, we denote by $\mathcal{U}(\sigma)$ the uniform distribution over all functions of the form $f: \{0, 1\}^* \rightarrow \{0, 1\}^\sigma$. Equivalently, if f is sampled from $\mathcal{U}(\sigma)$, then for every input x it holds that $y := f(x)$ is a uniformly random σ -bit string (sampled independently for each input). We refer to f sampled from $\mathcal{U}(\sigma)$ as a *random oracle* with output size σ .

Following the sampling notation in Section 1.1, we denote sampling f from $\mathcal{U}(\sigma)$ as

$$f \leftarrow \mathcal{U}(\sigma).$$

The random function f is an idealization of a real-world hash function because each input in $\{0, 1\}^*$ is mapped to a random output, independently of the output for any other input; in contrast, any hash function that is computable via a polynomial-time algorithm (sampled from some distribution) cannot satisfy such a strong property.¹

Oracle algorithms A random oracle f sampled from $\mathcal{U}(\sigma)$ cannot be given as input to an algorithm because f has infinite size. Instead, algorithms are granted *query access* to f . In more detail, an oracle algorithm is an algorithm that, in addition to an input, receives query access to an oracle, which it can query any number of times via a special operation. The operation consists of specifying an input for the oracle (a *query*) and in the following computational step receiving the corresponding oracle output (an *answer*). An oracle algorithm may be deterministic or probabilistic (it has the ability to use randomness separately from any random answers from the oracle itself).

For an algorithm A , oracle f , and input a , we use the notation

$$b \leftarrow A^f(a)$$

¹The description of the hash function would have entropy bounded via some polynomial, whereas answering queries like a random oracle requires arbitrarily large entropy.

to denote the fact that A , given query access to f and given input a , outputs b ; note that b may be a random variable if A is probabilistic (i.e., if A has its own private randomness).

For $t \in \mathbb{N}$, an oracle algorithm is t -query if it makes at most t queries (regardless of the answers).

Note that if an algorithm A runs in time T then A can only query the given oracle f at inputs x whose size is at most T . In particular, only a finite part of f of size $2^T \cdot \sigma$ matters to A 's execution.

The model The ROM can be formally stated according to the following definition.

Definition 2.1.1. *The ROM with output size $\sigma \in \mathbb{N}$ is the model where all parties (honest and malicious) are oracle algorithms and they are each given query access to the same function $f: \{0,1\}^* \rightarrow \{0,1\}^\sigma$ sampled from the distribution $\mathcal{U}(\sigma)$.*

The key aspect of the ROM is that everyone has access to the *same* random function. This is different from the model where each party has query access to its own independently sampled random function; in fact, such an alternative model is trivial because each party can efficiently simulate its own random function (via lazy sampling as discussed in Section 2.2). In other words, the ROM ensures that all parties access the same random answers for every possible query.

Security analyses of cryptographic protocols in the ROM differ depending on which class of adversaries is considered for the purposes of establishing security.

- *Bounded-query adversaries.* In this setting we bound the number of queries that the adversary can make to the random oracle, but do *not* bound any of the adversary's other resources (in time, space, randomness, and so on). We use the integer symbol $t \in \mathbb{N}$ to denote the adversary's query bound. Intuitively, as t grows so does the ability of a t -query adversary to break the security of a given cryptographic protocol.
- *Bounded-time adversaries.* In this setting we bound the running time of an adversary, and in particular also bound its number of queries to the random oracle. (An adversary can make no more queries than its running time because performing a query consumes one unit of time.)

(More generally, one could consider bounding additional or other resources beyond running time, such as memory consumption. This would lead to corresponding settings.)

The setting of bounded-query adversaries can be viewed as the “pure” ROM where protocols can leverage, and only leverage, the same random function given to everyone. On the other hand, the setting of bounded-time adversaries enables additionally leveraging hardness assumptions (e.g., the existence of one-way functions). In particular, any result achieved in the setting of bounded-query adversaries directly holds in the setting of bounded-time adversaries (but not vice versa).

All constructions of arguments in this book are in the setting of bounded-query adversaries (the “pure” ROM), and so by default we always refer to this setting when we discuss the ROM.

Additional notions The notions below help in describing and analyzing protocols in the ROM.

- **Query encodings.** A random oracle f can receive as input any query $x \in \{0, 1\}^*$. Sometimes we define a query x to be a tuple assembled from other binary strings. For example, $x = (x_1, x_2)$ where x_1, x_2 are other binary strings of a certain length; in such a case we implicitly use an unambiguous encoding of the tuple (x_1, x_2) as a binary string in $\{0, 1\}^*$. Sometimes we include integers in the tuple (e.g., $x = (3, x')$) and it is understood that the integer is represented via a suitable binary string of a certain length (usually prescribed by a protocol.)
- **Query-answer traces.** Sometimes we analyze the *query-answer trace* between an algorithm A and an oracle f . We use the notation

$$b \xleftarrow{\text{tr}} A^f(a)$$

to denote the fact that tr is the ordered list of query-answer pairs made/received by the algorithm A on input a ; note that tr is a random variable if the algorithm A is probabilistic. We denote by $|\text{tr}|$ the number of query-answer pairs in the trace tr .

- **Query size.** An algorithm A can query the random oracle f at inputs of different size. The *query size* of a query x to f is defined to be $\text{len}(x)$ (the number of bits in the binary string x). For example, if an algorithm queries f at 001, 0101, and 010, then the algorithm made 2 queries of size 3 and 1 query of size 4.

Remark 2.1.2 (input size of the random oracle). Sometimes a random oracle is defined in a more restricted way, with a domain that fixes input size rather than allowing any input size. Letting $\mathcal{U}(\lambda \rightarrow \sigma)$ be the uniform distribution over all functions of the form $f: \{0, 1\}^\lambda \rightarrow \{0, 1\}^\sigma$, a random oracle with input size λ and output size σ is a function f sampled from $\mathcal{U}(\lambda \rightarrow \sigma)$. A function sampled from $\mathcal{U}(\lambda \rightarrow \sigma)$ has size $2^\lambda \cdot \sigma$, while a function sampled from $\mathcal{U}(\sigma)$ has infinite size. With only few exceptions in this book we consider random oracles sampled from $\mathcal{U}(\sigma)$ (i.e., oracles that can take inputs of any size). This simplifies constructions and analyses.

Remark 2.1.3. The *uniform random string model* (URSM) is the setting where all parties (honest and malicious) have access to a common random string of polynomial size. This model differs from the ROM in the size of this random string: the ROM can be viewed as the setting where everyone has access to the same exponentially-large random string, whereas the URSM requires the random string to be polynomially-large. The URSM is a weaker model than the ROM but does not allow for constructions of succinct arguments with unconditional security. (See Section 6.5.)

2.2 Simulation by lazy sampling

A random oracle is an object that cannot be efficiently sampled in its entirety. Nevertheless, a probabilistic algorithm can efficiently emulate the input-output behavior of a random oracle *by maintaining state across queries*. We call this **lazy sampling** of a random oracle.

A random oracle sampled from $\mathcal{U}(\sigma)$ assigns an independently chosen random answer in $\{0, 1\}^\sigma$ to every input query. The simulating algorithm maintains a list of past query-answer pairs. For each received query, if the query appears in the list, then the algorithm returns the corresponding answer; otherwise, the algorithm samples a random answer, adds the new query-answer pair to the list, and returns the sampled answer.

Formally, lazily sampling an oracle from $\mathcal{U}(\sigma)$ corresponds to an instance of the following probabilistic stateful algorithm.

- *Initialization:* Set the internal state S to be an empty mapping.
- *To answer a query x :* If S contains the key x then set $y := S[x]$; otherwise, sample $y \leftarrow \{0, 1\}^\sigma$, and set $S[x] := y$. Output y .

Throughout the book, we use the phrase

“lazily sample an oracle $\dot{f} \leftarrow \mathcal{U}(\sigma)$ ”

with the notation \dot{f} to denote the process of using a fresh instance of the aforementioned probabilistic stateful algorithm. The notation \dot{f} enables us to use this random variable like any other oracle, while also reminding the reader that the oracle is lazily sampled.

Note that lazy sampling is *not* a realization of a random oracle because multiple parties, each performing lazy sampling, will disagree on answers to the same queries (because they each independently sample different randomness for the answers).

Nevertheless, lazy sampling is valuable for multiple reasons. First, lazy sampling demonstrates that a random oracle has *simple structure*; indeed, not all distributions over functions allow for (efficient) lazy sampling.² Furthermore, from a pragmatic standpoint, lazy sampling is a *common ingredient in security reductions*. In a security reduction, an algorithm may run as a subroutine another algorithm that expects to access a random oracle. If the outer algorithm can simply forward all queries to an external random oracle sampled for all parties then no lazy sampling is needed. However, the outer algorithm may need to rely on lazy sampling to answer queries. This may be because an external random oracle does not exist in the setting of the outer algorithm, or may be because the outer algorithm sometimes answers queries using an external random oracle and sometimes answers queries via lazy sampling, possibly correlating these latter with other quantities. We use lazy sampling within security reductions numerous times throughout this book.

2.3 Pseudorandom functions vs. random oracles

A pseudo-random function is a cryptographic realization of a random oracle that plays a fundamental role in modern cryptography. Here we review and discuss its definition. The takeaway is that pseudorandom functions do *not* play an important role in the material in this book because their security guarantee requires that the (description of) the pseudorandom function remains secret.

The security property of a pseudorandom function is that its input-output behavior is computationally indistinguishable from that of a random oracle. The adversary sees *only* the input-output behavior and, in particular, does not receive as input the description of the function. For simplicity, below we consider the notion of a random oracle with fixed input size discussed in Remark 2.1.2.

²Here is a simple (and degenerate) example; other more interesting examples are also possible. Given a table $g: \{0, 1\}^\sigma \rightarrow \{0, 1\}^\sigma$, consider the (degenerate) distribution \mathcal{D}_g that assigns probability 1 to the oracle $f: \{0, 1\}^* \rightarrow \{0, 1\}^\sigma$ that answers queries in $\{0, 1\}^\sigma$ according to the table g and all other queries with 0^σ . With high probability, the shortest description of a random table $g: \{0, 1\}^\sigma \rightarrow \{0, 1\}^\sigma$ has $\Omega(2^\sigma)$ bits, in which case \mathcal{D}_g does not have efficient lazy sampling. We conclude that there exists a distribution over functions that has no efficient lazy sampling.

Definition 2.3.1. A pseudorandom function family with error ϵ is a family $\{F_{\lambda,\sigma}\}_{\sigma \in \mathbb{N}}$, where each $F_{\lambda,\sigma}$ is a distribution over functions $f: \{0,1\}^\lambda \rightarrow \{0,1\}^\sigma$, such that, for every input size λ and output size σ of the random oracle and t -time oracle algorithm A , the following two distributions are $\epsilon(\lambda, \sigma, t)$ -close in statistical distance:

$$\left\{ A^f \mid f \leftarrow \mathcal{U}(\lambda \rightarrow \sigma) \right\} \text{ and } \left\{ A^f \mid f \leftarrow F_{\lambda,\sigma} \right\}.$$

The functions in the family should also be *efficient to sample* and *efficient to evaluate*: there must exist a probabilistic algorithm that samples f from $F_{\lambda,\sigma}$ in time $\text{poly}(\lambda, \sigma)$, and a deterministic algorithm that given the sampled function f and an input x computes $f(x)$ in time $\text{poly}(\lambda, \sigma)$.

Pseudorandom functions are efficient derandomizations of random oracles. Sampling a random oracle from $\mathcal{U}(\lambda \rightarrow \sigma)$ requires independently sampling for each input in $\{0,1\}^\lambda$ a random string in $\{0,1\}^\sigma$ to assign to its output. This involves $2^\lambda \cdot \sigma$ random bits. On the other hand, sampling a pseudorandom function can be done by running the sampling algorithm, which uses $\text{poly}(\lambda, \sigma)$ random bits because the sampling algorithm runs in time $\text{poly}(\lambda, \sigma)$.

Pseudorandom functions are useful in settings where multiple parties who share the same secret wish to be “synchronized” in accessing the same exponentially-large random-looking string. These parties can use f sampled from $F_{\lambda,\sigma}$ applied to relevant inputs in order to derive outputs that “look random” to everyone else (who have no information about f besides input-output pairs).

For example, pseudorandom functions directly imply a secret-key encryption scheme that can be viewed as a derandomization of the one-time pad. The two parties who wish to communicate via encrypted messages initially agree on a function $f: \{0,1\}^\sigma \rightarrow \{0,1\}^\sigma$ sampled from $F_{\sigma,\sigma}$, which serves as their shared secret key. A party encrypts a message $m \in \{0,1\}^\sigma$ by sampling a random salt $r \in \{0,1\}^\sigma$ and sending the ciphertext $c := (r, f(r) \oplus m) \in \{0,1\}^{2\sigma}$. This encryption scheme can be proved secure (e.g., against chosen plaintext attacks), and the intuition is that the pseudorandom function f enables the two parties to choose the ephemeral secret key $f(r)$ to encrypt m via the one-time pad; other parties, who have no information about f learn no information about $f(r)$ from ciphertexts.

Pseudorandom functions are a “lightweight” cryptographic object. They can be derived from basic cryptographic objects (e.g., one-way functions) or from hardness assumptions about groups (e.g., the discrete logarithm assumption), lattices (e.g., the learning with errors assumption), and more. In contrast, from a theoretical standpoint, the ROM is a strong idealized model with notable limitations regarding realizations with formal security guarantees (see Section 2.5).

An insecure realization of the ROM The definition of a pseudorandom function suggests a natural, but *insecure*, realization of the ROM: publish the description of a pseudorandom function to be used as a random oracle. This realization is insecure because the security property of a pseudorandom function *requires that its description remain secret*. But in this realization, the description of the pseudorandom function is published for everyone (honest and malicious parties) to use, so we cannot rely on the security property. This concern is not hypothetical: known pseudorandom functions can be efficiently distinguished from a random oracle given their description.

Nevertheless, pseudorandom functions are an ingredient to some candidate realizations of the ROM, including via hardware tokens and software obfuscation (see Section 2.4.2). However, these candidate realizations are not particularly useful in practice, and, more generally, pseudorandom

functions do not play an important role in realizing random oracles in practice. Instead, in practice, the ROM is typically heuristically realized via suitable cryptographic hash functions (see Section 2.4.1), whose descriptions are known to everyone (and do not rely on any secrets).

2.4 Realizations

The ROM is an idealized model used for describing and analyzing protocols. In practice the ROM must be efficiently realized in some way (as it is inefficient to sample a random function and publish it for everyone to use). We informally discuss different types of realizations of the ROM: in Section 2.4.1 we discuss the most common, and practically useful, realization of a random oracle via cryptographic hash functions; and in Section 2.4.2 we discuss alternative realizations that, while less practically useful, may be relevant in some settings.

2.4.1 Cryptographic hash functions

Random oracles are typically realized via specific hash functions that are believed to be “cryptographically strong” in the sense that their input-output behavior is believed to sufficiently approximate that of a random oracle, at least for the purpose of preserving the security of certain protocols. Below, we discuss this heuristic realization and comment on examples of hash functions.

The ROM heuristic Let \mathbb{P} be a protocol that uses a random oracle $f: \{0, 1\}^* \rightarrow \{0, 1\}^\sigma$ sampled from the distribution $\mathcal{U}(\sigma)$. Let $g: \{0, 1\}^* \rightarrow \{0, 1\}^\sigma$ be an efficiently computable function (there is an algorithm that computes $g(x)$ in time $\text{poly}(\sigma, \text{len}(x))$). Instantiating the ROM for \mathbb{P} with g means the following: (1) g is published for everyone to see (more precisely, the algorithm to compute g is published); and (2) \mathbb{P} is run with g in place of f (any query to f is replaced with a corresponding evaluation of g via its efficient algorithm). These modifications lead to a new protocol \mathbb{P}' , not anymore in the ROM, where all parties are assumed to know the hash function g . (Cryptographers say that such a protocol is in the *common reference string model*.)

Since the function g is not a random oracle, the new protocol \mathbb{P}' does not, in general, inherit the security proofs of \mathbb{P} in the ROM. Hence *the new protocol \mathbb{P}' may have no proofs for its security*.

Instead, we *assume* that \mathbb{P}' is secure, which is a plausible assumption as long as g is a sufficiently “strong” hash function. Formally stating this assumption, which depends on \mathbb{P} and g , requires formulating suitable analogues of the security properties for the setting where there is no random oracle anymore. We refer to this assumption as the *ROM heuristic* for the protocol \mathbb{P} and hash function g . We use the term “heuristic” to highlight that, typically, this assumption is not known to be reducible to more elementary assumptions.

Which hash functions are good? What hash functions g are suitable for heuristically realizing a random oracle f (at least for a particular protocol of interest)? At a minimum, we would want the hash function g to plausibly satisfy a selection of security properties of a random oracle f .

For example, as proved in Section 3.2, a random oracle f is inversion resistant: given an output y , any adversary must make many queries to f in order to find an input x such that

$f(x) = y$. Hence, we want g to plausibly satisfy a similar property: finding x such that $g(x) = y$ requires a lot of time (and hence is infeasible in practice).

Other examples of security properties of a random oracle include: *collision resistance* (hardness of finding $x_1 \neq x_2$ such that $f(x_1) = f(x_2)$), and *prefix resistance* (hardness of finding x such that $f(x)$ begins with many zeros). We want g to plausibly satisfy similar hardness properties.

More generally, we want g to exhibit no useful structure in its input-output behavior so that it is as “random-looking” as possible (even though we know that g is not a random function). No attack should be known against g , beyond *generic attacks* (ones that use g only as a black-box).

Cryptanalysts study concrete proposals of hash functions g looking for non-generic attacks. They also establish partial security results, such as proving that certain classes of attacks cannot do better than generic ones. Eventually, the lack of non-generic attacks after careful study of g becomes circumstantial evidence that g can be a suitable heuristic realization of a random oracle f .

Hash functions in the real world Fortunately, there exist several constructions of hash functions that are widely used in practice as heuristic realizations of a random oracle, and they appear to be excellent candidates for this purpose. For many of them, there are no known non-generic attacks, and, in particular, the empirical hardness of basic security properties (such as inversion resistance and collision resistance) is similar to the corresponding hardness of a random oracle. Moreover, these hash functions are remarkably fast to compute because they involve only lightweight bit operations and are sometimes even available as single instructions on a processor.

Examples of hash functions used in practice as realizations of random oracles include SHA3 and BLAKE3. They come with different output sizes, corresponding to different choices of the output size σ of the random oracle. For example, SHA3-256 and SHA3-512 correspond to configurations of the SHA3 function to achieve $\sigma = 256$ and $\sigma = 512$. In the case of SHA3-256, on an x86 architecture, one can hash in less than 15 clock cycles per byte.

Achieving secure and efficient cryptographic hash functions has been a focus of cryptography for decades. This has resulted in substantial expertise and efforts in the design and analysis of cryptographic hash functions, providing confidence in their security.

- *Design principles.* While security properties of cryptographic hash functions cannot typically be reduced to simpler hardness assumptions, researchers have developed a rich cryptanalytical tool kit that offers design principles for cryptographic hash functions that provide security against large classes of attacks.
- *Demonstrated track record.* Cryptographic hash functions are critical building blocks in numerous real-world systems. These include secure communication protocols such as SSL/TLS, secure command execution such as SSH, secure network protocols such as IPsec, and cryptocurrencies such as Bitcoin. All these high-value targets provide substantial motivation to break the cryptographic hash functions used in these systems, but they have not been broken.
- *Community focus.* The cryptography community recognizes that the construction and analysis of cryptographic hash functions is arguably one of the most fundamental goals in cryptography. Over the years, this area has received considerable attention from numerous experts.

- *Priority for multiple stakeholders.* Academia, industry, and the government all recognize that cryptographic hash functions play a crucial role in the security of modern digital infrastructure. Substantial resources and time have been invested into coordinating knowledge from all these stakeholders in order to find the best cryptographic hash functions. For instance, the National Institute of Standards and Technology (NIST) organized an open competition to select the SHA3 standard for hash functions.

2.4.2 Alternative methods

The ROM heuristic described above is the standard approach to realize the ROM in practice. There are, however, alternative realizations of the ROM, such as: (1) trusted parties; (2) hardware realizations; and (3) software realizations. Some of these realizations rely on pseudorandom functions as an ingredient, but pseudorandom functions alone are an insecure realization of the ROM (see Section 2.3). All of these alternative realizations suffer from limitations that make them significantly less useful in practice than the ROM heuristic. We view the discussion below as largely hypothetical, and its primary aim is a pedagogical comparison.

(1) Trusted parties Everyone can issue queries to a single central party, who uses lazy sampling (see Section 2.2) to answer each query. The central party is efficient since lazy sampling is efficient. However, the central party is a bottleneck for latency and throughput: every query to the central party incurs a delay (e.g., a network roundtrip) to receive the answer, and the central party must answer every query of every party. Moreover, everyone must trust the central party to follow lazy sampling, because there is no way to detect malicious behavior.

The throughput concerns can be somewhat mitigated by relying on multiple designated parties each answering queries via the same pseudorandom function (used for coordination among these parties), so that these parties are each responsible for answering a fraction of the queries. However, now everyone must trust each of the designated parties, exacerbating the trust concerns.

Alternatively, the trust concerns can be somewhat mitigated by relying on a designated committee who runs a secure computation protocol to answer each query. This would require trusting only that a certain number of the designated parties are honest, rather than all of them. However, the cost of secure computation protocols exacerbates the latency and throughput concerns.

Overall, all of the above approaches are arguably unrealistic. The trust and efficiency challenges of each approach are substantial for real-world deployments.

(2) Hardware realizations A *hardware token* aims to realize a (stateless or stateful) functionality while exposing only the input-output behavior. Hardware tokens naturally lead to the following realization of the ROM: (1) a trusted central party samples a pseudorandom function, and then initializes and issues hardware tokens as needed with this pseudorandom function; and (2) every party obtains a hardware token from the trusted central party, and uses the token in place of the random oracle. Crucially, the fact that the pseudorandom function is efficient to evaluate implies that the functionality in the hardware token is an efficient algorithm.

Querying a hardware token issued by the trusted central party, rather than directly querying the trusted central party as in the previous item, potentially resolves the latency and throughput concerns. Each query is locally answered by a hardware token that is not shared with others.

Provided that the security features of all hardware tokens hold, the security guarantee of the pseudorandom function ensures that the input-output behavior of all hardware tokens is computationally indistinguishable from that of a random oracle.

However, protecting the functionality's internal state except for the input-output behavior is a tall order for a hardware token. History has shown that, in general, security features of hardware can be overcome with enough equipment and resources (which may be available if the hardware token is used for an application that constitutes a sufficiently valuable target). Moreover, even if the security features did work, substantial concerns remain: we need a trusted issuer to distribute cards to all relevant parties, which presents substantial trust and logistical concerns.

(3) Software realizations The cryptographic notion of *software obfuscation* captures the security goal of transforming an algorithm into another functionally-equivalent algorithm whose description reveals no information beyond the algorithm's input-output behavior (in particular, hiding the description of the algorithm). In other words, the aim of software obfuscation is a software equivalent of a secure hardware token: providing the obfuscation of an algorithm should be “as good as” providing a secure hardware token whose functionality is the algorithm.

This suggests a natural candidate of a software realization of a random oracle: (1) a trusted central party samples a pseudorandom function, and publishes its obfuscation; and (2) every party uses the obfuscated pseudorandom function as a random oracle. The fact that the pseudorandom function is efficient to evaluate implies that its obfuscation is also efficient to evaluate.

While there is a trusted central party, the trusted central party acts once at setup time (to sample and obfuscate the pseudorandom function) and is not needed thereafter; this improves on prior approaches where the trusted central party is required to be always active.

The main issue with this approach is that it is likely impossible. The aforementioned security property is known as *virtual black-box obfuscation* (VBB obfuscation), and cryptographers have proved that (based on plausible assumptions) such obfuscation is impossible in general, as well as impossible for specific functionalities like pseudorandom functions.

Cryptographers have also formulated weaker security notions for software obfuscation (such as indistinguishability obfuscation) and achieved constructions that satisfy these notions based on plausible assumptions. These weaker security notions, however, do not generically realize the ROM and, at best, provide heuristic realizations of it. Moreover, known constructions are notoriously expensive (the obfuscated pseudorandom function, while efficient, is much more expensive to evaluate than the original pseudorandom function), exacerbating latency concerns.

We conclude by noting that the above discussion is unrelated to non-cryptographic notions of software obfuscation (with no security guarantees) that appear in software engineering. For example, many libraries offer “superficial obfuscation”, which makes released code harder to reverse engineer via a collection of automated methods that change variable names, change function names, rearrange pieces of code, and so on. As another example, “recreational obfuscation” is a fun practice that challenges participants in competitions to write code that realizes simple functionalities via unusual and creative code, as a brain teaser.

Remark 2.4.1. A less apparent limitation of realizations of the ROM that involve trusted parties or hardware is that the query-to-answer computation typically cannot be mapped to a plain computation (without oracle calls). This precludes applications where it is useful to represent ROM computations as plain computations, such as when we need to produce a succinct

argument about a computation that in turn involves the verification of succinct arguments (this is known as *recursive proof composition*). Because of this the ROM heuristic in Section 2.4.1, and more generally any software-only realization, is convenient for such applications.

2.5 Benefits and drawbacks

The ROM plays a central role in cryptography, especially so for protocols that are practically useful. On the other hand, the ROM has notable limitations with regard to security guarantees once the ROM is instantiated. We discuss these two aspects: the benefits and the drawbacks of the ROM.

Benefits

- *Mathematically simple.* The ROM requires understanding only the notion of a random function. There is no need to understand abstract algebra (including topics such as finite fields or cyclic groups or polynomial rings) that are often required to understand cryptographic protocols. For example, most of the material of this book consists of security properties and security analyses against bounded-query adversaries; they can all be understood with only basic mathematical background in logic and probability (see Chapter 1).
- *Concrete security.* Security analyses in the ROM are typically straightforward and achieve error bounds with small explicit constants; this enables setting parameters in practice to achieve specific security levels. Moreover, establishing lower bounds is often a tractable goal, and in many settings, the lower bounds show that known security analyses are essentially tight. For example, in Section 3.3 we prove that finding collisions in a random oracle is hard and also show that the obtained error bound is tight up to small constants.
- *Efficient realization.* The ROM can be heuristically realized via cryptographic hash functions, which results in highly-efficient constructions; see the discussion in Section 2.4.1.
- *Broad applicability.* Numerous cryptographic protocols used in practice rely on (a heuristic realization of) the ROM, either by itself or additionally relying on cryptographic assumptions. For example, the succinct arguments in this book are obtained in the ROM without cryptographic assumptions. Moreover, many important cryptographic primitives, such as public-key encryption, signatures, identity-based encryption, and fully homomorphic encryption, are obtained in the ROM with cryptographic assumptions. This broad applicability has made the ROM a central tool in cryptography for decades.
- *Quantum security.* Many cryptographic protocols that are proved secure in the ROM can also be proved secure in the *quantum ROM* (QROM), which is the natural extension of the ROM where the random function can be queried in superposition.³ The error bounds typically incur predictable changes due to known speedups of quantum algorithms. This is the case for the succinct arguments in this book. (However, this is not the case in general: there exist constructions in the ROM that are not secure in the QROM.)

³As long as the protocol does not separately rely on cryptography that is quantum insecure.

Moreover, many of the cryptographic hash functions used in practice are considered secure against quantum adversaries, including those typically used for the ROM heuristic (replace the random oracle via a concrete hash function as discussed in Section 2.4.1).

Hence, realizing a cryptographic protocol in the (Q)ROM via the ROM heuristic typically results in a fast implementation that is plausibly secure against quantum adversaries.

Drawbacks

- *Sometimes unrealizable.* There exist cryptographic protocols that: (i) can be proved secure in the ROM, but (ii) are provably insecure when the random oracle is replaced with *any* efficient function. (Similar statements hold for the QROM.) This means that the ROM heuristic described in Section 2.4.1 does not work for these cryptographic protocols. Therefore, in general, security in the ROM (or QROM) does not imply security when the random oracle is replaced via a cryptographic hash function.

Nevertheless, the cryptographic protocols whose security fails are somewhat contrived. Such results are not known for “natural” constructions, including the constructions discussed in this book. Security in the ROM is widely seen as strong evidence that natural protocols in the ROM remain secure when replacing the random oracle with a cryptographic hash function.

- *Limited applicability.* As discussed, the ROM, combined with cryptographic assumptions, has broad applicability. In contrast, the ROM without cryptographic assumptions (i.e., the query-bounded setting where adversaries are all-powerful except for a query bound) has limited applicability; for example, it does not suffice to achieve basic cryptographic goals such as key agreement, public-key encryption, oblivious transfer, and indistinguishability obfuscation.

Nevertheless, the ROM without cryptographic assumptions *does* suffice for the succinct arguments discussed in this book.

2.6 The setting of multiple random oracles

Sometimes it is useful to have multiple random oracles, possibly with different output sizes. We use the notation

$$(f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]})$$

for the process of independently sampling $f_i \leftarrow \mathcal{U}(\ell_i)$ for each $i \in [k]$. We use the notation

$$b \xleftarrow{\text{tr}} A^{(f_i)_{i \in [k]}}(a)$$

to denote the fact that the algorithm A , given input a and query access to $(f_i)_{i \in [k]}$, outputs b ; moreover, the list of query-answer pairs to the oracles is tr . Each query-answer pair in tr is uniquely associated to one of the oracles.⁴

The rest of this section explains how to avoid explicitly sampling multiple random oracles. Specifically, the random oracles $(f_i)_{i \in [k]} \in \mathcal{U}((\ell_i)_{i \in [k]})$ can be derived from a single random oracle $f \in \mathcal{U}(\sigma)$, by using the techniques of *domain separation* and *output extension*.

⁴Formally, tr is a list of triples (i, x, y) denoting that the query x to oracle f_i received answer y .

Domain separation Suppose, for example, that we wish to use two independent random oracles f_0 and f_1 both sampled from $\mathcal{U}(\sigma)$. We can derive these from one random oracle f sampled from $\mathcal{U}(\sigma)$ by defining f_0 and f_1 as follows:

$$f_0(x) := f((0, x)) \text{ and } f_1(x) := f((1, x)).$$

The different prefixes in the two definitions “separate” the domains of the two random oracles: no pair of queries x_0, x_1 respectively to f_0, f_1 results in the same query to f .

More generally, we can derive k independent random oracles $(f_i)_{i \in [k]}$ sampled from $\mathcal{U}(\sigma)$ from one random oracle f sampled from $\mathcal{U}(\sigma)$: for every $i \in [k]$, define $f_i(x) := f((i, x))$. Here i is understood to be a $\lceil \log_2 k \rceil$ -bit prefix for domain separation. In particular, for every two distinct $i, i' \in [k]$, no pair of queries $x_i, x_{i'}$ respectively to $f_i, f_{i'}$ results in the same query to f .

Sometimes we do not make the prefixes explicit, and instead we say that we derive, via domain separation, a certain number of random oracles from the sampled random oracle. The total number of derived oracles determines the size of the query prefix.

Extending the output size The output size of a random oracle f sampled from $\mathcal{U}(\sigma)$ is σ . We can derive a random oracle with output size ℓ that is different from σ . If $\ell < \sigma$ then we truncate query answers to the first ℓ bits: a query x is answered with the first ℓ bits of $f(x)$. If $\ell > \sigma$ then we rely on $m := \lceil \ell/\sigma \rceil$ oracles $(f_i)_{i \in [m]}$ derived from f via domain separation: a query x is answered with $f_1(x) \parallel \dots \parallel f_{m-1}(x) \parallel y$, where y are the first $\ell - \lceil \ell/\sigma \rceil$ bits of $f_m(x)$. (The last answer is truncated to the appropriate size, in the event that σ does not evenly divide ℓ .) Each query to the new “virtual” oracle with output size ℓ is translated to m queries to f .

Separation and extension More generally, we can derive k independent random oracles $(f_i)_{i \in [k]}$ with respective output sizes $(\ell_i)_{i \in [k]}$, starting from one random oracle f with output size σ . For every $i \in [k]$, we set $f_i := g^f(i, \ell_i)$ where $g^f(i, \ell_i)$ is defined as follows:

- $g^f(i, \ell_i)(x)$:
1. Set $m_i := \lceil \ell_i/\sigma \rceil$.
 2. For every $j \in [m_i]$, set $y_j := f((i, j, x))$.
 3. Truncate y_{m_i} to its first $\ell_i - \lceil \ell_i/\sigma \rceil$ bits.
 4. Output $y := y_1 \parallel \dots \parallel y_{m_i}$.

Above, i and j are $\lceil \log_2 k \rceil$ -bit and $\lceil \log_2 m_i \rceil$ -bit prefixes used for domain separation.

Converting algorithms We can convert algorithms from the setting of a single oracle to the setting of multiple oracles, and vice versa. The conversion is formally given by the procedures M2S (multi to single) and S2M (single to multi) defined and analyzed below. We use these procedures in Chapter 7 when reducing non-interactive arguments in the setting of multiple random oracles to non-interactive arguments in the setting of a single random oracle.

Construction 2.6.1. Let M be an algorithm that queries oracles $(f_i)_{i \in [k]} \in \mathcal{U}((\ell_i)_{i \in [k]})$. Then M2S(M) defined below queries a single oracle $f \in \mathcal{U}(\sigma)$.

- M2S(M) $^f(a)$:
1. Compute the output $b \leftarrow M^{(g^f(i, \ell_i))_{i \in [k]}}(a)$.
 2. Output b .

For every $i \in [k]$, each query to f_i is a query to $g^f(i, \ell_i)$, which in turn corresponds to $\lceil \ell_i/\sigma \rceil$ queries to f . Hence if M makes t queries to $(f_i)_{i \in [k]}$ then $\text{M2S}(M)$ makes $(\max_{i \in [k]} \lceil \ell_i/\sigma \rceil) \cdot t$ queries to f . Moreover, the conversion is “local” in that M2S individually maps each query to a set of queries. Finally, M2S does not affect the algorithm’s output distribution, as we show next.

Lemma 2.6.2. *Let M be an algorithm that queries oracles $(f_i)_{i \in [k]} \in \mathcal{U}((\ell_i)_{i \in [k]})$. Then the following two distributions are identical:*

$$\left\{ b \mid \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]}) \\ b \leftarrow M^{(f_i)_{i \in [k]}} \end{array} \right\} \text{ and } \left\{ b \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ b \leftarrow \text{M2S}(M)^f \end{array} \right\}.$$

Proof. The lemma follows from the fact the queries of M in the left-side experiment (which are answered via $(f_i)_{i \in [k]}$) and in the right-side experiment (which are answered via $(g^f(i, \ell_i))_{i \in [k]}$) receive identically distributed answers, which implies that M ’s output is identically distributed in the two experiments. Indeed, suppose that M performs a query x to the i -th oracle.

- *Case 1: no prior query to the i -th oracle equals x .* In the left-side experiment M receives as answer $f_i(x)$, which is a random string of ℓ_i bits. In the right-side experiment M receives as answer $g^f(i, \ell_i)(x)$, which also is a random string of ℓ_i bits. This latter holds because, due to domain separation, f was not previously queried at any of $((i, j, x))_{j \in [m_i]}$.
- *Case 2: a prior query to the i -th oracle equals x .* In this case both the left-side experiment and the right-side experiment answer consistently (with $f_i(x)$ and $g^f(i, \ell_i)(x)$ respectively).

□

Next we define and analyze the “reverse” conversion S2M .

Construction 2.6.3. Let M be an algorithm that queries a single oracle $f \in \mathcal{U}(\sigma)$. Then $\text{S2M}(M)$ defined below queries oracles $(f_i)_{i \in [k]} \in \mathcal{U}((\ell_i)_{i \in [k]})$.

$\text{S2M}(M)^{(f_i)_{i \in [k]}}(a)$:

1. Lazily sample an oracle $\dot{g}_\perp \leftarrow \mathcal{U}(\sigma)$ (to answer malformed queries).
2. For every $i \in [k]$, lazily sample an oracle $\dot{g}_i \leftarrow \mathcal{U}(\ell_i - \lceil \ell_i/\sigma \rceil)$ (to pad randomness when ℓ_i is not a multiple of σ).
3. Simulate $M(a)$ while answering each query x as follows.
 - a) If x has the form (i, j, x') with $i \in [k]$ and $j \in [m_i]$ where $m_i = \lceil \ell_i/\sigma \rceil$ then:
 - i. query f_i at x' to obtain y ;
 - ii. parse y into blocks y_1, \dots, y_{m_i} each of σ bits, except for y_{m_i} of $\ell_i - \lceil \ell_i/\sigma \rceil$ bits;
 - iii. pad y_{m_i} to σ bits using $\dot{g}_i(x')$;
 - iv. return y_j .
 - b) Otherwise, answer with $y := \dot{g}_\perp(x)$.
4. Output the output b produced by $M(a)$.

Each query to f is mapped to at most one query to $(f_i)_{i \in [k]}$. (Some queries are answered with internal randomness of S2M .) Hence, if M makes at most t queries to f then $\text{S2M}(M)$ makes at most t queries to $(f_i)_{i \in [k]}$.

We could prove a statement analogous to Lemma 2.6.2, saying that the conversion performed by S2M does not change an algorithm's output distribution. We prove a stronger statement.

We consider a process where an algorithm M_1 produces an output b_1 and then another algorithm M_2 receives as input b_1 and produces another output b_2 . However, the two algorithms M_1 and M_2 are in different settings and so they cannot be used as is: M_1 is in the setting of a single oracle and M_2 is in the setting of multiple oracles. In order for them to be in the same setting, we can either apply S2M to M_1 , or M2S to M_2 . Lemma 2.6.4 below states that either option leads to the same joint distribution of (b_1, b_2) . After that, in Lemma 2.6.5, we prove an analogous lemma with M_1 in the setting of multiple oracles and M_2 in the setting of a single oracle.

Lemma 2.6.4. *Let M_1 be an algorithm that queries an oracle $f \in \mathcal{U}(\sigma)$, and let M_2 be an algorithm that queries oracles $(f_i)_{i \in [k]} \in \mathcal{U}((\ell_i)_{i \in [k]})$. Then the following two distributions are identical:*

$$\left\{ (b_1, b_2) \middle| \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]}) \\ b_1 \leftarrow \text{S2M}(M_1)^{(f_i)_{i \in [k]}} \\ b_2 \leftarrow M_2^{(f_i)_{i \in [k]}}(b_1) \end{array} \right\} \text{ and } \left\{ (b_1, b_2) \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ b_1 \leftarrow M_1^f \\ b_2 \leftarrow \text{M2S}(M_2)^f(b_1) \end{array} \right\}.$$

Proof. We proceed in two steps. First, we apply Lemma 2.6.2 to the left-side distribution. Since M2S is local (it separately transforms each query without sharing state), we can separately apply it to both lines of the left-side experiment. We deduce that the following two distributions are identical:

$$\left\{ (b_1, b_2) \middle| \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]}) \\ b_1 \leftarrow \text{S2M}(M_1)^{(f_i)_{i \in [k]}} \\ b_2 \leftarrow M_2^{(f_i)_{i \in [k]}}(b_1) \end{array} \right\} \text{ and } \left\{ (b_1, b_2) \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ b_1 \leftarrow \text{M2S}(\text{S2M}(M_1))^f \\ b_2 \leftarrow \text{M2S}(M_2)^f(b_1) \end{array} \right\}.$$

The second step is to argue that the following two distributions are equivalent (concluding the proof):

$$\left\{ (b_1, b_2) \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ b_1 \leftarrow \text{M2S}(\text{S2M}(M_1))^f \\ b_2 \leftarrow \text{M2S}(M_2)^f(b_1) \end{array} \right\} \text{ and } \left\{ (b_1, b_2) \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ b_1 \leftarrow M_1^f \\ b_2 \leftarrow \text{M2S}(M_2)^f(b_1) \end{array} \right\}.$$

Consider a query x to f performed by M_1 .

- *Case 1: x has the form (i, j, x') with $i \in [k]$ and $j \in [m_i]$ where $m_i = \lceil \ell_i / \sigma \rceil$. $\text{S2M}(M_1)$ maps M_1 's query x to the query x' for f_i , obtains the ℓ_i -bit answer y , and returns the j -th σ -bit chunk y_j of y . (If σ does not evenly divide ℓ_i then y_{m_i} is padded to σ bits using $g_i(x')$.) Then $\text{M2S}(\text{S2M}(M_1))$ maps $\text{S2M}(M_1)$'s query x' for f_i back to the query x for f .*
- *Case 2: x does not have the form (i, j, x') as above. $\text{S2M}(M_1)$ uses internal randomness to simulate the answer to x as $g_\perp(x)$, which means that $\text{M2S}(\text{S2M}(M_1))$ does not make a corresponding query to f .*

In both cases, the answers to the queries of M_1 and $\text{M2S}(\text{S2M}(M_1))$ are identically distributed, which means that b_1 is identically distributed in both experiments.

This alone does *not* suffice to conclude that (b_1, b_2) are identically distributed in both experiments. In the left-side experiment malformed queries of M_1 are answered internally;

whereas in the right-side experiment malformed queries of M_1 are answered by f (like all other queries). In principle this could cause the two distributions to differ because $\text{M2S}(M_2)$, which receives as input b_1 , may detect the inconsistency by making a malformed query to x . However, since $\text{M2S}(M_2)$ performs only well-formed queries, we can conclude that b_2 has the same distribution as well. \square

Lemma 2.6.5. *Let M_1 be an algorithm that queries oracles $(f_i)_{i \in [k]} \in \mathcal{U}((\ell_i)_{i \in [k]})$ and let M_2 be an algorithm that queries an oracle $f \in \mathcal{U}(\sigma)$. Then the following two distributions are identical:*

$$\left\{ (b_1, b_2) \middle| \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]}) \\ b_1 \leftarrow M_1^{(f_i)_{i \in [k]}} \\ b_2 \leftarrow \text{S2M}(M_2)^{(f_i)_{i \in [k]}}(b_1) \end{array} \right\} \text{ and } \left\{ (b_1, b_2) \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ b_1 \leftarrow \text{M2S}(M_1)^f \\ b_2 \leftarrow M_2^f(b_1) \end{array} \right\}.$$

Proof. We proceed in two steps. First, we apply Lemma 2.6.2 to the left-side distribution. Since M2S is local (it separately transforms each query without sharing state), we can separately apply it to both lines of the left-side experiment. We deduce that the following two distributions are identical:

$$\left\{ (b_1, b_2) \middle| \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]}) \\ b_1 \leftarrow M_1^{(f_i)_{i \in [k]}} \\ b_2 \leftarrow \text{S2M}(M_2)^{(f_i)_{i \in [k]}}(b_1) \end{array} \right\} \text{ and } \left\{ (b_1, b_2) \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ b_1 \leftarrow \text{M2S}(M_1)^f \\ b_2 \leftarrow \text{M2S}(\text{S2M}(M_2))^f(b_1) \end{array} \right\}.$$

The second step is to argue that the following two distributions are equivalent (concluding the proof):

$$\left\{ (b_1, b_2) \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ b_1 \leftarrow \text{M2S}(M_1)^f \\ b_2 \leftarrow \text{M2S}(\text{S2M}(M_2))^f(b_1) \end{array} \right\} \text{ and } \left\{ (b_1, b_2) \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ b_1 \leftarrow \text{M2S}(M_1)^f \\ b_2 \leftarrow M_2^f(b_1) \end{array} \right\}.$$

Consider a query x to f performed by M_2 .

- *Case 1: x has the form (i, j, x') with $i \in [k]$ and $j \in [m_i]$ where $m_i = \lceil \ell_i / \sigma \rceil$. $\text{S2M}(M_2)$ maps M_1 's query x to the query x' for f_i , obtains the ℓ_i -bit answer y , and returns the j -th σ -bit chunk y_j of y . (If σ does not evenly divide ℓ_i then y_{m_i} is padded to σ bits using $\dot{g}_i(x')$.) Then $\text{M2S}(\text{S2M}(M_2))$ maps $\text{S2M}(M_2)$'s query x' for f_i back to the query x for f .*
- *Case 2: x does not have the form (i, j, x') as above. $\text{S2M}(M_2)$ uses internal randomness to simulate the answer to x as $\dot{g}_{\perp}(x)$, which means that $\text{M2S}(\text{S2M}(M_2))$ does not make a corresponding query to f . Since we know that $\text{M2S}(M_1)$ does not perform such query, the answer is both distributions is uniformly random.*

In both cases, the answers to the queries of M_2 and $\text{M2S}(\text{S2M}(M_2))$ are identically distributed, which means that b_1 , and b_2 is identically distributed in both experiments. \square

3 Basic cryptographic properties

We discuss basic cryptographic properties of a random oracle:

- *unpredictability* (Section 3.1);
- *inversion resistance* (Section 3.2); and
- *collision resistance* (Section 3.3).

These are useful to gain familiarity with basic security proofs in the random oracle model (ROM), and we rely on them throughout this book to establish security properties of various constructions.

3.1 Unpredictability

An algorithm that queries a random oracle f learns a query-answer pair with each new query to f . Predicting the answers to queries that have not been made is not possible (with probability better than random guessing), as we now discuss.

Consider the following **prediction problem**: *given query access to a random oracle $f \leftarrow \mathcal{U}(\sigma)$, output (x, y) such that $f(x) = y$ but (x, y) is not a query-answer pair resulting from querying f .*

By definition of a random oracle f sampled from $\mathcal{U}(\sigma)$, any new query to f is answered via a random string in $\{0, 1\}^\sigma$, regardless of any other prior queries. This tells us that the success probability in the prediction problem is at most $\frac{1}{2^\sigma}$, regardless of the resources (in computation or in queries to the random oracle) of the attacker. Below we formally state and prove this fact.

Lemma 3.1.1. *For every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, and t -query algorithm A that outputs a candidate query-answer pair (x, y) ,*

$$\Pr \left[\begin{array}{l} (x, y) \notin \text{tr} \\ \wedge f(x) = y \\ \wedge (x, y) \xleftarrow{\text{tr}} A^f \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (x, y) \xleftarrow{\text{tr}} A^f \end{array} \right] \leq \frac{1}{2^\sigma}.$$

Proof. Let E be the event that A queries f at x . We distinguish two disjoint cases, which together yield the claimed bound.

- *Case 1: A queries x .* If A queries f at x and $(x, y) \notin \text{tr}$ then it must be that $f(x) \neq y$. Hence

$$\Pr \left[\begin{array}{l} (x, y) \notin \text{tr} \\ \wedge f(x) = y \\ \wedge E \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (x, y) \xleftarrow{\text{tr}} A^f \end{array} \right] = 0.$$

- *Case 2: A does not query x .* If A does not query f at x (in particular, $(x, y) \notin \text{tr}$) then the value of $f(x)$ is independent of tr . Hence $f(x)$ is independent of y , because y is a random variable that depends only on tr (and possibly internal randomness of A). Hence we can write

$$\Pr \left[\begin{array}{l} (x, y) \notin \text{tr} \\ \wedge f(x) = y \\ \wedge \overline{E} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (x, y) \xleftarrow{\text{tr}} A^f \end{array} \right] = \Pr \left[\begin{array}{l} y' = y \\ \mid f \leftarrow \mathcal{U}(\sigma) \\ \mid (x, y) \xleftarrow{\text{tr}} A^f \\ \mid y' \leftarrow \{0, 1\}^\sigma \end{array} \right] = \frac{1}{2^\sigma}.$$

□

3.2 Inversion resistance

Consider the following **inversion problem**: *given query access to a random oracle $f \leftarrow \mathcal{U}(\sigma)$ and given a target $y \in \{0, 1\}^\sigma$, find an input $x \in \{0, 1\}^*$ such that $f(x) = y$.*

Since any query to the random oracle is answered via a random string in $\{0, 1\}^\sigma$, there is a natural “attack” to this problem: given a budget of t queries, make t arbitrary distinct queries x_1, \dots, x_t and output any query answered by y ; else abort if none is found. The probability that any single query has answer y is exactly $\frac{1}{2^\sigma}$, and there are t such queries. By the inclusion-exclusion principle (Lemma 1.2.3), the probability of inverting the target y can be lower bounded as follows:

$$\begin{aligned} & \Pr[\exists i \in [t] : f(x_i) = y] \\ & \geq \sum_{i \in [t]} \Pr[f(x_i) = y] - \sum_{\substack{i, j \in [t] \\ \text{with } i \neq j}} \Pr[f(x_i) = y \wedge f(x_j) = y] \\ & = \frac{t}{2^\sigma} - \binom{t}{2} \cdot \frac{1}{2^{2\sigma}} \\ & \geq \frac{1}{2} \cdot \frac{t}{2^\sigma}. \end{aligned}$$

The above success probability grows linearly in t , which in particular confirms the intuition that more queries to the random oracle lead to a higher inversion probability. Nevertheless, the success probability is small, in the sense that if the query bound t is some polynomial in σ , then the success probability is a negligible function of the output size σ of the random oracle.

The lemma below implies that the above strategy is essentially optimal: the probability that *any* t -query algorithm makes a query that maps to y is upper bounded by $O(\frac{t}{2^\sigma})$. This tells us that a random oracle is **inversion resistant** (cryptographers sometimes refer to such properties as *one-wayness*).

Lemma 3.2.1. *For every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, t -query algorithm A , and target $y \in \{0, 1\}^\sigma$,*

$$\Pr \left[\begin{array}{l} \exists \text{ query } x \text{ in } \text{tr} \\ \text{such that } f(x) = y \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \perp \xleftarrow{\text{tr}} A^f \end{array} \right] \leq \frac{t}{2^\sigma}.$$

Proof. Assume, without loss of generality, that A ’s queries to the random oracle are distinct. For every $i \in [t]$, let X_i be the indicator random variable for the event that the i -th query by the algorithm A maps to y . It holds that $\Pr[X_i = 1] = \frac{1}{2^\sigma}$, because the response to a query (that has not been made before) is a random value in $\{0, 1\}^\sigma$, regardless of any other queries performed. By a union bound,

$$\begin{aligned} & \Pr \left[\begin{array}{l} \exists \text{ query } x \text{ in } \text{tr} \\ \text{such that } f(x) = y \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \perp \xleftarrow{\text{tr}} A^f \end{array} \right] \\ & = \Pr[\exists i \in [t] \text{ such that } X_i = 1] \end{aligned}$$

$$\leq \sum_{i \in [t]} \Pr[X_i = 1] = \sum_{i \in [t]} \frac{1}{2^\sigma} = \frac{t}{2^\sigma}.$$

□

The above statement directly extends to when the target y is sampled according to any distribution over $\{0, 1\}^\sigma$, or is chosen by the adversary in a precomputation step without access to the random oracle. This follows by the convexity of probability distributions, as the upper bound holds individually for each target y in the support of the distribution.

But what happens if the target y is chosen based on the random oracle? In this case the adversary could choose y to be $f(x)$ for some fixed x , in which case it has (trivially) found a query that maps to y . We can nevertheless consider the case where x is random (independent of the random oracle) *but not known to the adversary*. In the statement below we upper bound the probability that an adversary inverts a target y that is an answer to a random n -bit query x .

Lemma 3.2.2. *For every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, t -query algorithm A , and parameter $n \in \mathbb{N}$,*

$$\Pr \left[(x, y) \in \text{tr} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ x \leftarrow \{0, 1\}^n \\ y \leftarrow f(x) \\ \perp \xleftarrow{\text{tr}} A^f(y) \end{array} \right] \leq \frac{t}{2^n}.$$

Proof. Since $y = f(x)$ is random in $\{0, 1\}^\sigma$, we can modify the experiment (the right side of the probability statement) with an experiment where y is sampled from $\{0, 1\}^\sigma$ and x is sampled from $\{0, 1\}^n$ *after* the adversary's computation. The probability that x equals any of the queries in the adversary's trace tr is at most $\frac{t}{2^n}$. That is,

$$\Pr \left[(x, y) \in \text{tr} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ x \leftarrow \{0, 1\}^n \\ y \leftarrow f(x) \\ \perp \xleftarrow{\text{tr}} A^f(y) \end{array} \right] = \Pr \left[(x, y) \in \text{tr} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ y \leftarrow \{0, 1\}^\sigma \\ \perp \xleftarrow{\text{tr}} A^f(y) \\ x \leftarrow \{0, 1\}^n \end{array} \right] \leq \frac{t}{2^n}.$$

□

We extend the above lemma to consider the setting where an adversary is tasked with inverting every target in a list y_1, \dots, y_k , where each target y_i is with respect to a different random oracle f_i .

Lemma 3.2.3. *For every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, t -query algorithm A , parameter $n \in \mathbb{N}$, and number of targets $k \in \mathbb{N}$,*

$$\Pr \left[\forall i \in [k], (x_i, y_i) \in \text{tr} \middle| \begin{array}{l} \forall i \in [k] : \\ \bullet f_i \leftarrow \mathcal{U}(\sigma) \\ \bullet x_i \leftarrow \{0, 1\}^n \\ \bullet y_i \leftarrow f_i(x_i) \\ \perp \xleftarrow{\text{tr}} A^{f_1, \dots, f_k}(y_1, \dots, y_k) \end{array} \right] \leq \left(\frac{t}{2^n} \right)^k.$$

Proof. We follow the approach of the proof of Lemma 3.2.2. For every $i \in [k]$, $y_i = f_i(x_i)$ is random in $\{0, 1\}^\sigma$, and thus we can modify the experiment (the right side of the probability statement) with an experiment where y_i is sampled from $\{0, 1\}^\sigma$ and x_i is sampled from $\{0, 1\}^n$ after the adversary's computation. The probability that x_i equals any of the queries in the adversary's trace tr is at most $\frac{t}{2^n}$, and is independent from the event that $x_{i'}$ equals any of the queries in the adversary's trace tr , for any $i' \neq i$. Therefore,

$$\begin{aligned} & \Pr \left[\begin{array}{l} \forall i \in [k], \\ (x_i, y_i) \in \text{tr} \end{array} \middle| \begin{array}{l} \forall i \in [k] : \\ \bullet f_i \leftarrow \mathcal{U}(\sigma) \\ \bullet x_i \leftarrow \{0, 1\}^n \\ \bullet y_i \leftarrow f_i(x_i) \\ \perp \xleftarrow{\text{tr}} A^{f_1, \dots, f_k}(y_1, \dots, y_k) \end{array} \right] \\ &= \Pr \left[\begin{array}{l} \forall i \in [k], \\ (x_i, y_i) \in \text{tr} \end{array} \middle| \begin{array}{l} \forall i \in [k] : \\ \bullet f_i \leftarrow \mathcal{U}(\sigma) \\ \bullet y_i \leftarrow \{0, 1\}^\sigma \\ \perp \xleftarrow{\text{tr}} A^{f_1, \dots, f_k}(y_1, \dots, y_k) \\ \forall i \in [k], x_i \leftarrow \{0, 1\}^n \end{array} \right] \\ &\leq \left(\frac{t}{2^n} \right)^k. \end{aligned}$$

□

3.3 Collision resistance

Consider the following **collision problem**: given query access to a random oracle $f \leftarrow \mathcal{U}(\sigma)$, find distinct x and x' in $\{0, 1\}^*$ such that $f(x) = f(x')$.

Like the inversion problem, there is a natural “attack” to the collision problem: given a query bound t queries, make t arbitrary distinct queries to the random oracle and output any collision induced by these queries (else give up). Unlike the inversion problem, the success probability of this strategy is qualitatively different, as we now explain.

The probability that a single query creates a collision depends on the number of prior queries, because the query can collide with any of the previous queries. For every two queries x and x' with $x \neq x'$, the probability that $f(x) = f(x')$ is $\frac{1}{2^\sigma}$ (as for any value given to $f(x) = y$ the probability that $f(x') = y$ is $\frac{1}{2^\sigma}$). Since there are $\binom{t}{2}$ such pairs when the attacker makes t distinct queries, the success probability of this attack is $\Omega(\frac{t^2}{2^\sigma})$ (see Claim 3.3.2).

In other words, the above success probability grows *quadratically* in t , which means that the collision problem is “easier” than the inversion problem. In order to get a constant probability of success (say $1/2$), the number of queries required by the adversary is roughly the square root of the number of possible outputs of the random oracle. This phenomenon is known as the “birthday paradox”. That said, the quadratic growth does not change the fact that the success probability is small (i.e., if the query bound t is some polynomial in σ then the success probability is a negligible function in the output size σ of the random oracle).

We prove that the above strategy is essentially optimal: the probability that *any* t -query algorithm finds a collision is at most $O(t^2/2^\sigma)$. Namely, a random oracle is **collision resistant**.

Lemma 3.3.1. *For every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, and t -query algorithm A ,*

$$\Pr \left[\begin{array}{c} \exists (x, y), (x', y) \in \text{tr} \\ \text{with } x \neq x' \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \perp \xleftarrow{\text{tr}} A^f \end{array} \right] \leq \frac{1}{2} \cdot \frac{(t-1) \cdot t}{2^\sigma}.$$

Proof. No 1-query algorithm can satisfy the condition in the probability statement because the condition involves two distinct query-answer pairs in the trace. So suppose that $t > 1$. Denote by x_1, \dots, x_t the queries made by A to the random oracle f . For every $i, j \in [t]$ with $i \neq j$, let $E_{i,j}$ be the event that $x_i \neq x_j$ and $f(x_i) = f(x_j)$. The condition “there exist $(x, y), (x', y) \in \text{tr}$ with $x \neq x'$ ” is equivalent to $\vee_{i,j \in [t], i \neq j} E_{i,j}$. Moreover, $\Pr[E_{i,j}] \leq \frac{1}{2^\sigma}$. Indeed, if $x_i = x_j$ then $E_{i,j}$ does not hold; else if $x_i \neq x_j$ then $E_{i,j}$ holds with probability exactly $\frac{1}{2^\sigma}$. We conclude the proof via a union bound:

$$\begin{aligned} & \Pr \left[\begin{array}{c} \exists (x, y), (x', y) \in \text{tr} \\ \text{with } x \neq x' \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \perp \xleftarrow{\text{tr}} A^f \end{array} \right] \\ &= \Pr \left[\vee_{i,j \in [t], i \neq j} E_{i,j} \right] \\ &\leq \sum_{i,j \in [t], i \neq j} \Pr[E_{i,j}] \leq \binom{t}{2} \cdot \frac{1}{2^\sigma} = \frac{1}{2} \cdot \frac{(t-1) \cdot t}{2^\sigma}. \end{aligned}$$

□

The expression in Lemma 3.3.1 is tight up to a constant factor.

Claim 3.3.2. *For every output size $\sigma \in \mathbb{N}$ of the random oracle and query bound $t \in \mathbb{N}$ with $t > 1$ and $\frac{t^2}{2^\sigma} \leq 1/4$, there exists a t -query algorithm A such that*

$$\Pr \left[\begin{array}{c} \exists (x, y), (x', y) \in \text{tr} \\ \text{with } x \neq x' \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \perp \xleftarrow{\text{tr}} A^f \end{array} \right] \geq \frac{1}{10} \cdot \frac{t^2}{2^\sigma}.$$

Proof. The algorithm A performs any arbitrary sequence of distinct queries x_1, \dots, x_t . For every $i, j \in [t]$ with $i \neq j$, let $E_{i,j}$ be the event that $x_i \neq x_j$ and $f(x_i) = f(x_j)$; the same as in the proof of Lemma 3.3.1. (Such events can be defined since $t > 1$.) Since A makes distinct queries, $\Pr[E_{i,j}] = \frac{1}{2^\sigma}$. By the inclusion-exclusion principle (Lemma 1.2.3), we can lower bound the success probability of A as follows:

$$\begin{aligned} & \Pr \left[\begin{array}{c} \exists (x, y), (x', y) \in \text{tr} \\ \text{with } x \neq x' \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \perp \xleftarrow{\text{tr}} A^f \end{array} \right] \\ &= \Pr \left[\vee_{i,j \in [t], i \neq j} E_{i,j} \right] \\ &\geq \sum_{i,j \in [t], i \neq j} \Pr[E_{i,j}] - \sum_{\text{distinct } i,j,k \in [t]} \Pr[E_{i,j} \wedge E_{j,k}] - \sum_{\text{distinct } i,j,k,\ell \in [t]} \Pr[E_{i,j} \wedge E_{k,\ell}] \\ &= \binom{t}{2} \cdot \frac{1}{2^\sigma} - \binom{t}{3} \cdot \frac{1}{2^{2\sigma}} - \binom{t}{4} \cdot \frac{1}{2^{2\sigma}} \\ &\geq \frac{t^2}{4} \cdot \frac{1}{2^\sigma} - \frac{e^3 \cdot t^3}{3^3} \cdot \frac{1}{2^{2\sigma}} - \frac{e^4 \cdot t^4}{4^4} \cdot \frac{1}{2^{2\sigma}} \quad (\text{by Lemma 1.4.1}) \end{aligned}$$

$$\begin{aligned} &\geq \frac{t^2}{4} \cdot \frac{1}{2^\sigma} - \frac{6}{10} \cdot \frac{t^4}{2^{2\sigma}} \\ &\geq \frac{1}{10} \cdot \frac{t^2}{2^\sigma} \quad (\text{since } \frac{t^2}{2^\sigma} \leq 1/4). \end{aligned}$$

□

4 Arguments in the ROM

We introduce the basic definitions of arguments in the random oracle model (ROM).

- In Section 4.1 we introduce the main object of study in this book: *non-interactive arguments* (NARGs) in the ROM.
- In Section 4.2 we introduce *interactive arguments* (IARGs) in the ROM. These serve as an intermediate notion in order to build intuition towards non-interactive counterparts.

Later in Chapter 5 we discuss additional security definitions of NARGs that are useful in applications: *knowledge soundness* and *zero knowledge*.

4.1 Non-interactive arguments

A *non-interactive argument* (NARG) in the ROM is a tuple

$$\text{NARG} = (\mathcal{P}, \mathcal{V})$$

where \mathcal{P} is an oracle algorithm known as the *argument prover* and \mathcal{V} is an oracle algorithm known as the *argument verifier*.

A random oracle f is sampled from the distribution $\mathcal{U}(\sigma)$, for a given output size $\sigma \in \mathbb{N}$. Anyone, including the argument prover \mathcal{P} and the argument verifier \mathcal{V} , can query f . The argument prover \mathcal{P} receives as input an instance \mathbf{x} and witness \mathbf{w} , and outputs an argument string π . The argument verifier \mathcal{V} receives as input the instance \mathbf{x} and argument string π , and outputs a bit denoting whether to accept (the bit is 1) or reject (the bit is 0). See Figure 4.1a for a diagram.

We say that $\text{NARG} = (\mathcal{P}, \mathcal{V})$ is a non-interactive argument in the ROM for a relation \mathcal{R} if it satisfies two main properties: *completeness* and *soundness*.

- **Completeness.** Informally, for every instance-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, the (honest) argument prover \mathcal{P} on input (\mathbf{x}, \mathbf{w}) outputs an argument string π such that the argument verifier \mathcal{V} on input (\mathbf{x}, π) accepts, with probability 1. The probability here is taken over the choice of random oracle f , as well as any randomness of \mathcal{P} and of \mathcal{V} .

See Definition 4.1.1 below for a formal statement.

- **Soundness.** Informally, every malicious argument prover $\tilde{\mathcal{P}}$ convinces the (honest) argument verifier \mathcal{V} to accept an instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ with at most a small error probability that we denote by ϵ_{ARG} (the probability is over the choice of random oracle f). The malicious argument prover $\tilde{\mathcal{P}}$ is computationally unbounded but may make at most t queries to the random oracle, for a given query bound $t \in \mathbb{N}$. In general, the error probability ϵ_{ARG} depends on the query bound t , in addition to the output size σ of the random oracle.

An important technical detail for soundness is how the instance \mathbf{x} is chosen. The notion of *non-adaptive soundness* captures the case where \mathbf{x} is fixed before the random oracle is

sampled; see Definition 4.1.2 below for a formal statement. This definition is too weak for many applications. Indeed, a random oracle is typically sampled once and then used to prove multiple statements. These statements may be chosen by an adversary depending on the random oracle. Non-adaptive soundness provides no security guarantees in this case.

The notion of *adaptive soundness* captures the case where the malicious argument prover $\tilde{\mathcal{P}}$ chooses the instance \mathbf{x} adaptively after interacting with the random oracle; see Definition 4.1.3 below for a formal statement.

All constructions of NARGs that we study satisfy the stronger notion of adaptive soundness. We discuss the notion of non-adaptive soundness mainly for pedagogical purposes: its simpler definition serves as a helpful warmup step before establishing adaptive soundness.

Below we provide the definitions of completeness, non-adaptive soundness, and adaptive soundness.

Definition 4.1.1: completeness

A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has **(perfect) completeness** if for every output size $\sigma \in \mathbb{N}$ of the random oracle and instance-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$,

$$\Pr \left[\mathcal{V}^f(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \mathcal{P}^f(\mathbf{x}, \mathbf{w}) \end{array} \right] = 1.$$

The probability is taken over f and any randomness of the argument prover \mathcal{P} and verifier \mathcal{V} .

Definition 4.1.2: non-adaptive soundness

A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has **non-adaptive soundness error** ϵ_{ARG} if for every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, t -query malicious argument prover $\tilde{\mathcal{P}}$, and instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$,

$$\Pr \left[\mathcal{V}^f(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \leq \epsilon_{\text{ARG}}(\sigma, t, \mathbf{x}).$$

The probability is taken over f and any randomness of the argument verifier \mathcal{V} .

Definition 4.1.3: adaptive soundness

A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has **adaptive soundness error** ϵ_{ARG} if for every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$,

t -query malicious argument prover $\tilde{\mathcal{P}}$, and instance size bound $n \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \leq \epsilon_{\text{ARG}}(\sigma, t, n).$$

The probability is taken over f and any randomness of the argument verifier \mathcal{V} .

Remark 4.1.4 (randomness of the argument prover). There is no need for the malicious argument prover in Definitions 4.1.2 and 4.1.3 to be probabilistic: the malicious argument prover is all-powerful and so can use the best choice of any randomness to maximize the acceptance probability of the argument verifier. In contrast, the honest argument prover's randomness is useful to achieve properties such as zero knowledge (discussed in Section 5.2), and so this randomness appears in the completeness definition (Definition 4.1.1).

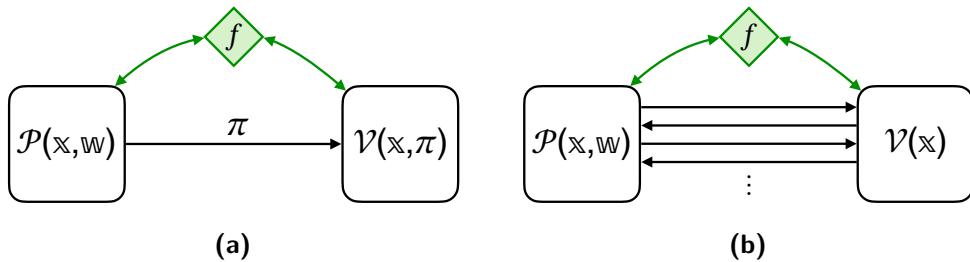


Figure 4.1: Diagram of a non-interactive argument (left) and an interactive argument (right), in the random oracle model.

4.2 Interactive arguments

The notion of a non-interactive argument can be relaxed to allow the argument prover and argument verifier to interact over some number of rounds. We introduce this relaxation because in this book we sometimes study simpler interactive protocols before studying non-interactive counterparts.

An *interactive argument* (IARG) in the ROM is a tuple

$$\text{IARG} = (\mathcal{P}, \mathcal{V})$$

where \mathcal{P} is an oracle interactive algorithm known as the *argument prover* and \mathcal{V} is an oracle interactive algorithm known as the *argument verifier*.

A random oracle f is sampled from the distribution $\mathcal{U}(\sigma)$, for a given output size $\sigma \in \mathbb{N}$. Anyone, including the argument prover \mathcal{P} and the argument verifier \mathcal{V} , can query f . The argument prover \mathcal{P} receives as input an instance \mathbf{x} and witness \mathbf{w} , and the argument verifier \mathcal{V} receives as input the instance \mathbf{x} . The argument prover \mathcal{P} and argument verifier \mathcal{V} interact over several rounds, and after this interaction the argument verifier \mathcal{V} outputs a bit denoting whether to accept (the bit is 1) or reject (the bit is 0). See Figure 4.1b for a diagram.

We say that $\text{IARG} = (\mathcal{P}, \mathcal{V})$ is an interactive argument in the ROM for a relation \mathcal{R} if it satisfies two main properties: *completeness* and *soundness*.

- **Completeness.** Informally, for every instance-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, the (honest) argument prover \mathcal{P} on input (\mathbf{x}, \mathbf{w}) makes the argument verifier \mathcal{V} on input \mathbf{x} accept, with probability 1. The probability here is taken over the choice of random oracle f , as well as any randomness of \mathcal{P} and of \mathcal{V} .

See Definition 4.2.1 below for a formal definition.

- **Soundness.** Informally, every malicious argument prover $\tilde{\mathcal{P}}$ convinces the (honest) argument verifier \mathcal{V} to accept an instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ with at most a small error probability ϵ_{ARG} (over the choice of random oracle f). The malicious argument prover $\tilde{\mathcal{P}}$ may query the random oracle t for some given query bound t , and is otherwise computationally unbounded.

Similarly to the case of a NARG, an important technical detail is how the instance \mathbf{x} is chosen. In Definition 4.2.2 below we provide a formal statement for non-adaptive soundness, and in Definition 4.2.3 below we provide a formal statement for adaptive soundness.

Below we provide the definitions of completeness, non-adaptive soundness, and adaptive soundness. Given two interactive oracle algorithms A and B , we use the notation $\langle A, B \rangle_{\text{ARG}}$ to indicate the random variable that equals the output of B after interacting with A , and where the probability is taken over any randomness used by A or B (as well as the random oracle given to A and B).

Definition 4.2.1: completeness

An interactive argument $\text{IARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} is **(perfectly) complete** if for every output size $\sigma \in \mathbb{N}$ of the random oracle and instance-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$,

$$\Pr \left[\langle \mathcal{P}^f(\mathbf{x}, \mathbf{w}), \mathcal{V}^f(\mathbf{x}) \rangle_{\text{ARG}} = 1 \mid f \leftarrow \mathcal{U}(\sigma) \right] = 1.$$

The probability is also over f and any randomness of the argument prover \mathcal{P} and verifier \mathcal{V} .

Definition 4.2.2: non-adaptive soundness

An interactive argument $\text{IARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has **non-adaptive soundness error** ϵ_{ARG} if for every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, t -query malicious argument prover $\tilde{\mathcal{P}}$, and instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$,

$$\Pr \left[\langle \tilde{\mathcal{P}}^f, \mathcal{V}^f(\mathbf{x}) \rangle_{\text{ARG}} = 1 \mid f \leftarrow \mathcal{U}(\sigma) \right] \leq \epsilon_{\text{ARG}}(\sigma, t, \mathbf{x}).$$

The probability is taken over f and any randomness of the argument verifier \mathcal{V} .

Definition 4.2.3: adaptive soundness

An interactive argument $\text{IARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has **adaptive soundness error** ϵ_{ARG} if for every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, t -query malicious argument prover $\tilde{\mathcal{P}}$, and instance size bound $n \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \langle \tilde{\mathcal{P}}^f(\mathbf{aux}), \mathcal{V}^f(\mathbf{x}) \rangle_{\text{ARG}} = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \mathbf{aux}) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \leq \epsilon_{\text{ARG}}(\sigma, t, n).$$

The probability is taken over f and any randomness of the argument verifier \mathcal{V} .

Public coins We study interactive arguments that are *public-coin*, which means that each message of the argument verifier \mathcal{V} is a random message. A non-interactive argument is trivially public coin because, in this case, the argument verifier does not send any messages.

Definition 4.2.4. $\text{IARG} = (\mathcal{P}, \mathcal{V})$ is **public-coin** if each message ρ_i by the argument verifier \mathcal{V} is a random binary string of some prescribed size r_i . In this case, the decision bit of the argument verifier \mathcal{V} is a function only of the random oracle f , the instance \mathbf{x} , and the transcript of interaction $(\alpha_1, \rho_1, \dots, \alpha_k, \rho_k)$; we denote this bit by $\mathcal{V}^f(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$.

4.3 Efficiency measures

We study several efficiency measures for interactive and non-interactive arguments. All measures can be functions of the output size σ of the random oracle and of the instance \mathbf{x} (or the instance size bound n).

Prover cost We denote by $\mathbf{t}_{\mathcal{P}}$ the running time of the prover algorithm \mathcal{P} , and by $\mathbf{q}_{\mathcal{P}}$ the number of queries that it makes to the random oracle (regardless of their size). If \mathcal{P} is part of an interactive argument then \mathcal{P} is an interactive algorithm, and these costs refer to the total running time and number of queries across all rounds of interaction.

Verifier cost We denote by $\mathbf{t}_{\mathcal{V}}$ the running time of the verifier algorithm \mathcal{V} , and by $\mathbf{q}_{\mathcal{V}}$ the number of queries that it makes to the random oracle (regardless of their size). If \mathcal{V} is part of an interactive argument then \mathcal{V} is an interactive algorithm, and these costs refer to the total running time and number of queries across all rounds of interaction.

Communication In the case of an *interactive* argument, we denote by $\mathbf{c}_{\mathcal{P}}$ the total number of bits sent by the argument prover \mathcal{P} (across all rounds of interaction) and we denote by $\mathbf{c}_{\mathcal{V}}$ the total number of bits sent by the argument verifier \mathcal{V} (across all rounds of interaction). In particular, the transcript of interaction between \mathcal{P} and \mathcal{V} consists of $\mathbf{c} = \mathbf{c}_{\mathcal{P}} + \mathbf{c}_{\mathcal{V}}$ bits.

In the case of a *non-interactive* argument, there is a single message π from the argument prover \mathcal{P} and no messages from the argument verifier \mathcal{V} , which means that $\mathbf{c}_{\mathcal{V}} = 0$. In this case we denote by $\mathbf{s} = \mathbf{c}_{\mathcal{P}}$ the size of the argument string π .

4.4 Succinct arguments

A central goal of the non-interactive arguments described in this book is **succinctness**, which is a spectrum of efficiency goals with differing strengths. Briefly, succinctness can refer to *small communication* or, additionally, *fast verification*. We elaborate on these notions below.

Baseline: the trivial argument system Every relation \mathcal{R} has a trivial proof system: the prover sends the witness, and the verifier checks the validity of the witness. Expressing this as a trivial non-interactive argument yields two main baselines that enable defining succinctness.

The (honest) argument prover, given as input an instance \mathbf{x} and witness w , outputs the argument string $\pi := w$; and the argument verifier, given as input the instance \mathbf{x} and argument string π , checks that $(\mathbf{x}, \pi) \in \mathcal{R}$. This non-interactive argument does not use the random oracle, and has perfect completeness and zero soundness error. (As discussed in Remark 5.2.2, the trivial non-interactive argument is not zero knowledge; but this is irrelevant to the present discussion on efficiency.)

The trivial non-interactive argument for \mathcal{R} sets two important baselines.

- The argument size is the witness size: $s := \text{len}(w)$.
- The verification time is the time to decide the relation: $t_v := \text{time}_{\mathcal{R}}(\mathbf{x}, w)$.

Succinctness covers several goals that involve doing better in argument size or verification time.

Succinctness in communication Argument systems that achieve communication that is better than the witness size are considered *succinct in communication*. The relevant metric for a non-interactive argument is the argument size s , while the relevant metric for an interactive argument is the prover-to-verifier communication \mathbf{c}_p (or perhaps even the total communication $\mathbf{c} = \mathbf{c}_p + \mathbf{c}_v$).

But what does “better than the witness size” mean?

A modest goal is communication that is sublinear in witness size while allowing for some dependence on the output size of the random oracle: $\text{poly}(\sigma) \cdot o(\text{len}(w))$.¹ Even better is a (poly)logarithmic dependence on witness size while again allowing for some dependence on the output size of the random oracle: $\text{poly}(\sigma, \log \text{len}(w))$. These are not the only regimes of interest, and in fact usually it is helpful to express the communication succinctness of a succinct argument as an explicit expression of the relevant parameters. Overall, the goal is to achieve $s \ll \text{len}(w)$.

Succinctness in verification Argument systems that achieve verifier time that is better than the time to decide the relation are considered *succinct in verification*. The relevant metric for both interactive arguments and non-interactive arguments is the time t_v of the argument verifier.

Again, we can ask “what does better than the time to decide the relation” mean?

Similarly to above, a modest goal is a verification time that is sublinear in the relation decision time while allowing for some dependence on the output size of the random oracle: $\text{poly}(\sigma) \cdot o(\text{time}_{\mathcal{R}}(\mathbf{x}, w))$. Even better is a (poly)logarithmic dependence on the relation decision time while again allowing for some dependence on the output size of the random oracle: $\text{poly}(\sigma, \log \text{time}_{\mathcal{R}}(\mathbf{x}, w))$. It is helpful to express the verification succinctness of a succinct

¹The “ $o(n)$ ” notation includes all functions $g(n)$ such that, for every $c > 0$, $g(n) \leq c \cdot n$ for large enough n .

argument as an explicit expression of the relevant parameters, and overall the goal is to achieve $t_v \ll \text{time}_{\mathcal{R}}(\mathbf{x}, \mathbf{w})$.

Observe that the verifier time of an argument system is an upper bound on communication ($t_v \geq s$), which means that succinctness in verification usually implies (some form of) succinctness in communication. In constructions of succinct arguments, the achieved communication complexity is often better than the upper bound implied by the verifier time.

Special case: no witnesses Relations \mathcal{R} where there are no witnesses are a notable special case. In this case \mathcal{R} can be viewed as a language \mathcal{L} , and the trivial non-interactive argument above collapses to a solitary verifier: the argument verifier decides on its own whether the input instance \mathbf{x} is \mathcal{L} , without receiving anything from the argument prover (as there are no witnesses).

The baseline of witness size for \mathcal{L} is zero, so it is *not meaningful to consider succinctness in communication*. Indeed, sending any argument string is more expensive in communication than sending nothing. So, there is no way to improve in communication over the baseline.

On the other hand, succinctness in verification is meaningful, namely, the goal is to achieve an argument verifier that runs in time better than the time $\text{time}_{\mathcal{L}}(\mathbf{x})$ to decide membership of \mathbf{x} in the language \mathcal{L} . As before, one can consider a modest goal of time $\text{poly}(\sigma) \cdot o(\text{time}_{\mathcal{L}}(\mathbf{x}))$, aim for a stronger goal such as $\text{poly}(\sigma, \log \text{time}_{\mathcal{L}}(\mathbf{x}))$, or, more generally, achieve $t_v \ll \text{time}_{\mathcal{L}}(\mathbf{x})$.

A common example of a relation with no witnesses supported by succinct arguments is $\text{DTIME}(T)$, which consists of all (deterministic) machines that accept in time T . It is the analogue of the relation $\text{NTIME}(T)$ but without witnesses.

Terminology for succinct arguments An interactive argument or non-interactive argument is generally called **succinct** if any of the above improvements over baseline are met. Moreover, the acronym **SNARG** stands for a succinct *non-interactive* argument. In particular, in this book we study constructions of SNARGs in the random oracle model (ROM). If a SNARG satisfies knowledge soundness then it may be called a **SNARK**, and if it additionally satisfies zero knowledge it may be called a **zkSNARK**.

In practice, the random oracle is heuristically instantiated via a cryptographic hash function (see Section 2.4.1), which yields an argument system in the *standard model* (what cryptographers call a setting without oracles). The specification of the hash function can then be viewed as the public parameters, or *setup*, of the argument system. Making the choice of cryptographic hash function typically does not involve any secret randomness, in which case the SNARG is said to have a **transparent setup** (or **public-coin setup**). Thus, for example, a zkSNARK in the ROM becomes a **zkSNARK with a transparent setup**, when the random oracle is heuristically instantiated. A notable example of SNARGs in the ROM are **STARKs** (scalable transparent arguments of knowledge), whose security in the ROM is covered by the material discussed in this book.

5 Additional security definitions

In applications, arguments are often required to satisfy additional security definitions beyond completeness and soundness. Since non-interactive arguments (NARGs) are the focus of this book, below we provide these additional security definitions only for non-interactive arguments.

- In Section 5.1 we discuss *knowledge soundness*.
- In Section 5.2 we discuss *zero knowledge*.

In this book we describe how to achieve these additional security definitions for each NARG construction that we study.

5.1 Knowledge soundness

The soundness property (see Definitions 4.1.2 and 4.1.3) states that every malicious argument prover cannot convince the argument verifier to accept instances not in the language (associated to the relation), except for a small error probability.

Knowledge soundness strengthens this by requiring that whenever an argument prover convinces the argument verifier to accept then, except for a small error probability, the argument prover “knows” a valid witness that shows that the instance is in the language.¹

If the instance is in the language, then a witness exists and so can be found, for example, via brute-force search (an inefficient procedure). Knowledge soundness requires that there exists an efficient procedure, called the *extractor*, that finds a witness from the argument prover (thereby demonstrating the malicious prover’s knowledge).

There are different flavors of knowledge soundness, depending on how the extractor accesses information about the malicious prover. Below we consider two flavors, a stronger one and a weaker one. Both are *adaptive*, in that the argument prover can use the random oracle to choose the instance for which to provide the argument string.²

Straightline knowledge soundness The definition below captures a flavor of knowledge soundness that is known as *straightline*, because the extractor only needs information associated to a single execution of the argument prover. In more detail, the extractor finds the witness given only the argument prover’s output and query-answer trace with the random oracle.³ This strong notion is particularly useful in a multitude of settings.

¹Arguments that satisfy this property are sometimes called *proofs of knowledge* or, if emphasizing the fact that security holds only against suitably bounded malicious provers, *arguments of knowledge*.

²Corresponding non-adaptive relaxations can be straightforwardly derived from these definitions.

³In particular, since the extraction procedure does not “change” the random oracle, this property is said to hold in the *non-programmable* ROM.

Definition 5.1.1: straightline knowledge soundness

A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has **straightline knowledge soundness error** κ_{ARG} if there exists a polynomial-time deterministic algorithm \mathcal{E} (the extractor) such that for every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, t -query deterministic argument prover $\tilde{\mathcal{P}}$, and instance size bound $n \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ \mathbf{w} \leftarrow \mathcal{E}(\mathbf{x}, \pi, \text{tr}) \end{array} \right] \leq \kappa_{\text{ARG}}(\sigma, t, n).$$

Rewinding knowledge soundness The definition below captures a relaxation known as *rewinding*, where the extraction additionally receives the argument prover (as a black-box). This enables the extractor to re-run the argument prover multiple times on inputs of its choice. The extractor is also given access to the random oracle, for example to facilitate these re-runs. The extractor error is additionally allowed to depend on failure probability of the argument prover (in convincing the verifier), which is also defined below.

Definition 5.1.2. Let $\text{NARG} = (\mathcal{P}, \mathcal{V})$ be a non-interactive argument. A deterministic argument prover $\tilde{\mathcal{P}}$ has **failure probability** $\delta_{\tilde{\mathcal{P}}}$ if for every output size $\sigma \in \mathbb{N}$ of the random oracle and instance size bound $n \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} |\mathbf{x}| > n \\ \vee \mathcal{V}^f(\mathbf{x}, \pi) = 0 \end{array} \middle| (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \right] \leq \delta_{\tilde{\mathcal{P}}}(\sigma, n).$$

Definition 5.1.3: rewinding knowledge soundness

A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has **rewinding knowledge soundness error** κ_{ARG} with **extraction time** et_{ARG} if there exists a probabilistic algorithm \mathcal{E} (the extractor) such that for every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, t -query deterministic argument prover $\tilde{\mathcal{P}}$ with failure probability $\delta_{\tilde{\mathcal{P}}}$ and running time $\tau_{\tilde{\mathcal{P}}}$, and instance size bound $n \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ \mathbf{w} \leftarrow \mathcal{E}^f(\mathbf{x}, \pi, \text{tr}, \tilde{\mathcal{P}}) \end{array} \right] \leq \kappa_{\text{ARG}}(\sigma, t, n, \delta_{\tilde{\mathcal{P}}}(\sigma, n)).$$

Moreover, \mathcal{E} runs in expected time $\text{et}_{\text{ARG}}(\sigma, t, n, \delta_{\tilde{\mathcal{P}}}(\sigma, n), \tau_{\tilde{\mathcal{P}}}(\sigma, n))$ (over the given inputs and internal randomness).

Definition 5.1.1 implies Definition 5.1.3 because the two properties are the same except that in the latter the extractor is randomized and also has access to the malicious argument prover.

Remark 5.1.4 (probabilistic provers). The knowledge soundness definitions are stated for *deterministic* provers for simplicity. These imply definitions for probabilistic provers, as we now explain.

- The definition of *straightline* knowledge soundness for probabilistic argument provers $\tilde{\mathcal{P}}$ is the same as in Definition 5.1.1 except that the execution $(\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f$ in the experiment also takes into account the randomness of $\tilde{\mathcal{P}}$.
- The definition of *rewinding* knowledge soundness for probabilistic argument provers $\tilde{\mathcal{P}}$ is a slight variant of Definition 5.1.3. Namely, letting \mathcal{R} be the set of possible random strings for $\tilde{\mathcal{P}}$, the experiment is modified to explicitly account for the randomness of $\tilde{\mathcal{P}}$ as follows:

$$\left[\begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \xi \leftarrow \mathcal{R} \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f(\xi) \\ \mathbf{w} \leftarrow \mathcal{E}^f(\mathbf{x}, \pi, \text{tr}, \tilde{\mathcal{P}}(\xi)) \end{array} \right]$$

The extractor \mathcal{E} has black-box access to the argument prover $\tilde{\mathcal{P}}$ with its randomness ξ hardcoded. The rest of the definition is the same.

In both of the above cases, the knowledge error and extractor time bounds for probabilistic provers are directly implied by the corresponding bounds for deterministic provers. Indeed, the probabilistic argument prover $\tilde{\mathcal{P}}$ can be expressed as $|\mathcal{R}|$ deterministic argument provers, each corresponding to a different choice of randomness $\xi \in \mathcal{R}$ (and all with the same query bound). Since the knowledge soundness definition applies to all deterministic provers, the error bound κ_{ARG} holds for each of these $|\mathcal{R}|$ provers. The error of $\tilde{\mathcal{P}}$ is the average of these errors, and hence is upper bounded by κ_{ARG} . Similarly for the expected running time of the extractor: the upper bound et_{ARG} holds for each of the $|\mathcal{R}|$ deterministic provers; the expected running time of \mathcal{E} for $\tilde{\mathcal{P}}$ is the average of the expected running times for the deterministic provers, and this average is upper bounded by et_{ARG} .

Remark 5.1.5 (monotonicity of κ_{ARG}). The rewinding knowledge soundness error κ_{ARG} depends on the parameters $\sigma, t, n, \delta_{\tilde{\mathcal{P}}}$. Throughout this book, we assume that κ_{ARG} is (weakly) monotone increasing with respect to these parameters. Except for contrived examples, known error functions κ_{ARG} behave this way. For example, if t increases then we expect κ_{ARG} to not decrease because the argument prover has more power (it can ask more queries to the random oracle). Moreover, if $\delta_{\tilde{\mathcal{P}}}$ increases then we expect κ_{ARG} to not decrease because knowledge extraction becomes harder if the probability that the argument prover convinces the argument verifier decreases.

Remark 5.1.6. It is straightforward to construct a non-interactive argument that is *not* knowledge sound (for any reasonable definition of knowledge soundness including the ones we give). Let $\mathcal{G} = \{g_\ell : \{0, 1\}^\ell \rightarrow \{0, 1\}^\ell\}_{\ell \in \mathbb{N}}$ be a collection of permutations and consider the relation $\mathcal{R} := \{(\mathbf{x}, \mathbf{w}) : g_{\text{len}(\mathbf{x})}(\mathbf{w}) = \mathbf{x}\}$. Note that $\mathcal{L}(\mathcal{R}) = \{0, 1\}^*$, that is, the language associated to \mathcal{R} is trivial (it contains all binary strings). Consider the following trivial non-interactive argument for \mathcal{R} :

- the argument prover $\mathcal{P}^f(\mathbf{x}, \mathbf{w})$ outputs $\pi := \perp$ (an empty string); and
- the argument verifier $\mathcal{V}^f(\mathbf{x}, \pi)$ always accepts.

This non-interactive argument is perfectly complete and perfectly sound (since there are no instances not in the language).

On the other hand, any algorithm that receives as input an instance \mathbf{x} and outputs a witness \mathbf{w} such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ has found a preimage of \mathbf{x} under $g_{\text{len}(\mathbf{x})}$. Additionally, receiving as input information about a malicious prover (e.g., its query-answer trace or even the code of the malicious prover itself) does not provide any help towards inverting $g_{\text{len}(\mathbf{x})}$ because a malicious argument prover can convince the argument verifier without making any queries or producing any argument string.

Hence, if inverting functions from the collection of permutations \mathcal{G} is hard (e.g., \mathcal{G} is a collection of one-way permutations), then no efficient algorithm can find a witness for the given instance with non-negligible probability (in $\text{len}(\mathbf{x})$). We conclude that the above non-interactive argument does not satisfy knowledge soundness.

Equivalent definition of rewinding knowledge soundness We use a definition equivalent to Definition 5.1.3 when establishing rewinding knowledge soundness for non-interactive arguments. Below we discuss this alternative definition, which in this book is more convenient to work with. Briefly, the extractor \mathcal{E} receives as input the query-answer trace of the argument verifier, instead of receiving query access to the random oracle.

Definition 5.1.7: rewinding knowledge soundness

A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has **rewinding knowledge soundness error** κ_{ARG} **with extraction time** et_{ARG} if there exists a probabilistic algorithm \mathcal{E} (the extractor) such that for every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, t -query deterministic argument prover $\tilde{\mathcal{P}}$ with failure probability $\delta_{\tilde{\mathcal{P}}}$ and running time $\tau_{\tilde{\mathcal{P}}}$, and instance size bound $n \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ b \xleftarrow{\text{try}} \mathcal{V}^f(\mathbf{x}, \pi) \\ \mathbf{w} \leftarrow \mathcal{E}(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}}) \end{array} \right] \leq \kappa_{\text{ARG}}(\sigma, t, n, \delta_{\tilde{\mathcal{P}}}(\sigma, n)).$$

Moreover, \mathcal{E} runs in expected time $\text{et}_{\text{ARG}}(\sigma, t, n, \delta_{\tilde{\mathcal{P}}}(\sigma, n), \tau_{\tilde{\mathcal{P}}}(\sigma, n))$ (over the given inputs and internal randomness).

Below we prove that Definition 5.1.3 and Definition 5.1.7 are equivalent. Here we provide some intuition for this equivalence. What use is the random oracle f to the extractor \mathcal{E} ? Information about queries performed by the malicious argument prover $\tilde{\mathcal{P}}$ are included in the trace (an input to \mathcal{E}), so \mathcal{E} does not need to access f for such queries. Queries not performed by $\tilde{\mathcal{P}}$ or (honest) argument verifier \mathcal{V} are random and independent of all inputs given to \mathcal{E} , which means that \mathcal{E} can sample the answer to these queries without access to f . Hence, the only queries that are potentially “valuable” to \mathcal{E} are those performed by \mathcal{V} (and not by $\tilde{\mathcal{P}}$). The alternative definition provides the query-answer trace of \mathcal{V} as an input to \mathcal{E} to cover this case.

Claim 5.1.8. A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has knowledge soundness error κ_{ARG} with respect to definition Definition 5.1.3 if and only if it has knowledge soundness error κ_{ARG} with respect to definition Definition 5.1.7.

Proof. We prove each direction of the claim separately.

Definition 5.1.7 \implies Definition 5.1.3 Suppose that NARG has knowledge soundness error κ_{ARG} with respect to Definition 5.1.7, and let \mathcal{E}' be its extractor. We show that NARG has knowledge soundness error κ_{ARG} with respect to Definition 5.1.3, using the below extractor \mathcal{E} .

$\mathcal{E}^f(\mathbf{x}, \pi, \text{tr}, \tilde{\mathcal{P}})$:

1. Compute $b \xleftarrow{\text{try}} \mathcal{V}^f(\mathbf{x}, \pi)$.
2. Compute $\mathbf{w} \leftarrow \mathcal{E}'(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}})$.
3. Output \mathbf{w} .

If $\mathbf{et}'_{\text{ARG}}$ is the running time of \mathcal{E}' and \mathbf{t}_v is the running time of \mathcal{V} , then the running time of \mathcal{E} is at most $\mathbf{et}'_{\text{ARG}}(\sigma, t, n, \delta_{\tilde{\mathcal{P}}}(\sigma, n), \tau_{\tilde{\mathcal{P}}}(\sigma, n)) + \mathbf{t}_v$.

Next we upper bound the knowledge soundness error. By definition of \mathcal{E} ,

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ \mathbf{w} \leftarrow \mathcal{E}^f(\mathbf{x}, \pi, \text{tr}, \tilde{\mathcal{P}}) \end{array} \right] \\ & \leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ b \xleftarrow{\text{try}} \mathcal{V}^f(\mathbf{x}, \pi) \\ \mathbf{w} \leftarrow \mathcal{E}'(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}}) \end{array} \right] \\ & = \kappa_{\text{ARG}}(\sigma, t, n, \delta_{\tilde{\mathcal{P}}}(\sigma, n)). \end{aligned}$$

Definition 5.1.3 \implies Definition 5.1.7 Suppose that NARG has knowledge soundness error κ_{ARG} with respect to Definition 5.1.3, and let \mathcal{E} be its extractor. We show that NARG has knowledge soundness error κ_{ARG} with respect to Definition 5.1.7, using the below extractor \mathcal{E}' .

$\mathcal{E}'(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}})$:

1. Lazily sample an oracle $\dot{f}' \leftarrow \mathcal{U}(\sigma)$.
2. Define $\dot{f}'_{\text{tr}, \text{tr}_v}(x) := \begin{cases} y & \text{if } (x, y) \in \text{tr} \cup \text{tr}_v \\ \dot{f}'(x) & \text{otherwise} \end{cases}$.
3. Output $\mathbf{w} \leftarrow \mathcal{E}^{\dot{f}'_{\text{tr}, \text{tr}_v}}(\mathbf{x}, \pi, \text{tr}, \tilde{\mathcal{P}})$.

We bound the running time of \mathcal{E}' . Let \mathbf{et}_{ARG} is the running time of \mathcal{E} . Each query to $\dot{f}'_{\text{tr}, \text{tr}_v}(x)$ can be answered in time $\log(t + \mathbf{q}_v)$ where $|\text{tr}| = t$ and $|\text{tr}_v| = \mathbf{q}_v$ assuming the traces are sorted first. Hence the running time of \mathcal{E}' is at most $\log(t + \mathbf{q}_v) \cdot \mathbf{et}_{\text{ARG}}(\sigma, t, n, \delta_{\tilde{\mathcal{P}}}(\sigma, n), \tau_{\tilde{\mathcal{P}}}(\sigma, n)) + O((t + \mathbf{q}_v) \cdot \log(t + \mathbf{q}_v))$.

Next we upper bound the knowledge soundness error. If tr and tr_v are both traces with respect to the oracle f (and thus are consistent with f), then f is equivalent to $\dot{f}'_{\text{tr}, \text{tr}_v}$. Hence

we can write

$$\begin{aligned}
& \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}^f(\mathbf{x}, \pi) \\ \mathbf{w} \leftarrow \mathcal{E}'(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}}) \end{array} \right] \\
&= \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ f' \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}^f(\mathbf{x}, \pi) \\ \mathbf{w} \leftarrow \mathcal{E}'^{f_{\text{tr}, \text{tr}_v}}(\mathbf{x}, \pi, \text{tr}, \tilde{\mathcal{P}}) \end{array} \right] \quad (\text{by definition of } \mathcal{E}') \\
&= \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ f' \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}^f(\mathbf{x}, \pi) \\ \mathbf{w} \leftarrow \mathcal{E}^f(\mathbf{x}, \pi, \text{tr}, \tilde{\mathcal{P}}) \end{array} \right] \quad (\text{since } f \equiv f'_{\text{tr}, \text{tr}_v}) \\
&= \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ \mathbf{w} \leftarrow \mathcal{E}^f(\mathbf{x}, \pi, \text{tr}, \tilde{\mathcal{P}}) \end{array} \right] \quad (\text{since } f' \text{ is not used}) \\
&\leq \kappa_{\text{ARG}}(\sigma, t, n, \delta_{\tilde{\mathcal{P}}}(\sigma, n)) .
\end{aligned}$$

□

5.2 Zero knowledge

Soundness is a security definition that protects the argument verifier against malicious argument provers; similarly for knowledge soundness. Here we discuss a security definition that, in contrast, protects the *privacy* of the (honest) argument prover.

While the instance \mathbf{x} is an input to the argument prover and argument verifier, the witness \mathbf{w} is an input to the argument prover only. Indeed, the argument verifier receives the argument string π produced by the argument prover, instead of the witness \mathbf{w} . This typically enables improved efficiency: π is much smaller than \mathbf{w} or, additionally, π is much faster to validate than directly checking that (\mathbf{x}, \mathbf{w}) is in the given relation \mathcal{R} .

A different goal is *privacy*: π does not reveal any information about \mathbf{w} , beyond what is implied by the fact that \mathbf{x} is in the language $\mathcal{L}(\mathcal{R})$ (corresponding to the relation \mathcal{R}). This goal is called *zero knowledge*, and can be useful on its own (even without other efficiency benefits).

Zero knowledge is captured via an efficient probabilistic procedure \mathcal{S} called the *simulator*. Informally, we consider the statistical distance between the output of an adversary \mathcal{A} in two experiments.

- In a “real-world” experiment, the adversary \mathcal{A} receives an argument string π produced by the (honest) argument prover \mathcal{P} given as input the instance \mathbf{x} and (valid) witness \mathbf{w} .
- In a “simulated-world” experiment, the adversary \mathcal{A} receives an argument string π produced by the simulator \mathcal{S} given as input the instance \mathbf{x} only.

Zero knowledge requires that the outputs of \mathcal{A} in these experiments are statistically close.

In other words, the simulator \mathcal{S} efficiently samples, up to a small error, from the distribution of a “real” argument string π while only receiving the instance \mathbf{x} as input. In particular, the argument string π does not “leak” hard-to-compute information about the witness that produced it. Note that it is crucial to consider only instance-witness pairs (\mathbf{x}, \mathbf{w}) in the relation \mathcal{R} , because these pairs are the ones for which the honest argument prover \mathcal{P} is intended to work.

Formalizing the above setting requires care, especially so in the ROM. For example, as justified in Section 6.6, it is essential that the simulator \mathcal{S} can change answers of the random oracle. The simulator \mathcal{S} outputs, in addition to an argument string π , a list of query-answer pairs μ that are used to modify the random oracle. In the cryptography literature, this is known as the *explicitly-programmable random oracle model* (EPROM).⁴

Below we describe a non-adaptive definition of zero knowledge, and then describe an adaptive definition of zero knowledge.

(1) Non-adaptive zero knowledge We begin by considering the setting of zero knowledge for a single instance chosen before the random oracle is sampled. We require that, for every instance-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, the adversary’s outputs in the real-world and simulated-world experiments are statistically close. In general, the statistical distance depends on the output size $\sigma \in \mathbb{N}$ of the random oracle, the query bound $t \in \mathbb{N}$ for the adversary, and the instance \mathbf{x} .

Definition 5.2.1. A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has **non-adaptive zero-knowledge error** z_{ARG} (in the EPROM) if there exists a probabilistic polynomial-time simulator \mathcal{S} such that, for every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, t -query algorithm \mathcal{A} , and instance-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, the following two distributions are $z_{\text{ARG}}(\sigma, t, \mathbf{x})$ -close in statistical distance:

$$\mathcal{D}_{\text{real}} := \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \mathcal{P}^f(\mathbf{x}, \mathbf{w}) \\ \text{out} \leftarrow \mathcal{A}^f(\pi) \end{array} \right. \right\} \quad \text{and} \quad \mathcal{D}_{\text{sim}} := \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\pi, \mu) \leftarrow \mathcal{S}^f(\mathbf{x}) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\pi) \end{array} \right. \right\}.$$

Above, $f[\mu]$ denotes the function f modified to be consistent with the query-answer list μ .

In the simulated-world experiment (the one on the right), the simulator has access to the random oracle, and “programs” the random oracle via a list μ of query-answer pairs. (This relaxes the non-programmable notion of zero knowledge discussed in Section 6.6, which is too strong to achieve.)

We remark that, since we quantify over all algorithms \mathcal{A} and instance-witness pairs $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, the adversary \mathcal{A} may have (\mathbf{x}, \mathbf{w}) hardcoded in its description. Hence, Definition 5.2.1 implies that \mathcal{A} cannot distinguish the two experiments even when it knows the argument prover’s inputs.

Remark 5.2.2. It is straightforward to construct a non-interactive argument that is *not* zero knowledge according to Definition 5.2.1. Consider the “trivial” (non-succinct) non-interactive argument:

- the argument prover $\mathcal{P}^f(\mathbf{x}, \mathbf{w})$ outputs $\pi := \mathbf{w}$; and

⁴This is in contrast to the *fully-programmable random oracle model* (FPROM), where the simulator directly impersonates the random oracle given to the adversary and hence can adaptively choose the answer of queries by the adversary (rather outputting in advance a list of query-answer pairs that modify the random oracle).

- the argument verifier $\mathcal{V}^f(\mathbf{x}, \pi)$ checks that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ (by running \mathcal{R} 's decider algorithm).

This argument is perfectly complete and perfectly sound (because it has no soundness error even against unbounded malicious provers).

On the other hand, for a given relation \mathcal{R} , if the trivial argument satisfies Definition 5.2.1 then the (efficient) simulator finds a valid witness \mathbf{w} given the input instance \mathbf{x} (up to a small error). This implies that the relation \mathcal{R} is easy to solve (and thus unlikely to be NP-complete).

We learn that the trivial argument is *not* zero knowledge for hard languages. In particular Definition 5.2.1 rules out trivial constructions for languages of interest.

(2) Adaptive zero knowledge An adversary may, however, choose the instance based on the random oracle, which leads to an *adaptive* notion of zero knowledge. We consider a strengthening of Definition 5.2.1 where, in an initial phase, the adversary queries the random oracle and outputs its choice of instance \mathbf{x} and witness \mathbf{w} , as well as a private state for continuing its computation after receiving the (real or simulated) argument string π . Note that we only need to consider adversaries that are *admissible* in the sense that the chosen \mathbf{w} is a valid witness for \mathbf{x} (and, in particular, the chosen \mathbf{x} is in the language). This is because zero knowledge protects the honest argument prover.

Definition 5.2.3: adaptive zero knowledge

A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has **adaptive zero-knowledge error** z_{ARG} (in the EPROM) if there exists a probabilistic polynomial-time simulator \mathcal{S} such that, for every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, t -query admissible adversary \mathcal{A} , and instance bound $n \in \mathbb{N}$, the following two distributions are $z_{\text{ARG}}(\sigma, t, n)$ -close in statistical distance:

$$\mathcal{D}_{\text{real}} := \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}^f \\ \pi \leftarrow \mathcal{P}^f(\mathbf{x}, \mathbf{w}) \\ \text{out} \leftarrow \mathcal{A}^f(\text{aux}, \pi) \end{array} \right. \right\} \quad \text{and} \quad \mathcal{D}_{\text{sim}} := \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}^f \\ (\pi, \mu) \leftarrow \mathcal{S}^f(\mathbf{x}) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\text{aux}, \pi) \end{array} \right. \right\}.$$

Above, \mathcal{A} is admissible if it always outputs \mathbf{x}, \mathbf{w} such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ and $|\mathbf{x}| \leq n$.

Witness indistinguishability Zero knowledge implies a weaker privacy property called *witness indistinguishability*. Informally, this property requires that argument strings produced for the same instance (in the language) but using different valid witnesses are distributed almost identically. Below we discuss witness indistinguishability because the definition is straightforward, and helps to understand zero knowledge better.

Definition 5.2.4. A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has **adaptive witness indistinguishability error** z_{ARG} if for every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, t -query admissible adversary \mathcal{A} , and instance size bound $n \in \mathbb{N}$, the random

variables X_0 and X_1 are $z_{\text{ARG}}(\sigma, t, n)$ -close in statistical distance where

$$X_b := \left\{ \text{out} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \mathbf{w}_0, \mathbf{w}_1, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ \pi \leftarrow \mathcal{P}^f(\mathbf{x}, \mathbf{w}_b) \\ \text{out} \leftarrow \mathcal{A}(\mathbf{aux}, \pi) \end{array} \right\}.$$

Above, \mathcal{A} is admissible if it always outputs $\mathbf{x}, \mathbf{w}_0, \mathbf{w}_1$ such that $(\mathbf{x}, \mathbf{w}_0), (\mathbf{x}, \mathbf{w}_1) \in \mathcal{R}$ and $|\mathbf{x}| \leq n$.

Lemma 5.2.5. If $\text{NARG} = (\mathcal{P}, \mathcal{V})$ has (adaptive) zero-knowledge error z_{ARG} (see Definition 5.2.3) then it has (adaptive) witness indistinguishability error $2 \cdot z_{\text{ARG}}$.

Proof. Fix \mathcal{A} that is admissible according to Definition 5.2.4. Define $X_{\mathcal{S}}$ to be the distribution of the adversary's output when the argument string is produced by the simulator \mathcal{S} provided by the zero-knowledge property:

$$X_{\mathcal{S}} := \left\{ \text{out} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \mathbf{w}_0, \mathbf{w}_1, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ \pi \leftarrow \mathcal{S}^f(\mathbf{x}) \\ \text{out} \leftarrow \mathcal{A}(\mathbf{aux}, \pi) \end{array} \right\}.$$

The zero-knowledge property tells us that X_0 and $X_{\mathcal{S}}$ are $z_{\text{ARG}}(\sigma, t, n)$ -close in statistical distance (if we consider $(\mathbf{x}, \mathbf{w}_0)$ as the instance-witness pair chosen by the adversary), and also that X_1 and $X_{\mathcal{S}}$ are $z_{\text{ARG}}(\sigma, t, n)$ -close in statistical distance (if we consider $(\mathbf{x}, \mathbf{w}_1)$ as the instance-witness pair chosen by the adversary). Indeed, since \mathcal{A} is admissible according to Definition 5.2.4, we know that $(\mathbf{x}, \mathbf{w}_0), (\mathbf{x}, \mathbf{w}_1) \in \mathcal{R}$ and $|\mathbf{x}| \leq n$, and so we can indeed invoke the zero-knowledge property on each of the instance-witness pairs. We conclude via Claim 1.2.7 that X_0 and X_1 are $2 \cdot z_{\text{ARG}}(\sigma, t, n)$ -close in statistical distance. \square

Note that witness indistinguishability provides no guarantees if witnesses are unique. For example, the trivial argument described in Remark 5.2.2, where the argument string is simply a witness, is trivially witness indistinguishable for any (even hard) relation where every instance in the language has a unique witness. This, in particular, provides a separation between the notions of zero knowledge and witness indistinguishability. More generally, witness indistinguishability may “leak” information about the set of all valid witnesses for an instance, whereas zero knowledge requires that no information is leaked besides the fact the instance is in the language.

Remark 5.2.6. Definition 5.2.4 can be relaxed to a corresponding non-adaptive variant, where the instance \mathbf{x} and witnesses \mathbf{w}_0 and \mathbf{w}_1 are chosen before the random oracle is sampled. In such a case, an analogous statement to Lemma 5.2.5 holds: non-adaptive zero knowledge implies non-adaptive witness indistinguishability, again with a multiplicative factor of 2 in the error bound.

6 Basic observations about arguments

We discuss basic observations about arguments in the random oracle model (ROM).

- In Section 6.1 we discuss error reduction via repetition.
- In Section 6.2 we discuss the gap between non-adaptive and adaptive soundness.
- In Section 6.3 we show how to achieve small argument size with an inefficient honest prover.
- In Section 6.4 we prove a lower bound on argument size.
- In Section 6.5 we prove a limitation on the amount of public randomness.
- In Section 6.6 we prove a limitation on a strong notion of zero knowledge.
- In Section 6.7 we discuss zero knowledge simulation on instances that are not in the language.

6.1 Error reduction

The soundness error of a non-interactive argument can be reduced by running multiple independent executions, with each execution using an independent random oracle derived via domain separation from the given random oracle.

Construction 6.1.1. Given a non-interactive argument $(\mathcal{P}, \mathcal{V})$ and a repetition parameter $k \in \mathbb{N}$, define the non-interactive argument $(\mathcal{P}_k, \mathcal{V}_k)$ as follows:

- $\mathcal{P}_k^f(\mathbf{x}, \mathbf{w})$:
 1. For every $i \in [k]$, compute the argument string $\pi_i := \mathcal{P}^{f(i,\cdot)}(\mathbf{x}, \mathbf{w})$.
 2. Output $\pi := (\pi_i)_{i \in [k]}$.
- $\mathcal{V}_k^f(\mathbf{x}, \pi)$:
 1. Parse π as a tuple $(\pi_i)_{i \in [k]}$.
 2. For every $i \in [k]$, check that $\mathcal{V}^{f(i,\cdot)}(\mathbf{x}, \pi_i) = 1$.

Above, $f(i, \cdot)$ denotes the fact that the oracle f is accessed by prefixing every query with a string of $\lceil \log_2 k \rceil$ bits that uniquely encodes the index $i \in [k]$.

The argument size increases by a multiplicative factor of k , because each argument string of $(\mathcal{P}_k, \mathcal{V}_k)$ contains k argument strings of $(\mathcal{P}, \mathcal{V})$. Moreover, if $(\mathcal{P}, \mathcal{V})$ has zero-knowledge error z_{ARG} then $(\mathcal{P}_k, \mathcal{V}_k)$ has zero-knowledge error $k \cdot z_{\text{ARG}}$ because each of the argument strings “leaks” independently of the other. The lemma below shows that, on the other hand, soundness improves *exponentially* in the repetition parameter k .

Lemma 6.1.2. *If $(\mathcal{P}, \mathcal{V})$ is a non-interactive argument for a relation \mathcal{R} then, for every repetition parameter $k \in \mathbb{N}$, $(\mathcal{P}_k, \mathcal{V}_k)$ in Construction 6.2.1 is a non-interactive argument for \mathcal{R} such that:*

- if $(\mathcal{P}, \mathcal{V})$ has non-adaptive soundness error ϵ_{ARG} then $(\mathcal{P}_k, \mathcal{V}_k)$ has non-adaptive soundness error ϵ_{ARG}^* such that $\epsilon_{\text{ARG}}^*(\sigma, t, \mathbf{x}) \leq \epsilon_{\text{ARG}}(\sigma, t, \mathbf{x})^k$; and

- if $(\mathcal{P}, \mathcal{V})$ has adaptive soundness error ϵ_{ARG} then $(\mathcal{P}_k, \mathcal{V}_k)$ has adaptive soundness error ϵ_{ARG}^* such that $\epsilon_{\text{ARG}}^*(\sigma, t, n) \leq \epsilon_{\text{ARG}}(\sigma, t, n)^k$.

Proof. We prove each item of the lemma via similar, but distinct, proofs.

The non-adaptive case. Fix a security parameter $\sigma \in \mathbb{N}$, query bound $t \in \mathbb{N}$, t -query malicious argument prover $\tilde{\mathcal{P}}$, and instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$. We can write:

$$\begin{aligned} & \Pr \left[\mathcal{V}_k^f(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ &= \Pr \left[\forall i \in [k] \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \mathcal{V}^{f(i,\cdot)}(\mathbf{x}, \pi_i) = 1 \quad (\pi_i)_{i \in [k]} \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ &= \Pr \left[\forall i \in [k] \mid \begin{array}{l} \forall i \in [k], f_i \leftarrow \mathcal{U}(\sigma) \\ \mathcal{V}^{f_i}(\mathbf{x}, \pi_i) = 1 \quad (\pi_i)_{i \in [k]} \leftarrow \tilde{\mathcal{P}}^{f_1, \dots, f_k} \end{array} \right] \\ &= \prod_{i \in [k]} \Pr \left[\begin{array}{l} \mathcal{V}^{f_i}(\mathbf{x}, \pi_i) = 1 \\ \text{conditioned on} \\ \forall j < i, \mathcal{V}^{f_j}(\mathbf{x}, \pi_j) = 1 \end{array} \mid \begin{array}{l} \forall i \in [k], f_i \leftarrow \mathcal{U}(\sigma) \\ (\pi_i)_{i \in [k]} \leftarrow \tilde{\mathcal{P}}^{f_1, \dots, f_k} \end{array} \right]. \end{aligned}$$

Each probability in the product is upper bounded by $\epsilon_{\text{ARG}}(\sigma, t, \mathbf{x})$. Indeed, for every $i \in [k]$, the conditioning in the i -th event affects only the oracles f_1, \dots, f_{i-1} , which are sampled independently from the oracle f_i . Hence, for every $i \in [k]$,

$$\Pr \left[\begin{array}{l} \mathcal{V}^{f_i}(\mathbf{x}, \pi_i) = 1 \\ \text{conditioned on} \\ \forall j < i, \mathcal{V}^{f_j}(\mathbf{x}, \pi_j) = 1 \end{array} \mid \begin{array}{l} \forall i \in [k], f_i \leftarrow \mathcal{U}(\sigma) \\ (\pi_i)_{i \in [k]} \leftarrow \tilde{\mathcal{P}}^{f_1, \dots, f_k} \end{array} \right] \leq \epsilon_{\text{ARG}}(\sigma, t, \mathbf{x}).$$

We conclude that

$$\begin{aligned} & \prod_{i \in [k]} \Pr \left[\begin{array}{l} \mathcal{V}^{f_i}(\mathbf{x}, \pi_i) = 1 \\ \text{conditioned on} \\ \forall j < i, \mathcal{V}^{f_j}(\mathbf{x}, \pi_j) = 1 \end{array} \mid \begin{array}{l} \forall i \in [k], f_i \leftarrow \mathcal{U}(\sigma) \\ (\pi_i)_{i \in [k]} \leftarrow \tilde{\mathcal{P}}^{f_1, \dots, f_k} \end{array} \right] \\ & \leq \prod_{i \in [k]} \epsilon_{\text{ARG}}(\sigma, t, \mathbf{x}) \\ & = \epsilon_{\text{ARG}}(\sigma, t, \mathbf{x})^k. \end{aligned}$$

The adaptive case. We follow the same logic as the non-adaptive case, while allowing the prover to choose the instance. Fix an output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, t -query malicious argument prover $\tilde{\mathcal{P}}$, and instance size bound $n \in \mathbb{N}$. We can write:

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}_k^f(\mathbf{x}, \pi) = 1 \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ &= \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \forall i \in [k], \mathcal{V}^{f(i,\cdot)}(\mathbf{x}, \pi_i) = 1 \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, (\pi_i)_{i \in [k]}) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ &= \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \forall i \in [k], \mathcal{V}^{f_i}(\mathbf{x}, \pi_i) = 1 \end{array} \mid \begin{array}{l} \forall i \in [k], f_i \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, (\pi_i)_{i \in [k]}) \leftarrow \tilde{\mathcal{P}}^{f_1, \dots, f_k} \end{array} \right] \end{aligned}$$

$$= \prod_{i \in [k]} \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^{f_i}(\mathbf{x}, \pi_i) = 1 \\ \text{conditioned on} \\ \forall j < i, \mathcal{V}^{f_j}(\mathbf{x}, \pi_j) = 1 \end{array} \middle| \begin{array}{l} \forall i \in [k], f_i \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, (\pi_i)_{i \in [k]}) \leftarrow \tilde{\mathcal{P}}^{f_1, \dots, f_k} \end{array} \right].$$

Each probability in the product is upper bounded by $\epsilon_{\text{ARG}}(\sigma, t, n)$. Indeed, for every $i \in [k]$, the conditioning in the i -th event affects only the oracles f_1, \dots, f_{i-1} , which are sampled independently from the oracle f_i . Hence, for every $i \in [k]$,

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^{f_i}(\mathbf{x}, \pi_i) = 1 \\ \text{conditioned on} \\ \forall j < i, \mathcal{V}^{f_j}(\mathbf{x}, \pi_j) = 1 \end{array} \middle| \begin{array}{l} \forall i \in [k], f_i \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, (\pi_i)_{i \in [k]}) \leftarrow \tilde{\mathcal{P}}^{f_1, \dots, f_k} \end{array} \right] \leq \epsilon_{\text{ARG}}(\sigma, t, n).$$

We conclude that

$$\begin{aligned} & \prod_{i \in [k]} \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^{f_i}(\mathbf{x}, \pi_i) = 1 \\ \text{conditioned on} \\ \forall j < i, \mathcal{V}^{f_j}(\mathbf{x}, \pi_j) = 1 \end{array} \middle| \begin{array}{l} \forall i \in [k], f_i \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, (\pi_i)_{i \in [k]}) \leftarrow \tilde{\mathcal{P}}^{f_1, \dots, f_k} \end{array} \right] \\ & \leq \prod_{i \in [k]} \epsilon_{\text{ARG}}(\sigma, t, n) \\ & = \epsilon_{\text{ARG}}(\sigma, t, n)^k. \end{aligned}$$

□

Remark 6.1.3 (error reduction in practice). Lemma 6.1.2 shows that using multiple independent executions of a non-interactive argument reduces soundness error at an exponential rate, from ϵ_{ARG} to ϵ_{ARG}^k with k executions; this is at the cost of increasing the argument size by a factor of k . However, this method is rarely used in practice because typically soundness error can be reduced in a more effective way by modifying the construction of the non-interactive argument.

For example, suppose that a non-interactive argument has argument size σ and soundness error $\frac{t^2}{2^\sigma}$. The method of independent executions increases the argument size to $k \cdot \sigma$ and reduces the soundness error to $(\frac{t^2}{2^\sigma})^k = \frac{t^{2k}}{2^{k\sigma}}$. Alternatively, one can increase the output size of the random oracle from σ to $k \cdot \sigma$. This achieves the same argument size but a much better soundness error, $\frac{t^2}{2^{k\sigma}}$.

In a similar fashion, the soundness error of non-interactive arguments that we study in this book can be reduced by directly increasing the output of the random oracle and improving the soundness error of an underlying probabilistic proof. This is better than multiple independent executions.

We conclude by noting that the method of independent executions does *not* increase the query budget needed to attack the non-interactive argument. This is apparent in the regime where $\epsilon_{\text{ARG}}(\sigma, t, n)$ is very close to 1: in this case $\epsilon_{\text{ARG}}(\sigma, t, n)^k$ is also close to 1. In other words, when considering ϵ_{ARG} as a function of t , the amplification process reduces the error for the

part of the function that is sufficiently far from one, while causing minimal impact on the part close to 1. (In contrast, modifying the parameters of a non-interactive argument, such as in the example above, also increases the query budget needed to attack.)

6.2 Non-adaptive vs. adaptive security

We explain how any non-interactive argument that satisfies non-adaptive soundness (Definition 4.1.2) can be modified to achieve adaptive soundness (Definition 4.1.3), while preserving argument size (and essentially all other efficiency measures). This approach has two drawbacks: (a) it incurs a multiplicative loss of t (the query bound on the malicious prover) in the soundness error; and (b) it requires including the instance in every query (or at least a hash of it). These additional costs (especially the first) are undesirable in practice and, fortunately, every construction that we study is directly proved adaptively sound (and knowledge sound) without incurring the multiplicative loss or including the instance in every query (but only in a certain query that is carefully chosen).

Construction 6.2.1. Given a non-interactive argument $(\mathcal{P}, \mathcal{V})$, define the non-interactive argument $(\mathcal{P}_*, \mathcal{V}_*)$ as follows:

- $\mathcal{P}_*^f(\mathbf{x}, \mathbf{w}) := \mathcal{P}^{f(\mathbf{x}, \cdot)}(\mathbf{x}, \mathbf{w})$. I.e., \mathcal{P}_* simulates \mathcal{P} , with each query prefixed by the instance \mathbf{x} .
- $\mathcal{V}_*^f(\mathbf{x}, \pi) := \mathcal{V}^{f(\mathbf{x}, \cdot)}(\mathbf{x}, \pi)$. I.e., \mathcal{V}_* simulates \mathcal{V} , with each query prefixed by the instance \mathbf{x} .

Lemma 6.2.2. If $(\mathcal{P}, \mathcal{V})$ is a non-interactive argument for a relation \mathcal{R} with non-adaptive soundness error ϵ_{ARG} then $(\mathcal{P}_*, \mathcal{V}_*)$ in Construction 6.2.1 is a non-interactive argument for \mathcal{R} with adaptive soundness error ϵ_{ARG}^* such that

$$\epsilon_{\text{ARG}}^*(\sigma, t, n) \leq (t+1) \cdot \max \{ \epsilon_{\text{ARG}}(\sigma, t, \mathbf{x}) \mid \mathbf{x} \in \{0,1\}^* \text{ with } |\mathbf{x}| \leq n \text{ and } \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \}.$$

Proof. Fix an output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, and instance size bound $n \in \mathbb{N}$.

Let $\tilde{\mathcal{P}}_*$ be a t -query malicious prover that wins the adaptive soundness game of the non-interactive argument $(\mathcal{P}_*, \mathcal{V}_*)$ with probability δ :

$$\delta := \Pr \left[\begin{array}{l} |\mathbf{x}_*| \leq n \\ \wedge \mathbf{x}_* \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}_*^f(\mathbf{x}_*, \pi) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}_*, \pi) \leftarrow \tilde{\mathcal{P}}_*^f \end{array} \right].$$

Below we construct a t -query malicious prover $\tilde{\mathcal{P}}$ that convinces \mathcal{V} to accept instances \mathbf{x} of size n not in $\mathcal{L}(\mathcal{R})$, which are chosen *independent* of the random oracle, with probability at least $\frac{\delta}{t+1}$:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \text{aux}) \leftarrow \tilde{\mathcal{P}} \\ f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \tilde{\mathcal{P}}^f(\text{aux}) \end{array} \right] \geq \frac{\delta}{t+1}.$$

The non-adaptive soundness of $(\mathcal{P}, \mathcal{V})$ tells us that the probability that a t -query malicious prover convinces \mathcal{V} to accept a fixed instance \mathbf{x} not in $\mathcal{L}(\mathcal{R})$ is at most $\epsilon_{\text{ARG}}(\sigma, t, \mathbf{x})$. We deduce that

$$\max \{ \epsilon_{\text{ARG}}(\sigma, t, \mathbf{x}) \mid \mathbf{x} \in \{0,1\}^* \text{ with } |\mathbf{x}| \leq n \text{ and } \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \} \geq \frac{\delta}{t+1}$$

and conclude that $\epsilon_{\text{ARG}}^*(\sigma, t, n) \leq (t+1) \cdot \max \{ \epsilon_{\text{ARG}}(\sigma, t, \mathbf{x}) \mid \mathbf{x} \in \{0,1\}^* \text{ with } |\mathbf{x}| \leq n \text{ and } \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \}$.

We are left to construct and analyze $\tilde{\mathcal{P}}$. We construct $\tilde{\mathcal{P}}$ as follows:

- $\tilde{\mathcal{P}}$ outputs an instance (and private state) without access to the random oracle:
 1. Sample an internal random oracle $f' \leftarrow \mathcal{U}(\sigma)$.
 2. Sample $i \in [t+1]$ at random.
 3. Set $S \leftarrow \emptyset$.
 4. Simulate $\tilde{\mathcal{P}}_*$ while answering each query (y, x) as follows:
 - a) If $y \notin S$, then add y to S .
 - b) If $|S| = i$ then output $\mathbf{x} := y$ and $\mathbf{aux} := (\mathbf{x}, \perp)$ (this query is not answered yet).
 - c) Otherwise, reply with $f'(y, x)$.
 5. If $i = t+1$ then finish the simulation of $\tilde{\mathcal{P}}_*$, obtain its final output (\mathbf{x}_*, π) and output $\mathbf{x} := \mathbf{x}_*$, and $\mathbf{aux} := (\mathbf{x}, \pi)$.
- $\tilde{\mathcal{P}}$, given access to the random oracle f and input $\mathbf{aux} = (\mathbf{x}, \pi)$, outputs an argument string:
 6. If $\pi \neq \perp$, then output (\mathbf{x}, π) and halt.
 7. If $\pi = \perp$, continue the simulation of $\tilde{\mathcal{P}}_*$ while answering each query (y, x) as follows: if $y = \mathbf{x}$ then answer with $f(x)$; otherwise answer with $f'(y, x)$. When $\tilde{\mathcal{P}}_*$ halts and outputs (\mathbf{x}_*, π) , output π .

Claim 6.2.3. *It holds that $\Pr[\mathbf{x} = \mathbf{x}_*] \geq \frac{1}{t+1}$.*

Proof. The malicious prover $\tilde{\mathcal{P}}_*$ performs queries of the form (y, x) . If $y = \mathbf{x}$ then $\tilde{\mathcal{P}}$ answers the query with the external random oracle; else if $y \neq \mathbf{x}$ then $\tilde{\mathcal{P}}$ answers the query with the internal random oracle. Overall $\tilde{\mathcal{P}}$ answers queries of $\tilde{\mathcal{P}}_*$ like a random oracle would. This means that $\tilde{\mathcal{P}}_*$ has no information about the index i . Since $|S| \leq t$ and either $\mathbf{x}_* \in S$ or \mathbf{x}_* appears first in the final output, we get that $\Pr[\mathbf{x} = \mathbf{x}_*] \geq 1/(t+1)$. \square

Whenever $\mathbf{x} = \mathbf{x}_*$, the answers to every query of the form (\mathbf{x}_*, x) is $f(x)$ (namely, it is given by external random oracle), and the answers to every query of the form (y, x) with $y \neq \mathbf{x}_*$ is given by $f'(y, x)$ (namely, it is given by the internal random oracle). Therefore, in this case, if $\tilde{\mathcal{P}}_*$ convinces \mathcal{V}_* on instance \mathbf{x}_* then $\tilde{\mathcal{P}}$ convinces \mathcal{V} on instance \mathbf{x}_* . Moreover, the event that $\mathbf{x} = \mathbf{x}_*$ depends only on the random choice of i , and is independent of the success probability of $\tilde{\mathcal{P}}_*$. Thus, the probability that verifier accepts the prover's proof and that $\mathbf{x} = \mathbf{x}_*$ can be written as the product of these two probabilities. We conclude that

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \mathbf{aux}) \leftarrow \tilde{\mathcal{P}} \\ f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}) \end{array} \right] \\ & \geq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}_*^f(\mathbf{x}, \pi) = 1 \\ \wedge \mathbf{x}_* = \mathbf{x} \end{array} \middle| \begin{array}{l} (\mathbf{x}, \mathbf{aux}) \leftarrow \tilde{\mathcal{P}} \\ f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}_*, \pi) \leftarrow \tilde{\mathcal{P}}_*^f \end{array} \right] \\ & \geq \Pr \left[\begin{array}{l} |\mathbf{x}_*| \leq n \\ \wedge \mathbf{x}_* \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}_*^f(\mathbf{x}_*, \pi) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}_*, \pi) \leftarrow \tilde{\mathcal{P}}_*^f \end{array} \right] \cdot \Pr \left[\mathbf{x}_* = \mathbf{x} \middle| \begin{array}{l} (\mathbf{x}, \mathbf{aux}) \leftarrow \tilde{\mathcal{P}} \\ f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}_*, \pi) \leftarrow \tilde{\mathcal{P}}_*^f \end{array} \right] \end{aligned}$$

$$\geq \delta \cdot \frac{1}{t+1}.$$

□

Remark 6.2.4 (separation for adaptivity). Establishing adaptive soundness is meaningful because adaptive soundness is strictly stronger than non-adaptive soundness. We show this by describing a non-interactive argument that is non-adaptively sound but is not adaptively sound.

Let \mathcal{R} be the empty relation, which implies that $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ for every \mathbf{x} . (The analysis below works also for non-empty relations, as long as they are sparse enough.) Let $(\mathcal{P}, \mathcal{V})$ be a non-interactive argument with non-adaptive soundness error ϵ_{ARG} for \mathcal{R} . Define $(\mathcal{P}_*, \mathcal{V}_*)$ to be the non-interactive argument that is defined as follows:

- $\mathcal{P}_*^f(\mathbf{x}, \mathbf{w}) := \mathcal{P}^f(\mathbf{x}, \mathbf{w})$.
- $\mathcal{V}_*^f(\mathbf{x}, \pi)$: If $f(0) = \mathbf{x}$ then accept; otherwise output $\mathcal{V}^f(\mathbf{x}, \pi)$.

Namely, the argument prover \mathcal{P}_* equals the argument prover \mathcal{P} while the argument verifier \mathcal{V}_* relaxes the argument verifier \mathcal{V} by directly accepting any instance \mathbf{x} that equals the output of the random oracle on input 0.

The completeness property is preserved: \mathcal{V}_* accepts at least the same inputs as \mathcal{V} , and \mathcal{P}_* outputs the same argument string as \mathcal{P} . Moreover, the non-adaptive soundness error of $(\mathcal{P}_*, \mathcal{V}_*)$ is at most $\epsilon_{\text{ARG}}(\sigma, t, \mathbf{x}) + \frac{1}{2^\sigma}$. Indeed, for every fixed choice of instance \mathbf{x} not in $\mathcal{L}(\mathcal{R})$, a t -query malicious prover can convince \mathcal{V}_* to accept \mathbf{x} either by producing an argument string that convinces \mathcal{V} to accept \mathbf{x} or it happens to be the case that $f(0) = \mathbf{x}$. The probability that the first case happens is at most $\epsilon_{\text{ARG}}(\sigma, t, \mathbf{x})$, and the probability that the second case happens is exactly $\frac{1}{2^\sigma}$ (regardless of what the malicious prover does).

On the other hand, $(\mathcal{P}_*, \mathcal{V}_*)$ is adaptively insecure: there is a malicious argument prover that breaks adaptive soundness with probability 1 using one query to the random oracle. Indeed, a malicious argument prover can query the random oracle to obtain $\mathbf{x} := f(0)$, which is not in $\mathcal{L}(\mathcal{R})$ (since the relation \mathcal{R} is empty). Then, the malicious argument prover sets the argument string $\pi := \perp$, and outputs (\mathbf{x}, π) . The argument verifier \mathcal{V}_* accepts this instance and argument string with probability 1 (over the choice of random oracle f).

6.3 Construction with an inefficient prover

We are interested in SNARGs where the honest prover and honest verifier are efficient. We show that constructing a SNARG with an *inefficient* honest prover is straightforward.

Construction 6.3.1. Consider the non-interactive argument NARG = $(\mathcal{P}, \mathcal{V})$ constructed as follows. The argument prover \mathcal{P} receives as input an instance \mathbf{x} and witness \mathbf{w} , and the argument verifier \mathcal{V} receives as input the instance \mathbf{x} . Both receive query access to a random oracle $f \in \mathcal{U}(\sigma)$.

- $\mathcal{P}^f(\mathbf{x}, \mathbf{w})$:
 1. Find (by exhaustive search) $x, x' \in \{0, 1\}^{\sigma+1}$ such that $f(x) = f(x')$ and $x \neq x'$.
 2. Output the argument string $\pi := (x, x')$.
- $\mathcal{V}^f(\mathbf{x}, \pi)$:

1. Parse the argument string π as a pair (x, x') with $x, x' \in \{0, 1\}^{\sigma+1}$.
2. Check that $x \neq x'$ and $f(x) = f(x')$.

Lemma 6.3.2. *For every relation \mathcal{R} , NARG in Construction 6.3.1 is a non-interactive argument for \mathcal{R} with adaptive soundness error*

$$\epsilon_{\text{ARG}}(\sigma, t, n) \leq \frac{1}{2} \cdot \frac{t^2}{2^\sigma}.$$

The argument size is $s = 2 \cdot (\sigma + 1)$ and the argument verifier makes $q_V = 2$ random oracle queries.

Given a query bound $t \in \mathbb{N}$, ensuring that the soundness error $\epsilon_{\text{ARG}}(\sigma, t, n)$ is at most a target $\epsilon_0 \in (0, 1)$ requires setting the security parameter σ to $\log \frac{t^2}{2\epsilon_0}$. In this case, the argument size is

$$s = 2 \cdot (\sigma + 1) = 2 \cdot \left(\log \frac{t^2}{2\epsilon_0} + 1 \right) = 2 \cdot \log \frac{t^2}{\epsilon_0} \leq 4 \cdot \log \frac{t}{\epsilon_0}.$$

As discussed in Section 6.4, this argument size is essentially optimal. For example, if we set $t := 2^{128}$ and $\epsilon_0 := 2^{-128}$ (a conservative parameter choice), then the argument size s is 768 bits (equivalently, 96 bytes).

Proof of Lemma 6.3.2. First we prove completeness (Definition 4.1.1). For every $f \in \mathcal{U}(\sigma)$, f has 2^σ distinct outputs, less than the number of inputs in $\{0, 1\}^{\sigma+1}$; so there exist inputs $x, x' \in \{0, 1\}^{\sigma+1}$ such that $x \neq x'$ and $f(x) = f(x')$. The argument prover \mathcal{P} , which can query f any number of times, can find such a pair and the argument verifier \mathcal{V} accepts it. Hence, regardless of the input instance \mathbf{x} and witness \mathbf{w} , \mathcal{P} convinces \mathcal{V} with probability 1 over the choice of random oracle f .

Next we prove adaptive soundness (Definition 4.1.3). The argument verifier \mathcal{V} ignores the input instance \mathbf{x} , and checks that the argument string π contains a valid collision. For every malicious t -query argument prover $\tilde{\mathcal{P}}$, by Lemma 3.3.1 the probability that the query-answer trace of $\tilde{\mathcal{P}}$ contains $x, x' \in \{0, 1\}^{\sigma+1}$ such that $x \neq x'$ and $f(x) = f(x')$ is at most $\frac{1}{2} \cdot \frac{(t-1) \cdot t}{2^\sigma}$. Moreover, if $\tilde{\mathcal{P}}$ outputs (x, x') such that either was not queried, then the probability that $f(x) = f(x')$ is $\frac{1}{2^\sigma}$ (this follows from Lemma 3.1.1). Together, we upper bound the success probability of $\tilde{\mathcal{P}}$ as follows:

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ & \leq \Pr \left[\begin{array}{l} f(x) = f(x') \\ \wedge x \neq x' \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, (x, x')) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ & \leq \frac{1}{2} \cdot \frac{(t-1) \cdot t}{2^\sigma} + \frac{1}{2^\sigma} \\ & \leq \frac{1}{2} \cdot \frac{t^2}{2^\sigma}. \end{aligned}$$

□

Remark 6.3.3 (knowledge soundness). The non-interactive argument with an inefficient prover in Construction 6.3.1 satisfies (even adaptive) soundness but does it also satisfy knowledge soundness?

Superficially the answer may appear to be no. Extraction should work for every prover that convinces the verifier (with high enough probability) and, in particular, for the honest prover. In this non-interactive argument, the honest prover searches for a collision, completely ignoring the input instance and witness; hence, the honest prover's output and query-answer trace contain no information about the witness. It seems that an extractor would have no chance to infer any information about valid witnesses for the given instance, when given the honest prover's input-output behavior and corresponding query-answer traces.

Nevertheless, the non-interactive argument *does* satisfy (even adaptive) knowledge soundness. This is because, as we proved, no malicious argument prover can convince the argument verifier to accept with probability more than $\frac{1}{2} \cdot \frac{t^2}{2^\sigma}$, even for an adaptively chosen instance \mathbf{x} (regardless of whether \mathbf{x} is in $\mathcal{L}(\mathcal{R})$ or not). Therefore, we upper bound the (adaptive) knowledge soundness error as $\kappa_{\text{ARG}}(\sigma, t, n) \leq \frac{1}{2} \cdot \frac{t^2}{2^\sigma}$, and let the knowledge extractor do nothing. The knowledge extractor is never tasked with extracting a witness as no malicious argument prover convinces the argument verifier with more than the aforementioned knowledge soundness error.

Remark 6.3.4 (zero knowledge). The non-interactive argument with an inefficient prover in Construction 6.3.1 can be modified to additionally satisfy zero knowledge (with perfect simulation). The honest argument prover finds by exhaustive search and outputs a pair (x, x') with $x, x' \in \{0, 1\}^{\sigma+1}$ such that $x \neq x'$ and $f(x) = f(x')$. To facilitate simulation, we modify the honest argument prover to uniformly sample pairs (x, x') until a collision is found. The simulator can then sample and output a pair (x, x') (conditioned on $x \neq x'$) and *program* the random oracle to satisfy $f(x) = f(x')$. The output of this simulator is identically distributed to the output of the honest argument prover.

Remark 6.3.5 (improving argument size). The argument size can be improved, at the cost of introducing a small completeness error. Informally, the idea is to challenge the prover to solve an inversion problem rather than a collision problem.

In more detail, for a parameter $\ell \in \mathbb{N}$, consider the argument verifier \mathcal{V} such that $\mathcal{V}^f(\mathbf{x}, \pi)$ checks that $\pi \in \{0, 1\}^{\sigma+\ell}$ and $f(\pi) = 0^\sigma$. The corresponding (honest) argument prover \mathcal{P} is tasked with finding a suitable argument string π to convince the verifier.

It is straightforward to show that the (adaptive) soundness error of this non-interactive argument is $\epsilon_{\text{ARG}}(\sigma, t, n) \leq \frac{t}{2^\sigma}$. Moreover, the argument size is $s = \sigma + \ell$ and the argument verifier makes $\mathbf{q}_\mathcal{V} = 1$ random oracle query.

Given a query bound $t \in \mathbb{N}$, ensuring that the soundness error $\epsilon_{\text{ARG}}(\sigma, t, n)$ is at most a target $\epsilon_0 \in (0, 1)$ requires setting the security parameter σ to $\log \frac{t}{\epsilon_0}$. In this case, the argument size is

$$\sigma + \ell = \log \frac{t}{\epsilon_0} + \ell.$$

However, it can be that 0^σ has no inverses in which case the honest argument prover fails. The probability that f maps a fixed input to 0^σ is $2^{-\sigma}$, independently of other inputs. So the probability that f does *not* map any input in $\{0, 1\}^{\sigma+\ell}$ to 0^σ (i.e., the honest argument prover fails) is

$$\left(1 - \frac{1}{2^\sigma}\right)^{2^{\sigma+\ell}} \leq e^{-\ell}.$$

6.4 Lower bound on argument size

We describe a lower bound on the argument size of any non-interactive argument scheme. We prove that any non-interactive argument in the random oracle model must have argument size $s = \Omega(\log \frac{t}{q_V \cdot \epsilon_{\text{ARG}}(\sigma, t, \mathbf{x})})$, where q_V is the number of queries performed by the argument verifier and ϵ_{ARG} is the non-adaptive soundness error. This lower bound does *not* depend on the number of queries q_P performed by the (honest) argument prover, so it holds even when the (honest) argument prover is inefficient.

The lower bound applies to any argument that is *non-trivial*. Indeed, an argument for an easy relation could have argument size $s = 0$ (and argument verifier query complexity $q_V = 0$), because the argument verifier may decide membership in the corresponding language by itself (without the help of the prover and without querying the random oracle). As explained in Remark 6.4.3, requiring that the argument is non-trivial, according to the definition below, is necessary for the relation to be hard.

Definition 6.4.1. A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} with argument size s is **non-trivial** if there exists an instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ such that for every $\sigma \in \mathbb{N}$

$$\Pr [\exists \pi \in \{0, 1\}^s : \mathcal{V}^f(\mathbf{x}, \pi) = 1 \mid f \leftarrow \mathcal{U}(\sigma)] \geq 1/2.$$

Lemma 6.4.2. Let $\text{NARG} = (\mathcal{P}, \mathcal{V})$ be a non-interactive argument for a relation \mathcal{R} with non-adaptive soundness error ϵ_{ARG} (Definition 4.1.2) and argument verifier query complexity q_V .¹ If NARG is non-trivial then there exists an instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ such that, for every query bound $t \in \mathbb{N}$ and random oracle output size $\sigma \in \mathbb{N}$ with $\epsilon_{\text{ARG}}(\sigma, t, \mathbf{x}) < 1/2$, the argument size of NARG satisfies

$$s \geq \frac{1}{4} \cdot \log \frac{t}{q_V \cdot \epsilon_{\text{ARG}}(\sigma, t, \mathbf{x})}.$$

Proof. Fix an instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ for which the non-triviality condition holds, and fix $\sigma, t \in \mathbb{N}$. For any such instance, we prove that $s \geq \log \frac{t}{q_V}$ and $s \geq \log \frac{1}{2\epsilon_{\text{ARG}}(\sigma, t, \mathbf{x})}$. We conclude that

$$s \geq \max \left\{ \log \frac{t}{q_V}, \log \frac{1}{2\epsilon_{\text{ARG}}} \right\} \geq \frac{1}{4} \cdot \log \frac{t}{q_V \cdot \epsilon_{\text{ARG}}(\sigma, t, \mathbf{x})}.$$

- $s \geq \log \frac{t}{q_V}$: Assume towards contradiction that $s < \log \frac{t}{q_V}$. Consider the t -query cheating prover $\tilde{\mathcal{P}}$ that, given a random oracle f , works as follows: (i) for every $\pi \in \{0, 1\}^s$, if $\mathcal{V}^f(\mathbf{x}, \pi) = 1$ then output π and halt; otherwise, (ii) output $\pi := \perp$. The number of queries made by $\tilde{\mathcal{P}}$ is at most $2^s \cdot q_V < (t/q_V) \cdot q_V = t$. By construction of $\tilde{\mathcal{P}}$, for every $f \in \mathcal{U}(\sigma)$, if there exists $\pi \in \{0, 1\}^s$ such that $\mathcal{V}^f(\mathbf{x}, \pi) = 1$ then $\tilde{\mathcal{P}}$ finds and outputs π . Hence, using the non-triviality condition, we can lower bound the success probability of $\tilde{\mathcal{P}}$:

$$\begin{aligned} &\Pr \left[\mathcal{V}^f(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ &= \Pr [\exists \pi \in \{0, 1\}^s : \mathcal{V}^f(\mathbf{x}, \pi) = 1 \mid f \leftarrow \mathcal{U}(\sigma)] \geq 1/2. \end{aligned}$$

We deduce that $\epsilon_{\text{ARG}}(\sigma, t, \mathbf{x}) \geq 1/2$, which contradicts the assumption that $\epsilon_{\text{ARG}}(\sigma, t, \mathbf{x}) < 1/2$.

¹And possibly imperfect completeness (the probability in Definition 4.1.1 is above a threshold rather than 1).

- $s \geq \log \frac{1}{2\epsilon_{\text{ARG}}(\sigma, t, \mathbf{x})}$: Consider the cheating prover $\tilde{\mathcal{P}}$ that outputs a random argument string in $\{0, 1\}^s$. Condition on the event that there exists $\pi \in \{0, 1\}^s$ such that $\mathcal{V}^f(\mathbf{x}, \pi) = 1$, which by the non-triviality condition holds with probability at least $1/2$; in this case, the probability that $\tilde{\mathcal{P}}$ outputs π is 2^{-s} . Thus, $\tilde{\mathcal{P}}$ convinces \mathcal{V} to accept \mathbf{x} with probability at least $1/2 \cdot 2^{-s}$. Since the soundness error is $\epsilon_{\text{ARG}}(\sigma, t, \mathbf{x})$, we conclude that $1/2 \cdot 2^{-s} \leq \epsilon_{\text{ARG}}(\sigma, t, \mathbf{x})$.

□

Remark 6.4.3 (trivial arguments). We explain that if a relation \mathcal{R} has a *trivial* argument $(\mathcal{P}, \mathcal{V})$ (it does not satisfy the condition in Definition 6.4.1) then the relation can be decided via a fast probabilistic algorithm. This tells us that the relation cannot be hard.

Consider the probabilistic algorithm that, on input an instance \mathbf{x} , works as follows. First, lazily sample an oracle $f \leftarrow \mathcal{U}(\sigma)$. Then, enumerate over all strings $\pi \in \{0, 1\}^s$, and check for each one if $\mathcal{V}^f(\mathbf{x}, \pi) = 1$. If there exists $\pi \in \{0, 1\}^s$ such that $\mathcal{V}^f(\mathbf{x}, \pi) = 1$ then output 1; otherwise output 0.

Since $(\mathcal{P}, \mathcal{V})$ is trivial, for every $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ it holds that

$$\Pr \left[\exists \pi \in \{0, 1\}^s : \mathcal{V}^f(\mathbf{x}, \pi) = 1 \mid f \leftarrow \mathcal{U}(\sigma) \right] < 1/2.$$

We analyze the success probability of the probabilistic algorithm.

- If $\mathbf{x} \in \mathcal{L}(\mathcal{R})$ then, by the completeness property, with high probability over the random oracle (in fact with probability 1 if the argument is perfectly complete), there exists $\pi \in \{0, 1\}^s$ such that $\mathcal{V}^f(\mathbf{x}, \pi) = 1$ and thus the algorithm outputs 1.
- If $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ then, by the triviality assumption, with probability at least $1/2$ there exists no $\pi \in \{0, 1\}^s$ such that $\mathcal{V}^f(\mathbf{x}, \pi) = 1$, so the algorithm outputs 0 with probability at least $1/2$.

The running time of the algorithm is $2^s \cdot \text{poly}(\sigma, \text{len}(\mathbf{x}))$.

If s is small then the relation \mathcal{R} is not hard. For example, the (randomized) exponential-time hypothesis states that 3SAT cannot be decided by probabilistic algorithms in time $2^{o(n)}$. Provided $s = o(n)$ (i.e., s is small), a trivial argument for 3SAT disproves this conjecture. (The regime $s = \Omega(n)$ is not interesting because a satisfying assignment consists of n bits.)

Remark 6.4.4 (the interactive case). A non-interactive argument is a special case of an interactive argument. Hence Lemma 6.4.2 provides, in particular, a lower bound on the communication complexity of any interactive argument.

Remark 6.4.5 (tightness). The non-interactive argument with an inefficient honest argument prover in Section 6.3 tells us that the lower bound in Lemma 6.4.2 cannot be improved (up to constants), unless we introduce additional assumptions (e.g., the honest argument prover is efficient). Indeed, that non-interactive argument achieves argument size $s = O(\log \frac{t}{\epsilon_{\text{ARG}}})$ with $\mathbf{q}_V = O(1)$, while the lower bound implies that $s = \Omega(\log \frac{t}{\epsilon_{\text{ARG}}})$ when $\mathbf{q}_V = O(1)$.

6.5 Limitation on public randomness

A random oracle is a large amount of public randomness, available to all (honest and malicious) parties. Relying on less randomness would be more convenient though (no cryptographic hash function heuristically realizing the random oracle would be needed). We explain why non-interactive arguments cannot rely on little randomness (in the information-theoretic setting of this book).²

Impossibility in the URSM A simple form of public randomness is a *uniform random string*: for a size $\sigma \in \mathbb{N}$, all (honest and malicious) parties receive as input a random string sampled from $\{0, 1\}^\sigma$. We should think of σ as polynomially bounded, because efficient parties must have enough time to read the entire random σ -bit string.

One could define non-interactive arguments in the *uniform random string model* (URSM). We do not do this explicitly, but the definitions for the URSM can be directly obtained from the definitions for the ROM in Section 4.1 by replacing the random oracle $f \leftarrow \mathcal{U}(\sigma)$ with a uniform random string $\text{urs} \leftarrow \{0, 1\}^\sigma$ (and viewing urs as an explicit input or as a degenerate oracle that answers every query with urs). Note that, for the resulting notion of a non-interactive argument in the URSM, query budgets are not relevant anymore, and so malicious parties are unbounded in every relevant resource (as we are in an information-theoretic setting).

It would be (far) more desirable to construct succinct non-interactive arguments in the URSM rather than in the ROM (since the URSM does not suffer from the realization challenges of the ROM discussed in Sections 2.4 and 2.5). However it is straightforward to prove that succinct non-interactive arguments in the URSM do not exist for interesting (i.e., hard) relations.

We argue this only informally.

A non-interactive argument in the URSM for a relation \mathcal{R} is *non-trivial* if there exists an instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ such that for every size $\sigma \in \mathbb{N}$ the probability, over the choice of $\text{urs} \in \{0, 1\}^\sigma$, that there exists an argument string that convinces the argument verifier to accept \mathbf{x} is at least $1/2$. This non-triviality notion is analogous to Definition 6.4.1 (which is in the ROM); and similarly to Remark 6.4.3 one can show that if a non-interactive argument for \mathcal{R} is trivial then it can be decided in probabilistic time $2^s \cdot \text{poly}(\sigma, \text{len}(\mathbf{x}))$, where s is the argument size. (In which case s cannot be small.)

For a non-interactive argument that is non-trivial, a malicious argument prover (who has unbounded resources) can find and output an argument string that convinces the argument verifier on the instance not in the language guaranteed by the non-triviality condition, if such an argument string exists; and such an argument string exists with probability at least $1/2$.

We deduce that a non-interactive argument in the URSM is trivial or has large soundness error.

A random oracle must be “big” The above discussion justifies why one considers a model where there are multiple uniform random strings rather a single one.

A similar reasoning as above also tells us that if a malicious argument prover has enough query budget to query every uniform random string available then it can conduct the same

²If we allow for computational hardness assumptions, and correspondingly limit adversaries to be computationally bounded, one can achieve interesting goals about argument systems by relying on less (or even no) public randomness.

attack, again leading us to conclude that if the public randomness contains too few uniform random strings then a non-interactive argument must be trivial or have large soundness error.

Below we formulate the above intuition in a broader sense. The notion of random oracle that we consider includes infinitely many uniform random strings (as the random oracle takes as input queries of any length). However, if the argument verifier considers only a small part of the random oracle, a malicious argument prover can again perform an attack like the above one. We describe a brute-force attack on any non-interactive argument where the argument verifier performs queries of bounded size. We deduce that the argument verifier must rely on query the random oracle at many locations (i.e., rely on many uniform random strings in the public randomness), telling us that succinct non-interactive arguments in the ROM indeed must rely on the availability of a large amount of public randomness.

Lemma 6.5.1. *Let $\text{NARG} = (\mathcal{P}, \mathcal{V})$ be a non-interactive argument for a relation \mathcal{R} with non-adaptive soundness error ϵ_{ARG} . Suppose that NARG is non-trivial (Definition 6.4.1) and the largest query size used by the argument verifier is ℓ . There exists an instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ such that for every $\sigma \in \mathbb{N}$ it holds that $\epsilon_{\text{ARG}}(\sigma, 2^\ell, \mathbf{x}) \geq 1/2$.*

Proof. Fix an instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ for which the non-triviality condition holds. Consider the t -query malicious argument prover $\tilde{\mathcal{P}}$ that, given a random oracle f , works as follows: (i) for every $\pi \in \{0, 1\}^s$, if $\mathcal{V}^f(\mathbf{x}, \pi) = 1$ then output π and halt; otherwise, (ii) output $\pi := \perp$. The number of queries made by $\tilde{\mathcal{P}}$ is at most 2^ℓ as that is the total number of possible queries to the random oracle with query size at most ℓ . By construction of $\tilde{\mathcal{P}}$, for every $f \in \mathcal{U}(\sigma)$, if there exists $\pi \in \{0, 1\}^s$ such that $\mathcal{V}^f(\mathbf{x}, \pi) = 1$ then $\tilde{\mathcal{P}}$ finds and outputs π . Hence, using the non-triviality condition, we can lower bound the success probability of $\tilde{\mathcal{P}}$:

$$\begin{aligned} & \Pr \left[\mathcal{V}^f(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ &= \Pr \left[\exists \pi \in \{0, 1\}^s : \mathcal{V}^f(\mathbf{x}, \pi) = 1 \mid f \leftarrow \mathcal{U}(\sigma) \right] \geq 1/2. \end{aligned}$$

We deduce that $\epsilon_{\text{ARG}}(\sigma, 2^\ell, \mathbf{x}) \geq 1/2$. □

6.6 Limitation on zero knowledge

Zero knowledge in the ROM can be defined in several ways. Here we prove that a simple and natural definition of zero knowledge is too strong, which explains why weaker, yet still meaningful, definitions are typically used instead (such as those discussed in Section 5.2).

In more detail, we prove that “simulation is as hard as decision” for what is called *zero knowledge in the non-programmable random oracle model* in cryptography literature.

In this notion of zero knowledge, the simulator cannot change answers of the random oracle in order to produce a simulated proof and, in particular, the simulator and the adversary have access to the same random oracle.³

Intuitively, this notion is too strong because the simulator has no structural advantage compared to a malicious argument prover, and so if a non-interactive argument did have a simulator then that simulator can be used to decide the language. Since the simulator is efficient, we deduce that the language is “easy”, and thus trivially has zero knowledge proofs.

³This is in contrast to the definitions that we consider in Section 5.2, where the simulator can change answers of the random oracle, in such a way that the adversary does not notice the difference, possibly up to a small error.

In more detail, the lemma below tells us that if a non-interactive argument is zero knowledge in the non-programmable random oracle model, then we can obtain an efficient probabilistic algorithm to decide the language. In particular, we should not expect there to exist non-interactive arguments with this type of zero knowledge for languages of interest (e.g., NP-complete languages).

Lemma 6.6.1. *Let $\text{NARG} = (\mathcal{P}, \mathcal{V})$ be a non-interactive argument for a relation \mathcal{R} with (non-adaptive) soundness error ϵ_{ARG} , where the argument verifier \mathcal{V} makes $\mathbf{q}_{\mathcal{V}}$ queries to the random oracle and runs in time $\text{Time}_{\mathcal{V}}$. Suppose that there exists a probabilistic algorithm \mathcal{S} such that for every output size $\sigma \in \mathbb{N}$ of the random oracle, query bound $t \in \mathbb{N}$, t -query algorithm \mathcal{A} , and instance-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, the following two distributions are $z_{\text{ARG}}(\sigma, t, \mathbf{x})$ -close in statistical distance:*

$$\left\{ \mathcal{A}^f(\pi) \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \mathcal{P}^f(\mathbf{x}, \mathbf{w}) \end{array} \right\} \text{ and } \left\{ \mathcal{A}^f(\pi) \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \mathcal{S}^f(\mathbf{x}) \end{array} \right\}.$$

If \mathcal{S} makes at most $t_{\mathcal{S}}$ queries and runs in time $\text{Time}_{\mathcal{S}}$ then, for every $\sigma \in \mathbb{N}$, $\mathcal{L}(\mathcal{R})$ can be decided via a probabilistic algorithm that runs in time $\text{Time}_{\mathcal{S}} + \text{Time}_{\mathcal{V}}$ and that has false negative error at most $z_{\text{ARG}}(\sigma, \mathbf{q}_{\mathcal{V}}, \mathbf{x})$ and false positive error at most $\epsilon_{\text{ARG}}(\sigma, t_{\mathcal{S}}, \mathbf{x})$.

Proof. Consider the probabilistic algorithm D that, given an instance \mathbf{x} , works as follows:

1. Lazily sample an oracle $\dot{f} \leftarrow \mathcal{U}(\sigma)$.
2. Run the simulator to sample an argument string $\pi \leftarrow \mathcal{S}^{\dot{f}}(\mathbf{x})$.
3. Compute and output $\mathcal{V}^{\dot{f}}(\mathbf{x}, \pi)$.

The running time of D is the running time of \mathcal{S} plus the running time of \mathcal{V} . Moreover, for every instance \mathbf{x} ,

$$\Pr[D(\mathbf{x}) = 1] = \Pr \left[\mathcal{V}^{\dot{f}}(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \mathcal{S}^f(\mathbf{x}) \end{array} \right]. \quad (6.1)$$

We analyze the error probability of D on the instance \mathbf{x} . If $\mathbf{x} \in \mathcal{L}(\mathcal{R})$ then we want $D(\mathbf{x})$ to output 1 with high probability (equivalently, bound the false negative error); and if $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ then we want $D(\mathbf{x})$ to output 1 with low probability (equivalently, bound the false positive error).

- *Case 1: $\mathbf{x} \in \mathcal{L}(\mathcal{R})$.* We argue that $\Pr[D(\mathbf{x}) = 1] \geq 1 - z_{\text{ARG}}(\sigma, \mathbf{q}_{\mathcal{V}}, \mathbf{x})$. Fix any witness \mathbf{w} such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$. By completeness of $(\mathcal{P}, \mathcal{V})$,

$$\Pr \left[\mathcal{V}^{\dot{f}}(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \mathcal{P}^f(\mathbf{x}, \mathbf{w}) \end{array} \right] = 1.$$

Hence, using the above equality and Equation 6.1, we can write

$$\begin{aligned} & |1 - \Pr[D(\mathbf{x}) = 1]| \\ &= \left| \Pr \left[\mathcal{V}^{\dot{f}}(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \mathcal{P}^f(\mathbf{x}, \mathbf{w}) \end{array} \right] - \Pr \left[\mathcal{V}^{\dot{f}}(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \mathcal{S}^f(\mathbf{x}) \end{array} \right] \right| . \end{aligned}$$

This absolute value is at most $z_{\text{ARG}}(\sigma, \mathbf{q}_{\mathcal{V}}, \mathbf{x})$ because we can use the (non-adaptive) zero-knowledge property of $(\mathcal{P}, \mathcal{V})$ on the argument verifier \mathcal{V} , which we view as an adversary that makes at most $\mathbf{q}_{\mathcal{V}}$ queries to the random oracle. We conclude that the probability that $D(\mathbf{x}) = 1$ is at least $1 - z_{\text{ARG}}(\sigma, \mathbf{q}_{\mathcal{V}}, \mathbf{x})$.

- *Case 2:* $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$. We can view the simulator as an adversary that makes at most t_S queries to the random oracle and therefore, by Equation 6.1, $\Pr[D(\mathbf{x}) = 1] \leq \epsilon_{\text{ARG}}(\sigma, t_S, \mathbf{x})$.

□

Remark 6.6.2. We do not expect Lemma 6.6.1 to hold if the simulator can program the random oracle. Indeed, this can be observed by the proof. In Case 2, the simulator cannot be considered an adversary as the simulator programs the random oracle, an act that an adversary cannot perform. Hence, we would not be able to conclude that $\Pr[D(\mathbf{x}) = 1] \leq \epsilon_{\text{ARG}}(\sigma, t_S, \mathbf{x})$.

6.7 Simulation on instances not in the language

The zero-knowledge property of a non-interactive argument states that argument strings produced by the simulator are statistically close to argument strings produced by the (honest) argument prover, for instances accompanied by valid witnesses (which means that the instances are in the language). But what happens if the simulator is invoked on instances that are *not* in the language?

Intuitively, if the simulator produces argument strings that do not convince the argument verifier then the language is easy to decide. Indeed, if an instance is in the language then the simulator produces convincing proofs. Hence for hard languages we expect the simulator to produce convincing proofs.

In more detail, the lemma below states that the probability that the simulator produces convincing proofs captures the error for deciding the language. In particular, for a hard language, we expect this to hold with all but negligible probability. Note that it suffices to consider non-adaptive zero-knowledge (Definition 5.2.1).

Lemma 6.7.1. *Let $\text{NARG} = (\mathcal{P}, \mathcal{V})$ be a non-interactive argument for a relation \mathcal{R} with (non-adaptive) zero-knowledge error z_{ARG} . Let \mathcal{S} be the zero-knowledge simulator, and suppose that for instances $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ it produces convincing proofs with probability δ :*

$$\delta(\sigma, \mathbf{x}) := \Pr \left[\mathcal{V}^{f[\mu]}(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\pi, \mu) \leftarrow \mathcal{S}^f(\mathbf{x}) \end{array} \right].$$

If \mathcal{S} makes at most t_S queries and runs in time $\text{Time}_{\mathcal{S}}$ then, for every $\sigma \in \mathbb{N}$, $\mathcal{L}(\mathcal{R})$ can be decided via a probabilistic algorithm that runs in time $\text{Time}_{\mathcal{S}} + \text{Time}_{\mathcal{V}}$ and that has false negative error at most $z_{\text{ARG}}(\sigma, \mathbf{q}_{\mathcal{V}}, \mathbf{x})$ and false positive error at most $\delta(\sigma, \mathbf{x})$.

The proof of the lemma is subtly different from that of Lemma 6.6.1. We deliberately write the proof with similar structure and notation, to facilitate a comparison between the two. The main difference is that in Case 1 we adjust the argument to account for the fact that the simulator programs the oracle and in Case 2 we rely on the hypothesis that the simulator produces convincing proofs with probability δ .

Proof. Consider the probabilistic algorithm D that, given an instance \mathbf{x} , works as follows:

1. Lazily sample an oracle $\dot{f} \leftarrow \mathcal{U}(\sigma)$.
2. Run the simulator to sample an argument string and programmed values $(\pi, \mu) \leftarrow \mathcal{S}^{\dot{f}}(\mathbf{x})$.
3. Compute and output $\mathcal{V}^{\dot{f}}(\mathbf{x}, \pi)$.

The running time of D is the running time of \mathcal{S} plus the running time of \mathcal{V} . Moreover, for every instance \mathbf{x} ,

$$\Pr[D(\mathbf{x}) = 1] = \Pr \left[\mathcal{V}^{f[\mu]}(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\pi, \mu) \leftarrow \mathcal{S}^f(\mathbf{x}) \end{array} \right]. \quad (6.2)$$

We analyze the error probability of D on the instance \mathbf{x} . If $\mathbf{x} \in \mathcal{L}(\mathcal{R})$ then we want $D(\mathbf{x})$ to output 1 with high probability (equivalently, bound the false negative error); and if $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ then we want $D(\mathbf{x})$ to output 1 with low probability (equivalently, bound the false positive error).

- *Case 1:* $\mathbf{x} \in \mathcal{L}(\mathcal{R})$. We argue that $\Pr[D(\mathbf{x}) = 1] \geq 1 - z_{\text{ARG}}(\sigma, \mathbf{q}_{\mathcal{V}}, \mathbf{x})$. Fix any witness \mathbf{w} such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$. By completeness of $(\mathcal{P}, \mathcal{V})$,

$$\Pr \left[\mathcal{V}^f(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \mathcal{P}^f(\mathbf{x}, \mathbf{w}) \end{array} \right] = 1.$$

Hence, using the above equality and Equation 6.2, we can write

$$\begin{aligned} & |1 - \Pr[D(\mathbf{x}) = 1]| = \\ & \left| \Pr \left[\mathcal{V}^f(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \mathcal{P}^f(\mathbf{x}, \mathbf{w}) \end{array} \right] - \Pr \left[\mathcal{V}^{f[\mu]}(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\pi, \mu) \leftarrow \mathcal{S}^f(\mathbf{x}) \end{array} \right] \right| \end{aligned}$$

This absolute value is at most $z_{\text{ARG}}(\sigma, \mathbf{q}_{\mathcal{V}}, \mathbf{x})$ because we can use the (non-adaptive) zero-knowledge property of $(\mathcal{P}, \mathcal{V})$ on the argument verifier \mathcal{V} , which we view as an adversary that makes at most $\mathbf{q}_{\mathcal{V}}$ queries to the random oracle. We conclude that the probability that $D(\mathbf{x}) = 1$ is at least $1 - z_{\text{ARG}}(\sigma, \mathbf{q}_{\mathcal{V}}, \mathbf{x})$.

- *Case 2:* $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$. By Equation 6.2, $\Pr[D(\mathbf{x}) = 1] = \delta(\sigma, \mathbf{x})$.

□

Remark 6.7.2 (simulators in this book). All simulators that we construct in this book are such that they output argument strings that convince the argument verifier to accept (provided that the simulator does not abort, which happens with small probability). This is regardless of whether the input instance is in the language (corresponding to the relation) or not. This follows by inspection of the simulators constructed in Sections 16.2, 22.2 and 26.2. (And given that the simulator for the underlying probabilistic proof samples accepting views for the verifier of the probabilistic proof.)

7 Arguments with multiple random oracles

The definitions for a non-interactive argument in Section 4.1 consider the setting where (honest and malicious) parties have query access to a *single* random oracle f sampled from $\mathcal{U}(\sigma)$. In this book we often work with a relaxed notion, where parties have query access to *multiple* random oracles $(f_i)_{i \in [k]}$, possibly with different output sizes $(\ell_i)_{i \in [k]}$. In this chapter we explain how this multiple-oracle setting directly follows from the single-oracle setting with no security loss, by using the oracle derivation techniques in Section 2.6; in particular, the number of oracles and their output sizes can be specified *after* the single oracle has been sampled. The takeaway is that we can feel free to use multiple oracles when convenient, as they can be “implemented” via a single oracle.

7.1 Definitions

We provide security definitions for non-interactive arguments in the setting of multiple random oracles. In comparison to the single-oracle setting, the main difference is that the number and output sizes of the random oracles depend on two parameters: a security parameter $\lambda \in \mathbb{N}$ and an instance size bound $n \in \mathbb{N}$. Both parameters are known to the argument prover and argument verifier, but we suppress them as inputs to simplify notation. In more detail, definitions of a non-interactive argument with multiple oracles are obtained from those with a single oracle by replacing the line “ $f \leftarrow \mathcal{U}(\sigma)$ ” with the line “ $f \leftarrow \mathcal{U}(\text{cnf}(\lambda, n))$ ” in a probability experiment, where cnf is an *oracle configuration function* that, given a security parameter λ and instance size bound n , specifies the output sizes $(\ell_i)_{i \in [k]}$ for the oracles to be sampled; thus, f sampled from $\mathcal{U}(\text{cnf}(\lambda, n)) = \mathcal{U}((\ell_i)_{i \in [k]})$ is a vector $(f_i)_{i \in [k]}$ where each random oracle f_i has output size ℓ_i . Below we list formal statements of the security definitions for the multiple-oracle setting, which extend the definitions for the single-oracle setting introduced in Section 4.1 and Chapter 5.

Definition 7.1.1 (oracle configuration). *Let cnf be a function that receives as input a security parameter $\lambda \in \mathbb{N}$ and an instance size bound $n \in \mathbb{N}$ and outputs a list of output sizes $(\ell_i)_{i \in [k]}$. A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ has **oracle configuration** cnf if the argument prover \mathcal{P} and argument verifier \mathcal{V} have query access to random oracles sampled from $\mathcal{U}(\text{cnf}(\lambda, n))$, when using security parameter λ and instances \mathbf{x} with $|\mathbf{x}| \leq n$.*

Definition 7.1.2 (completeness). *A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} with oracle configuration cnf has **(perfect) completeness** if for every security parameter $\lambda \in \mathbb{N}$, instance size bound $n \in \mathbb{N}$, and instance-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ with $|\mathbf{x}| \leq n$,*

$$\Pr \left[\mathcal{V}^f(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\text{cnf}(\lambda, n)) \\ \pi \leftarrow \mathcal{P}^f(\mathbf{x}, \mathbf{w}) \end{array} \right] = 1.$$

Definition 7.1.3 (non-adaptive soundness). *A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} with oracle configuration cnf has **non-adaptive soundness error** ϵ_{ARG} if for every*

security parameter $\lambda \in \mathbb{N}$, instance size bound $n \in \mathbb{N}$, query bound $t \in \mathbb{N}$, t -query malicious argument prover $\tilde{\mathcal{P}}$, and instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ with $|\mathbf{x}| \leq n$,

$$\Pr \left[\mathcal{V}^f(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\text{cnf}(\lambda, n)) \\ \pi \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \leq \epsilon_{\text{ARG}}(\lambda, t, \mathbf{x}).$$

Definition 7.1.4 (adaptive soundness). A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} with oracle configuration cnf has **adaptive soundness error** ϵ_{ARG} if for every security parameter $\lambda \in \mathbb{N}$, instance size bound $n \in \mathbb{N}$, query bound $t \in \mathbb{N}$, and t -query malicious argument prover $\tilde{\mathcal{P}}$,

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\text{cnf}(\lambda, n)) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \leq \epsilon_{\text{ARG}}(\lambda, t, n).$$

Definition 7.1.5. A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has **straight-line knowledge soundness error** κ_{ARG} if there exists a polynomial-time deterministic algorithm \mathcal{E} (the extractor) such that for every security parameter $\lambda \in \mathbb{N}$, instance size bound $n \in \mathbb{N}$, query bound $t \in \mathbb{N}$, and t -query deterministic argument prover $\tilde{\mathcal{P}}$,

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\text{cnf}(\lambda, n)) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ \mathbf{w} \leftarrow \mathcal{E}(\mathbf{x}, \pi, \text{tr}) \end{array} \right] \leq \kappa_{\text{ARG}}(\lambda, t, n).$$

Definition 7.1.6. Let $\text{NARG} = (\mathcal{P}, \mathcal{V})$ be a non-interactive argument. A deterministic argument prover $\tilde{\mathcal{P}}$ has **failure probability** $\delta_{\tilde{\mathcal{P}}}$ if for every security parameter $\lambda \in \mathbb{N}$ and instance size bound $n \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} |\mathbf{x}| > n \\ \vee \mathcal{V}^f(\mathbf{x}, \pi) = 0 \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\text{cnf}(\lambda, n)) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \leq \delta_{\tilde{\mathcal{P}}}(\lambda, n).$$

Definition 7.1.7. A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has **rewinding knowledge soundness error** κ_{ARG} with **extraction time** et_{ARG} if there exists a probabilistic algorithm \mathcal{E} (the extractor) such that for every security parameter $\lambda \in \mathbb{N}$, instance size bound $n \in \mathbb{N}$, query bound $t \in \mathbb{N}$, and t -query deterministic argument prover $\tilde{\mathcal{P}}$ with failure probability $\delta_{\tilde{\mathcal{P}}}$ and running time $\tau_{\tilde{\mathcal{P}}}$,

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\text{cnf}(\lambda, n)) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}^f(\mathbf{x}, \pi) \\ \mathbf{w} \leftarrow \mathcal{E}(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}}) \end{array} \right] \leq \kappa_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}}(\lambda, n)).$$

Moreover, \mathcal{E} runs in expected time $\text{et}_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}}(\lambda, n), \tau_{\tilde{\mathcal{P}}}(\lambda, n))$ (over the given inputs and internal randomness).

Definition 7.1.8. A non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has **adaptive zero-knowledge error** z_{ARG} (in the EPROM) if there exists a probabilistic polynomial-time simulator \mathcal{S} such that, for every security parameter $\lambda \in \mathbb{N}$, instance bound $n \in \mathbb{N}$, query bound

$t \in \mathbb{N}$, and t -query admissible adversary \mathcal{A} , the following two distributions are $z_{\text{ARG}}(\lambda, t, n)$ -close in statistical distance:

$$\mathcal{D}_{\text{real}} := \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}(\text{cnf}(\lambda, n)) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}^f \\ \pi \leftarrow \mathcal{P}^f(\mathbf{x}, \mathbf{w}) \\ \text{out} \leftarrow \mathcal{A}^f(\text{aux}, \pi) \end{array} \right. \right\} \quad \text{and} \quad \mathcal{D}_{\text{sim}} := \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}(\text{cnf}(\lambda, n)) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}^f \\ (\pi, \mu) \leftarrow \mathcal{S}^f(\mathbf{x}) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\text{aux}, \pi) \end{array} \right. \right\}.$$

Above, \mathcal{A} is admissible if it always outputs \mathbf{x}, \mathbf{w} such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ and $|\mathbf{x}| \leq n$.

7.2 Multiple oracles based on a single oracle

We explain how to reduce the multiple-oracle setting to the single-oracle setting for a non-interactive argument. Specifically, we use the procedure M2S (multi to single) from Construction 2.6.1 to modify a non-interactive argument that uses multiple oracles into a non-interactive argument that uses a single oracle.

Construction 7.2.1. Let $(\mathcal{P}, \mathcal{V})$ be a non-interactive argument that uses oracles $(f_i)_{i \in [k]}$ sampled from $\mathcal{U}((\ell_i)_{i \in [k]})$. We construct a non-interactive argument $(\mathcal{P}_*, \mathcal{V}_*)$ that uses an oracle $f \in \mathcal{U}(\sigma)$.

- $\mathcal{P}_*^f(\mathbf{x}, \mathbf{w}) := \text{M2S}(\mathcal{P})^f(\mathbf{x}, \mathbf{w})$.
- $\mathcal{V}_*^f(\mathbf{x}, \pi) := \text{M2S}(\mathcal{V})^f(\mathbf{x}, \pi)$.

If \mathcal{P} makes at most $\mathbf{q}_\mathcal{P}$ queries to $(f_i)_{i \in [k]}$ then \mathcal{P}_* makes at most $(\max_{i \in [k]} [\ell_i / \sigma]) \cdot \mathbf{q}_\mathcal{P}$ queries to f . If \mathcal{V} makes at most $\mathbf{q}_\mathcal{V}$ queries to $(f_i)_{i \in [k]}$ then \mathcal{V}_* make at most $(\max_{i \in [k]} [\ell_i / \sigma]) \cdot \mathbf{q}_\mathcal{V}$ queries to f .

Below we explain how $(\mathcal{P}_*, \mathcal{V}_*)$ inherits the security guarantees of $(\mathcal{P}, \mathcal{V})$, by invoking the lemmas proved in Section 2.6. We discuss, in turn, completeness, (adaptive) soundness, knowledge soundness (first straightline and then rewinding), and zero knowledge. In these discussions, we slightly overload notation and write: (a) M2S(tr) to mean the query-answer trace resulting from applying M2S from Construction 2.6.1 to any machine whose query-answer trace is tr ; and, similarly, (b) S2M(tr) to mean the query-answer trace resulting from applying S2M from Construction 2.6.3 to any machine whose query-answer trace is tr . Both operations are well-defined because M2S and S2M only modify the queries and answers of the input machine.

Completeness If $(\mathcal{P}, \mathcal{V})$ is (perfectly) complete then $(\mathcal{P}_*, \mathcal{V}_*)$ is (perfectly) complete. For every instance-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$,

$$\left\{ b \left| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \pi \leftarrow \mathcal{P}_*^f(\mathbf{x}, \mathbf{w}) \\ b = \mathcal{V}_*^f(\mathbf{x}, \pi) \end{array} \right. \right\} \equiv \left\{ b \left| \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]}) \\ \pi \leftarrow \mathcal{P}^{(f_i)_{i \in [k]}}(\mathbf{x}, \mathbf{w}) \\ b = \mathcal{V}^{(f_i)_{i \in [k]}}(\mathbf{x}, \pi) \end{array} \right. \right\}.$$

The above follows directly from Lemma 2.6.2: define M to be the right-side experiment and observe that, since the operation M2S is local (it transforms each query locally without sharing state), we can apply M2S to each step of the right-side experiment separately.

Soundness If $(\mathcal{P}, \mathcal{V})$ has (adaptive) soundness error $\epsilon_{\text{ARG}}(\sigma, t, n)$ then $(\mathcal{P}_*, \mathcal{V}_*)$ has (adaptive) soundness error $\epsilon_{\text{ARG}}(\sigma, t, n)$. Fix a query bound $t \in \mathbb{N}$ and t -query malicious argument prover $\tilde{\mathcal{P}}_*$ against \mathcal{V}_* , and instance \mathbf{x} . Then $\text{S2M}(\tilde{\mathcal{P}}_*)$, which makes at most t queries, has the same success probability against \mathcal{V} :

$$\left\{ b \wedge b' \left| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}_*^f \\ b = \mathcal{V}_*^f(\mathbf{x}, \pi) \\ b' = |\mathbf{x}| \leq n \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \end{array} \right. \right\} \equiv \left\{ b \wedge b' \left| \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \leftarrow \text{S2M}(\tilde{\mathcal{P}}_*)^{(f_i)_{i \in [k]}} \\ b = \mathcal{V}^{(f_i)_{i \in [k]}}(\mathbf{x}, \pi) \\ b' = |\mathbf{x}| \leq n \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \end{array} \right. \right\}.$$

The above follows directly from Lemma 2.6.4.

Straightline knowledge soundness If $(\mathcal{P}, \mathcal{V})$ has straightline knowledge soundness error $\kappa_{\text{ARG}}(\sigma, t, n)$ then $(\mathcal{P}_*, \mathcal{V}_*)$ has straightline knowledge soundness error $\kappa_{\text{ARG}}(\sigma, t, n)$. Let \mathcal{E} be the extractor for \mathcal{V} . Define the extractor \mathcal{E}_* for \mathcal{V}_* as follows:

$$\mathcal{E}_*(\mathbf{x}, \pi, \text{tr}) := \mathcal{E}(\mathbf{x}, \pi, \text{S2M}(\text{tr})).$$

Fix a query bound $t \in \mathbb{N}$ and t -query argument prover $\tilde{\mathcal{P}}_*$ against \mathcal{V}_* . First we argue that:

$$\left\{ (b, \mathbf{x}, \text{S2M}(\text{tr}_*)) \left| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}_*} \tilde{\mathcal{P}}_*^f \\ b = \mathcal{V}_*^f(\mathbf{x}, \pi) \end{array} \right. \right\} \equiv \left\{ (b, \mathbf{x}, \text{tr}) \left| \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \text{S2M}(\tilde{\mathcal{P}}_*)^{(f_i)_{i \in [k]}} \\ b = \mathcal{V}^{(f_i)_{i \in [k]}}(\mathbf{x}, \pi) \end{array} \right. \right\}.$$

The above follows from Lemma 2.6.4 as follows: define M_1 to run $\tilde{\mathcal{P}}_*$ and output its output (\mathbf{x}, π) and query-answer trace tr_* , and define M_2 to be the verifier \mathcal{V} . If the trace of $\text{S2M}(\tilde{\mathcal{P}})$ is tr and the trace of $\tilde{\mathcal{P}}_*$ is tr_* , then $\text{S2M}(\text{tr}_*) = \text{tr}$. Since the extractor \mathcal{E}_* depends only on $(\mathbf{x}, \pi, \text{S2M}(\text{tr}_*))$, we conclude that:

$$\left\{ (b, \mathbf{x}, w) \left| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}_*} \tilde{\mathcal{P}}_*^f \\ b = \mathcal{V}_*^f(\mathbf{x}, \pi) \\ w = \mathcal{E}_*(\mathbf{x}, \pi, \text{tr}_*) \end{array} \right. \right\} \equiv \left\{ (b, \mathbf{x}, w) \left| \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \text{S2M}(\tilde{\mathcal{P}}_*)^{(f_i)_{i \in [k]}} \\ b = \mathcal{V}^{(f_i)_{i \in [k]}}(\mathbf{x}, \pi) \\ w = \mathcal{E}(\mathbf{x}, \pi, \text{tr}) \end{array} \right. \right\}.$$

Rewinding knowledge soundness If $(\mathcal{P}, \mathcal{V})$ has rewinding knowledge soundness error $\kappa_{\text{ARG}}(\sigma, t, n, \delta_{\tilde{\mathcal{P}}})$ with extraction time $\text{et}_{\text{ARG}}(\sigma, t, n, \delta_{\tilde{\mathcal{P}}}, \tau_{\tilde{\mathcal{P}}})$ then $(\mathcal{P}_*, \mathcal{V}_*)$ has rewinding knowledge soundness error $\kappa_{\text{ARG}}(\sigma, t, n, \delta_{\tilde{\mathcal{P}}})$ with extraction time $\text{et}_{\text{ARG}}(\sigma, t, n, \delta_{\tilde{\mathcal{P}}}, \tau_{\tilde{\mathcal{P}}} + O(t \cdot \max_{i \in [k]} \{\ell_i\}))$. Fix a query bound $t \in \mathbb{N}$ and t -query argument prover $\tilde{\mathcal{P}}_*$ against \mathcal{V}_* . Let \mathcal{E} be the extractor for \mathcal{V} . Define the extractor \mathcal{E}_* for \mathcal{V}_* as follows:

$$\mathcal{E}_*(\mathbf{x}, \pi, \text{tr}, \tilde{\mathcal{P}}_*) := \mathcal{E}(\mathbf{x}, \pi, \text{S2M}(\text{tr}), \text{S2M}(\tilde{\mathcal{P}}_*)).$$

Since S2M is probabilistic, we explicitly write $\tilde{\mathcal{P}} \leftarrow \text{S2M}(\tilde{\mathcal{P}}_*)$ so that all invocations of $\tilde{\mathcal{P}}$ use the same randomness. Note that $\tilde{\mathcal{P}}$ can be realized given black-box access to $\tilde{\mathcal{P}}_*$.

From Lemma 2.6.4, we get that

$$\left\{ b \wedge b' \left| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}_*^f \\ b = \mathcal{V}_*^f(\mathbf{x}, \pi) \\ b' = |\mathbf{x}| \leq n \end{array} \right. \right\} \equiv \left\{ b \wedge b' \left| \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]}) \\ \tilde{\mathcal{P}} \leftarrow \text{S2M}(\tilde{\mathcal{P}}_*) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^{(f_i)_{i \in [k]}} \\ b = \mathcal{V}^{(f_i)_{i \in [k]}}(\mathbf{x}, \pi) \\ b' = |\mathbf{x}| \leq n \end{array} \right. \right\}.$$

Therefore, the failure probabilities of $\tilde{\mathcal{P}}_*$ (see Definition 5.1.2) and of $\tilde{\mathcal{P}}$ (see Definition 7.1.6) are equal. Let tr_* be the trace of $\tilde{\mathcal{P}}_*$ to f . Then the trace of $\tilde{\mathcal{P}}$ to $(f_i)_{i \in [k]}$ is $\text{tr} = \text{S2M}(\text{tr}_*)$. Therefore, by Lemma 2.6.4 we get that

$$\left\{ b \wedge b' \left| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}_*} \tilde{\mathcal{P}}_*^f \\ b = \mathcal{V}_*^f(\mathbf{x}, \pi) \\ \mathbf{w} = \mathcal{E}_*(\mathbf{x}, \pi, \text{tr}_*, \tilde{\mathcal{P}}_*) \\ b' = |\mathbf{x}| \leq n \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \end{array} \right. \right\} \equiv \left\{ b \wedge b' \left| \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]}) \\ \tilde{\mathcal{P}} \leftarrow \text{S2M}(\tilde{\mathcal{P}}_*) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^{(f_i)_{i \in [k]}} \\ b = \mathcal{V}^{(f_i)_{i \in [k]}}(\mathbf{x}, \pi) \\ \mathbf{w} = \mathcal{E}(\mathbf{x}, \pi, \text{tr}, \tilde{\mathcal{P}}) \\ b' = |\mathbf{x}| \leq n \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \end{array} \right. \right\}.$$

The probability of the right side to satisfy $b \wedge b' = 1$ is upper bounded by the rewinding knowledge soundness error of the multi-oracle scheme $(\mathcal{P}, \mathcal{V})$ with respect to the prover \mathcal{P} (which has the same failure probability as $\tilde{\mathcal{P}}_*$).

Zero knowledge If $(\mathcal{P}, \mathcal{V})$ has adaptive zero-knowledge error $z_{\text{ARG}}(\sigma, t, n)$ then $(\mathcal{P}_*, \mathcal{V}_*)$ has adaptive zero-knowledge error $z_{\text{ARG}}(\sigma, t, n)$. Let \mathcal{S} be the simulator for \mathcal{P} . Define the simulator \mathcal{S}_* for \mathcal{P}_* as follows.

- $\mathcal{S}_*^f(\mathbf{x})$:
1. $(\pi, \mu) \leftarrow \text{M2S}(\mathcal{S})^f(\mathbf{x})$.
 2. Set $\mu_* := \text{M2S}(\mu)$.
 3. Output (π, μ_*) .

Fix a query bound $t \in \mathbb{N}$ and t -query admissible adversary \mathcal{A}_* (i.e., an adversary that always outputs \mathbf{x}, \mathbf{w} such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ and $|\mathbf{x}| \leq n$). Since S2M is probabilistic, we explicitly write $\mathcal{A} \leftarrow \text{S2M}(\mathcal{A}_*)$ so that all invocations of \mathcal{A} use the same randomness. By applying Lemma 2.6.5, we get

$$\left\{ y \left| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}_*^f \\ \pi \leftarrow \mathcal{P}_*^f(\mathbf{x}, \mathbf{w}) \\ y \leftarrow \mathcal{A}_*^f(\text{aux}, \pi) \end{array} \right. \right\} \equiv \left\{ y \left| \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]}) \\ \mathcal{A} \leftarrow \text{S2M}(\mathcal{A}_*) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}^{(f_i)_{i \in [k]}} \\ \pi \leftarrow \mathcal{P}^{(f_i)_{i \in [k]}}(\mathbf{x}, \mathbf{w}) \\ y \leftarrow \mathcal{A}^{(f_i)_{i \in [k]}}(\text{aux}, \pi) \end{array} \right. \right\}.$$

Then, since $(\mathcal{P}, \mathcal{V})$ is zero knowledge, the following two distributions are $z_{\text{ARG}}(\sigma, t, n)$ -close in statistical distance:

$$\left\{ y \left| \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]}) \\ \mathcal{A} \leftarrow \text{S2M}(\mathcal{A}_*) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}^{(f_i)_{i \in [k]}} \\ \pi \leftarrow \mathcal{P}^{(f_i)_{i \in [k]}}(\mathbf{x}, \mathbf{w}) \\ y \leftarrow \mathcal{A}^{(f_i)_{i \in [k]}}(\text{aux}, \pi) \end{array} \right. \right\} \text{ and } \left\{ y \left| \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]}) \\ \mathcal{A} \leftarrow \text{S2M}(\mathcal{A}_*) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}^{(f_i)_{i \in [k]}} \\ (\pi, \mu) \leftarrow \mathcal{S}^{(f_i)_{i \in [k]}}(\mathbf{x}) \\ y \leftarrow \mathcal{A}^{(f_i[\mu])_{i \in [k]}}(\text{aux}, \pi) \end{array} \right. \right\}.$$

Next, we apply Lemma 2.6.5 with an algorithm M that computes $(\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}^{(f_i)_{i \in [k]}}$, then computes $(\pi, \mu) \leftarrow \mathcal{S}^{(f_i)_{i \in [k]}}(\mathbf{x})$, and finally outputs $y \leftarrow \mathcal{A}^{(f_i[\mu])_{i \in [k]}}(\text{aux}, \pi)$. All queries by \mathcal{A} to a programmed location $f_i[\mu]$ are handled internally in M (given the programmed answer in μ), and thus are not exposed to M2S . However, this is equivalent to first computing $\mu_* = \text{M2S}(\mu)$,

and then exposing all queries to $f[\mu_\star]$. This gives us the equivalence of the following two distributions:

$$\left\{ y \middle| \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\ell_i)_{i \in [k]}) \\ \mathcal{A} \leftarrow \text{S2M}(\mathcal{A}_\star) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}^{(f_i)_{i \in [k]}} \\ (\pi, \mu) \leftarrow \mathcal{S}^{(f_i)_{i \in [k]}}(\mathbf{x}) \\ y \leftarrow \mathcal{A}^{(f_i[\mu])_{i \in [k]}}(\text{aux}, \pi) \end{array} \right\} \text{ and } \left\{ y \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \mathcal{A} \leftarrow \text{S2M}(\mathcal{A}_\star) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \text{M2S}(\mathcal{A})^{(f)} \\ (\pi, \mu) \leftarrow \text{M2S}(\mathcal{S})^f \\ \mu_\star \leftarrow \text{M2S}(\mu) \\ y \leftarrow \text{M2S}(\mathcal{A})^{f[\mu_\star]}(\text{aux}, \pi) \end{array} \right\}.$$

Finally, we note that the right-side distribution is same as computing $(\pi, \mu_\star) \leftarrow \mathcal{S}_\star^f$ (by the definition of \mathcal{S}_\star) and then running $y \leftarrow \mathcal{A}_\star^{f[\mu_\star]}(\pi)$. We conclude that following two distributions are $z_{\text{ARG}}(\sigma, t, n)$ -close in statistical distance:

$$\left\{ y \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}_\star^f \\ \pi \leftarrow \mathcal{P}_\star^f(\mathbf{x}, \mathbf{w}) \\ y \leftarrow \mathcal{A}_\star^f(\text{aux}, \pi) \end{array} \right\} \text{ and } \left\{ y \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}_\star^f \\ (\pi, \mu_\star) \leftarrow \mathcal{S}_\star^f \\ y \leftarrow \mathcal{A}_\star^{f[\mu_\star]}(\text{aux}, \pi) \end{array} \right\}.$$

Part II

NARGs Based on SPs

Overview

We introduce sigma protocols (SPs), which are simple interactive protocols between a prover and a verifier. Then we describe how to use a hash function (the random oracle) to transform any sigma protocol into a corresponding non-interactive argument (NARG), thereby “removing” the interaction from the sigma protocol. This involves several delicate technical details, and so we proceed in two steps: first we describe a warmup construction that achieves only non-adaptive security; and then we describe a modified construction that achieves adaptive security. Finally, we introduce state-restoration soundness for SPs, a stronger notion of soundness that tightly captures the security guarantees of the NARG in terms of the underlying SP. While not essential to the case of SPs, state restoration soundness plays a crucial role in later parts of this book, and it is useful to get acquainted with it in the simple setting of SPs.

8 Sigma protocols	74
8.1 Definition	74
9 Warmup: non-adaptive security	79
9.1 Construction	79
9.2 Lower bound on the non-adaptive soundness error	80
9.3 Non-adaptive soundness	81
9.4 Non-adaptive knowledge soundness	84
9.5 Non-adaptive zero knowledge	86
9.6 Lack of adaptive security	88
10 The Fiat–Shamir transformation for SPs	90
10.1 Construction	90
10.2 Adaptive soundness	91
11 Additional security definitions	94
11.1 Knowledge soundness	94
11.2 Zero knowledge	97
12 State restoration	99
12.1 Definition	99
12.2 Security from state restoration	102
12.3 Comparison with standard soundness notions	104

8 Sigma protocols

We introduce notations and definitions for *sigma protocols* (SPs), which are 3-message protocols between a prover and a verifier. The 3-message back-and-forth interaction, when visualized, is reminiscent of the letter Σ , and hence the name “sigma protocol”.

In subsequent chapters we study how to construct non-interactive arguments from SPs. This is known as the *Fiat–Shamir transformation* [FS86]. Later on in Part III, we study how to construct non-interactive arguments from public-coin IPs, a generalization of SPs. We discuss the special case of SPs first because we can introduce key ideas in a simpler setting. Moreover, there are SPs for interesting relations, which makes the special case of SPs of standalone interest.

8.1 Definition

A sigma protocol (SP) is a tuple of algorithms

$$\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$$

that works as follows. The SP prover \mathbf{P}_{SP} receives as input an instance \mathbf{x} and witness w , and the SP verifier \mathbf{V}_{SP} receives as input the instance \mathbf{x} . First the SP prover sends a first message $\alpha_1 \in \{0, 1\}^{\text{PV}_1}$ known as the *commitment*, computed as $(\alpha_1, \text{aux}) := \mathbf{P}_{\text{SP}}(\mathbf{x}, w)$. Then the SP verifier sends a random message $\rho \in \{0, 1\}^r$ known as the *challenge*. Finally the SP prover sends a second message $\alpha_2 := \mathbf{P}_{\text{SP}}(\text{aux}, \rho) \in \{0, 1\}^{\text{PV}_2}$ known as the *response*. The SP verifier accepts if and only if $\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2)$ outputs 1. See Figure 8.1 for a diagram of an SP.

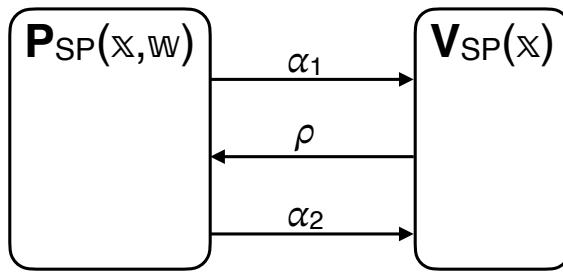


Figure 8.1: Diagram of an SP.

The tuple $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ is a SP for a relation \mathcal{R} with (*perfect*) *completeness* and *soundness error* ϵ_{SP} if it satisfies the two properties stated below.¹

¹This slightly deviates from the cryptography literature, where sigma protocols are assumed to satisfy stronger notions of security than the ones in this chapter. Specifically, in the cryptography literature, a sigma protocol satisfies *special soundness* (rather than merely standard soundness as in this chapter) and special honest-verifier zero-knowledge (rather than merely honest-verifier zero-knowledge as in this chapter). This choice is for consistency with other parts of the book, and we separately consider strong soundness notions like special soundness (in Chapter 30).

Definition 8.1.1. $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ for a relation \mathcal{R} has **perfect completeness** if for every $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$,

$$\Pr \left[\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \mid \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \mathbf{P}_{\text{SP}}(\mathbf{x}, \mathbf{w}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \mathbf{P}_{\text{SP}}(\text{aux}, \rho) \end{array} \right] = 1.$$

Definition 8.1.2. $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ for a relation \mathcal{R} has **soundness error** ϵ_{SP} if for every $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ and malicious SP prover $\tilde{\mathbf{P}}_{\text{SP}}$,

$$\Pr \left[\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \mid \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}} \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \end{array} \right] \leq \epsilon_{\text{SP}}(\mathbf{x}).$$

We additionally define $\epsilon_{\text{SP}}(n) := \max \{ \epsilon_{\text{SP}}(\mathbf{x}) \mid \mathbf{x} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \text{ and } \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \}$.

A malicious SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ is all-powerful, so the soundness condition in Definition 8.1.2 is equivalent to the following:

$$\forall \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \quad \forall \alpha_1 \in \{0, 1\}^{\text{PV}_1} \quad \Pr_{\rho \in \{0, 1\}^r} [\exists \alpha_2 \in \{0, 1\}^{\text{PV}_2} : \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1] \leq \epsilon_{\text{SP}}(\mathbf{x}). \quad (8.1)$$

Knowledge soundness The soundness notion (Definition 8.1.2) can be strengthened to a knowledge soundness notion, which informally means that whenever an SP prover convinces the SP verifier then an efficient extractor finds a witness, up to some error. In more detail, we require the existence of a probabilistic algorithm \mathbf{E}_{SP} that works as follows. Fix any instance \mathbf{x} and SP prover $\tilde{\mathbf{P}}_{\text{SP}}$. The SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ and SP verifier \mathbf{V}_{SP} (on input \mathbf{x}) interact; denote by $(\alpha_1, \rho, \alpha_2)$ the transcript of this interaction and by b the decision bit of the SP verifier \mathbf{V}_{SP} . The knowledge extractor \mathbf{E}_{SP} is tasked to find a witness \mathbf{w} when given as input the instance \mathbf{x} , the interaction transcript $(\alpha_1, \rho, \alpha_2)$, and black-box access to the SP prover $\tilde{\mathbf{P}}_{\text{SP}}$. This means that the knowledge extractor \mathbf{E}_{SP} is *rewinding*: it can re-run the SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ multiple times, possibly with correlated inputs. The knowledge soundness error is an upper bound on the probability that $b = 1$ ($\tilde{\mathbf{P}}_{\text{SP}}$ convinces \mathbf{V}_{SP}) and $(\mathbf{x}, \mathbf{w}) \notin \mathcal{R}$ (\mathbf{E}_{SP} does not find a witness). In general, the knowledge soundness error may be a function of the failure probability of $\tilde{\mathbf{P}}_{\text{SP}}$, which we define below.

Definition 8.1.3. Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP. A deterministic SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ has **failure probability** $\delta_{\tilde{\mathbf{P}}_{\text{SP}}}$ if for every instance \mathbf{x} :

$$\Pr \left[\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 0 \mid \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}} \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \end{array} \right] \leq \delta_{\tilde{\mathbf{P}}_{\text{SP}}}(\mathbf{x}).$$

Definition 8.1.4. $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ for a relation \mathcal{R} has **rewinding knowledge soundness error** κ_{SP} with extraction time et_{SP} if there exists a probabilistic algorithm \mathbf{E}_{SP} (the extractor) such that for every instance \mathbf{x} and deterministic SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ running in time $\tau_{\tilde{\mathbf{P}}_{\text{SP}}}$ the following holds:

$$\Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \mid \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}} \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}) \end{array} \right] \leq \kappa_{\text{SP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{SP}}}(\mathbf{x})).$$

Moreover, $\mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}})$ runs in expected time $\mathbf{et}_{\text{SP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{SP}}}(\mathbf{x}), \tau_{\tilde{\mathbf{P}}_{\text{SP}}}(\mathbf{x}))$ (over the given inputs and internal randomness). We additionally define

$$\begin{aligned}\kappa_{\text{SP}}(n, \delta_{\tilde{\mathbf{P}}_{\text{SP}}}) &:= \max \left\{ \kappa_{\text{SP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{SP}}}(\mathbf{x})) \mid \mathbf{x} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \right\}, \quad \text{and} \\ \mathbf{et}_{\text{SP}}(n, \delta_{\tilde{\mathbf{P}}_{\text{SP}}}, \tau_{\tilde{\mathbf{P}}_{\text{SP}}}) &:= \max \left\{ \mathbf{et}_{\text{SP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{SP}}}(\mathbf{x}), \tau_{\tilde{\mathbf{P}}_{\text{SP}}}(\mathbf{x})) \mid \mathbf{x} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \right\}.\end{aligned}$$

If $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ has knowledge soundness error κ_{SP} (with any extraction time \mathbf{et}_{SP}) then it has soundness error ϵ_{SP} such that $\epsilon_{\text{SP}}(\mathbf{x}) \leq \min_{\delta \in [0, 1]} \kappa_{\text{SP}}(\mathbf{x}, \delta)$: if $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ then the extractor cannot output a witness \mathbf{w} such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ (as none exists), in which case the event bounded by the knowledge soundness condition equals the event bounded by the soundness condition.

A notable special case of Definition 8.1.4 is *straightline extraction*: \mathbf{E}_{SP} is a polynomial-time deterministic algorithm that receives as input the instance \mathbf{x} and SP prover messages (α_1, α_2) (but not the SP verifier challenge ρ or access to \mathbf{P}_{SP}).

Zero knowledge An SP is *zero knowledge* if the SP prover, when interacting with the SP verifier, reveals no information beyond the fact that the proved statement is true. If the SP verifier is allowed to behave arbitrarily, then the property is called *malicious-verifier zero knowledge*; otherwise, if the SP verifier must follow the honest strategy, then the property is called *honest-verifier zero knowledge*. This latter (and weaker) property suffices for the applications that we consider, and so our definitions focus on that property only. The formal definition consists of two steps: (a) we formalize the *view* of the SP verifier, which is all the information that the SP verifier learns by interacting with the SP prover; (b) we say that the interactive proof is zero knowledge if the view of the SP verifier can be (approximately) simulated by a polynomial-time probabilistic algorithm, known as the *simulator*. The intuition here is that anything that is computable efficiently with randomness is “for free”, and so does not count as revealed knowledge. Crucially, the property is only required to hold for true statements, and the simulator is only given as input the statement (but not any “help” such as a witness, which may be hard to compute).

Definition 8.1.5. The SP verifier’s **view** in $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ on the instance-witness pair (\mathbf{x}, \mathbf{w}) , denoted $\text{View}_{\text{SP}}(\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}}, \mathbf{x}, \mathbf{w})$, is the random variable $(\mathbf{x}, \alpha_1, \rho, \alpha_2)$ where:

- $(\alpha_1, \text{aux}) \leftarrow \mathbf{P}_{\text{SP}}(\mathbf{x}, \mathbf{w})$;
- ρ is a random choice of randomness for the SP verifier \mathbf{V}_{SP} ; and
- $\alpha_2 \leftarrow \mathbf{P}_{\text{SP}}(\text{aux}, \rho)$.

Note that the honest SP prover $\mathbf{P}_{\text{SP}}(\mathbf{x}, \mathbf{w})$ may use its own private randomness (and is not part of the SP verifier’s view).

Definition 8.1.6. $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ for a relation \mathcal{R} has **honest-verifier zero-knowledge error** z_{SP} if there exists a polynomial-time probabilistic algorithm \mathbf{S}_{SP} such that for every instance-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ the following random variables are $z_{\text{SP}}(\mathbf{x})$ -close in statistical distance:

$$\text{View}_{\text{SP}}(\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}}, \mathbf{x}, \mathbf{w}) \text{ and } \mathbf{S}_{\text{SP}}(\mathbf{x}).$$

We additionally define $z_{\text{SP}}(n) := \max \left\{ z_{\text{SP}}(\mathbf{x}) \mid \mathbf{x}, \mathbf{w} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \text{ and } (\mathbf{x}, \mathbf{w}) \in \mathcal{R} \right\}$.

Remark 8.1.7 (comparison with prior definitions of knowledge soundness). Definition 8.1.4 strengthens definitions of rewinding knowledge soundness commonly used in the literature.

Definition 8.1.4 considers a single probability experiment in which the prover is run and subsequently the extractor receives as input the transcript of interaction and black-box access to the prover. This is in contrast to traditional definitions of rewinding knowledge soundness, which consider two probability experiments, one for the prover (considering the prover's success probability) and one for the extractor (considering the extractor's success probability).

Definition 8.1.4 facilitates achieving *adaptive* rewinding knowledge soundness for corresponding non-interactive arguments (according to Definition 5.1.3, and its equivalent Definition 5.1.7). We use the same “single-experiment format” as Definition 8.1.4 for all other notions of rewinding knowledge soundness in this book: Definition 13.1.5 for IPs and Definition 23.1.5 for IOPs; and also for all corresponding state-restoration strengthenings in Definitions 12.1.5, 13.2.5 and 23.2.5.

We show how Definition 8.1.4 implies a common definition of rewinding knowledge soundness (there are others in the literature, also implied similarly).

There exists an extractor \mathbf{E}'_{SP} such that for every adversary $\tilde{\mathbf{P}}_{\text{SP}}$:

$$\begin{aligned} & \Pr \left[(\mathbf{x}, \mathbf{w}) \in \mathcal{R} \mid \mathbf{w} \leftarrow \mathbf{E}'_{\text{SP}}(\mathbf{x}, \tilde{\mathbf{P}}_{\text{SP}}) \right] \\ & \geq \Pr \left[\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \mid \begin{array}{l} (\alpha_1, \mathsf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}} \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathsf{aux}, \rho) \end{array} \right] - \kappa_{\text{SP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{SP}}}(\mathbf{x})) . \end{aligned}$$

We show that if $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ satisfies Definition 8.1.4 with an extractor \mathbf{E}_{SP} then $\text{SP}' = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}}')$ satisfies the above definition with the extractor \mathbf{E}'_{SP} defined below.

$\mathbf{E}'_{\text{SP}}(\mathbf{x}, \tilde{\mathbf{P}}_{\text{SP}})$:

1. Compute $(\alpha_1, \mathsf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}$.
2. Sample $\rho \leftarrow \{0, 1\}^r$.
3. Compute $\alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathsf{aux}, \rho)$.
4. Compute $\mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}})$.
5. Output \mathbf{w} .

We get that

$$\begin{aligned} & \Pr \left[(\mathbf{x}, \mathbf{w}) \in \mathcal{R} \mid \mathbf{w} \leftarrow \mathbf{E}'_{\text{SP}}(\mathbf{x}, \tilde{\mathbf{P}}_{\text{SP}}) \right] \\ & = \Pr \left[(\mathbf{x}, \mathbf{w}) \in \mathcal{R} \mid \begin{array}{l} (\alpha_1, \mathsf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}} \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathsf{aux}, \rho) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}) \end{array} \right] \quad (\text{by definition of } \mathbf{E}'_{\text{SP}}) \\ & \geq \Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \in \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \mid \begin{array}{l} (\alpha_1, \mathsf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}} \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathsf{aux}, \rho) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}) \end{array} \right] \\ & = \Pr \left[\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \mid \begin{array}{l} (\alpha_1, \mathsf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}} \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathsf{aux}, \rho) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}) \end{array} \right] \end{aligned}$$

$$\begin{aligned}
& - \Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} (\alpha_1, \mathbf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}} \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathbf{aux}, \rho) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}) \end{array} \right] \\
& \geq \Pr \left[\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \middle| \begin{array}{l} (\alpha_1, \mathbf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}} \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathbf{aux}, \rho) \end{array} \right] - \kappa_{\text{SP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{SP}}}(\mathbf{x})) .
\end{aligned}$$

9 Warmup: non-adaptive security

We describe how to transform any sigma protocol (SP) into a non-interactive argument that has *non-adaptive* security, via a simple warmup construction.

Organization In Section 9.1 we describe the construction. In Section 9.2 we show a lower bound on the non-adaptive soundness error. In Section 9.3 we show non-adaptive soundness. In Section 9.4 we show non-adaptive knowledge soundness. In Section 9.5 we show non-adaptive zero knowledge. In Section 9.6 we explain why the construction does *not* have adaptive security.

9.1 Construction

The main idea is that the argument prover can use the random oracle to produce the SP verifier challenge in a way that is independent of the SP commitment, and subsequently send an argument string that contains the SP commitment and SP response.

In more detail, the SP verifier participates in the interaction only by sending its challenge ρ after receiving the SP prover commitment α_1 . Hence, the argument prover can query the random oracle f at α_1 to obtain $\rho := f(\alpha_1)$ as a challenge. By definition of the random oracle, the answer $f(\alpha_1)$ is uniformly random. The argument prover can then compute its response α_2 based on $f(\alpha_1)$, and send the argument string $\pi := (\alpha_1, \alpha_2)$. The argument verifier can then run the SP verifier decision predicate on the commitment α_1 , challenge $f(\alpha_1)$, and response α_2 .

The output size of the random oracle and the size of an SP challenge must equal. An SP challenge ρ consists of r bits, so the random oracle f is sampled from $\mathcal{U}(r)$.

Construction 9.1.1. Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP. Consider the non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ constructed as follows. The argument prover \mathcal{P} receives as input an instance \mathbf{x} and witness \mathbf{w} , and the argument verifier \mathcal{V} receives as input the instance \mathbf{x} and an argument string π . Both receive query access to a random oracle $f \in \mathcal{U}(r)$; this implies that the oracle configuration cnf is defined as $\text{cnf}(\lambda, n) := r$ (see Definition 7.1.1).

- $\mathcal{P}^f(\mathbf{x}, \mathbf{w})$:
 1. Run the SP prover to obtain the first message and auxiliary state: $(\alpha_1, \mathbf{aux}) \leftarrow \mathbf{P}_{\text{SP}}(\mathbf{x}, \mathbf{w})$.
 2. Derive the SP verifier challenge as $\rho := f(\alpha_1)$.
 3. Run the SP prover to obtain the second message: $\alpha_2 \leftarrow \mathbf{P}_{\text{SP}}(\mathbf{aux}, \rho)$.
 4. Output the argument string $\pi := (\alpha_1, \alpha_2)$.
- $\mathcal{V}^f(\mathbf{x}, \pi)$:
 1. Parse the argument string π as a commitment-response pair (α_1, α_2) .
 2. Derive the SP verifier challenge ρ as in Item 2 of the argument prover \mathcal{P} .
 3. Check that the SP verifier accepts: $\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1$.

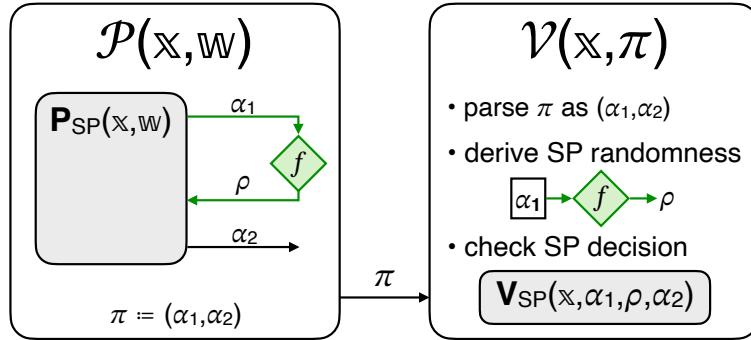


Figure 9.1: Diagram of Construction 9.1.1.

9.2 Lower bound on the non-adaptive soundness error

We describe a malicious argument prover that “attacks” Construction 9.1.1, thereby establishing a lower-bound on the non-adaptive soundness error. This provides the basic intuition for what non-adaptive soundness error we should expect of this construction.

Informally, while a malicious SP prover has only a *single* chance to convince the SP verifier, a malicious argument prover for Construction 9.1.1 has *multiple* chances to convince the SP verifier that underlies the argument verifier. Intuitively, this means that the soundness error of the non-interactive argument is *not* equal to the soundness error of the SP (not even up to a small additive error). Instead, in general, the transformation incurs a multiplicative factor in soundness error that is proportional to the number of chances to convince the underlying SP verifier.

In more detail, we formalize the above intuition via an attack that works when transforming SPs that satisfy a certain property.¹ Consider an SP for a given relation \mathcal{R} that satisfies the following mild requirement (satisfied by natural SPs):

$$\exists \mathbf{x} \notin \mathcal{L}(\mathcal{R}), \forall \alpha_1 \in \{0, 1\}^{\text{PV}_1} \Pr \left[\begin{array}{c} \exists \alpha_2 \in \{0, 1\}^{\text{PV}_2} \text{ s.t.} \\ \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \rho \leftarrow \{0, 1\}^r \right] = \epsilon_{\text{SP}}(\mathbf{x}). \quad (9.1)$$

For a given query bound $t \in \mathbb{N}$, consider the t -query argument prover $\tilde{\mathcal{P}}$ defined as follows:

1. For up to t times:
 - a) Choose any SP prover first message $\alpha_1 \in \{0, 1\}^{\text{PV}_1}$ that was not previously chosen.
 - b) Query the random oracle to compute the SP verifier challenge $\rho := f(\alpha_1)$.
 - c) If there exists $\alpha_2 \in \{0, 1\}^{\text{PV}_2}$ such that $\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1$ then output $\pi := (\alpha_1, \alpha_2)$.
2. Output $\pi := \perp$.

Lemma 9.2.1. *Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP for a relation \mathcal{R} that satisfies Equation 9.1. Then $\text{NARG} = (\mathcal{P}, \mathcal{V})$ defined in Construction 9.1.1 is a non-interactive argument for \mathcal{R} with non-adaptive soundness error ϵ_{ARG} (see Definition 7.1.3) such that there exists $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ such that for every security parameter $\lambda \in \mathbb{N}$ and query bound $t \in \mathbb{N}$,*

$$\epsilon_{\text{ARG}}(\lambda, t, \mathbf{x}) \geq t \cdot \epsilon_{\text{SP}}(\mathbf{x}) - \binom{t}{2} \cdot \epsilon_{\text{SP}}(\mathbf{x})^2.$$

¹We cannot expect an attack that works for *every* SP. For example, if the underlying SP has soundness error 0 then so does the non-interactive argument (because, in the soundness case, there is no way to convince the underlying SP verifier regardless of the number of attempts).

In particular, if $t \cdot \epsilon_{\text{SP}}(\mathbf{x}) \leq 1$ then the lower bound is at least $\frac{t}{2} \cdot \epsilon_{\text{SP}}(\mathbf{x})$.

Proof. Fix an instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ that satisfies Equation 9.1. We use the inclusion-exclusion principle (Lemma 1.2.3) to lower bound the winning probability of $\tilde{\mathcal{P}}$. For every $i \in [t]$, let $\alpha_{1,i} \in \{0,1\}^{\mathbf{PV}_1}$ be the i -th query of $\tilde{\mathcal{P}}$ and let X_i be the indicator random variable for the event that there exists $\alpha_2 \in \{0,1\}^{\mathbf{PV}_2}$ such that $\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_{1,i}, f(\alpha_{1,i}), \alpha_2) = 1$. The random variables $(X_i)_{i \in [t]}$ are independent. From Equation 9.1, we know that $\Pr[X_i = 1] = \epsilon_{\text{SP}}(\mathbf{x})$. Therefore we get that

$$\begin{aligned} & \Pr \left[\mathcal{V}^f(\mathbf{x}, \pi) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\mathbf{r}) \\ \pi \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ &= \Pr \left[\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, f(\alpha_1), \alpha_2) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\mathbf{r}) \\ (\alpha_1, \alpha_2) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \quad (\text{by definition of } \mathcal{V}) \\ &= \Pr[\exists i \in [t] : X_i = 1] \quad (\text{by definition of } \tilde{\mathcal{P}}) \\ &\geq \sum_{i \in [t]} \Pr[X_i = 1] - \sum_{\substack{i,j \in [t] \\ \text{with } i \neq j}} \Pr[X_i \cdot X_j = 1] \quad (\text{by Lemma 1.2.3}) \\ &= \sum_{i \in [t]} \Pr[X_i = 1] - \sum_{\substack{i,j \in [t] \\ \text{with } i \neq j}} \Pr[X_i = 1] \cdot \Pr[X_j = 1] \quad (\text{by independence}) \\ &= t \cdot \epsilon_{\text{SP}}(\mathbf{x}) - \binom{t}{2} \cdot \epsilon_{\text{SP}}(\mathbf{x})^2. \end{aligned}$$

□

We conclude that Construction 9.1.1 is *not* sound if the underlying SP does not have small enough soundness error. Next we wish to understand the converse: is Construction 9.1.1 sound if the underlying SP has small enough soundness error?

Remark 9.2.2. The malicious argument prover $\tilde{\mathcal{P}}$ described above attempts to convince the SP verifier with t queries, and if it fails it outputs the empty argument string $\pi := \perp$. A minor improvement is to *not* output the empty argument string: if all t queries fail then $\tilde{\mathcal{P}}$ outputs $\pi := (\alpha_1, \alpha_2)$ for any first message α_1 not queried so far and the best response α_2 for α_1 . For specific SPs, this is tantamount to $\tilde{\mathcal{P}}$ having an additional query due to the chance that a tuple (α_1, α_2) sampled this way convinces the SP verifier. This explains why, in the soundness analysis in Section 9.3, we obtain the multiplicative factor $t + 1$ in the soundness error, rather than t .

9.3 Non-adaptive soundness

We prove that Construction 9.1.1 has non-adaptive soundness error $\epsilon_{\text{ARG}}(\lambda, t, \mathbf{x}) \leq (t+1) \cdot \epsilon_{\text{SP}}(\mathbf{x})$. This shows that Construction 9.1.1 is non-adaptively sound if the underlying SP has a small enough soundness error and that the attack in Section 9.2 is essentially optimal. Later in Section 9.6.1 we explain why Construction 9.1.1 does not achieve adaptive soundness.

We provide two security analyses.

- In Section 9.3.1 we provide an analysis that directly upper bounds the non-adaptive soundness error of the non-interactive argument by invoking the upper bound on the soundness error of the underlying SP.

- In Section 9.3.2 we provide an alternative analysis: we prove that every malicious argument prover can be *efficiently* transformed into a corresponding malicious SP prover with a multiplicative loss of $t + 1$ in convincing probability.

The alternative analysis shows the same upper bound for the non-adaptive soundness error but is a stronger statement: it provides an efficient method to translate strategies for convincing the argument verifier into strategies for convincing the SP verifier.

In the rest of this book, we always provide *efficient security analyses* (efficient transformations of adversaries, analogous to that in Section 9.3.2) because an efficient translation of adversarial strategies is useful in many settings. We give two examples.

- An efficient translation of adversarial strategies is typically one of the steps to establishing knowledge soundness. In Section 9.4 we see how the efficient translation of adversarial strategies plays a role towards the knowledge soundness of Construction 9.1.1.
- If the underlying SP were to satisfy only computational soundness due to the use of cryptographic assumptions (i.e., the SP is in fact a 3-message interactive argument), proving soundness of Construction 9.1.1 would require constructing an efficient adversary for the interactive argument from the given adversary of the non-interactive argument.

9.3.1 An inefficient security reduction

Lemma 9.3.1. *Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP for a relation \mathcal{R} with soundness error ϵ_{SP} . Then $\text{NARG} = (\mathcal{P}, \mathcal{V})$ defined in Construction 9.1.1 is a non-interactive argument for \mathcal{R} with non-adaptive soundness error ϵ_{ARG} (see Definition 7.1.4) such that for every instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$,*

$$\epsilon_{\text{ARG}}(\lambda, t, \mathbf{x}) \leq (t + 1) \cdot \epsilon_{\text{SP}}(\mathbf{x}).$$

Proof. Let $\tilde{\mathcal{P}}$ be a t -query malicious argument prover that outputs an argument string $\pi = (\alpha_1, \alpha_2)$. We upper bound the probability that $\mathcal{V}^f(\mathbf{x}, \pi) = 1$.

- *Case 1: α_1 is a query in the trace.* We argue that

$$\Pr \left[\begin{array}{c} \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, f(\alpha_1), \alpha_2) = 1 \\ \wedge (\alpha_1, f(\alpha_1)) \in \text{tr} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\alpha_1, \alpha_2) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \end{array} \right] \leq t \cdot \epsilon_{\text{SP}}(\mathbf{x}).$$

We upper bound the probability that there exists a query-answer pair $(\alpha_1, \rho) \in \text{tr}$ and a response $\alpha_2 \in \{0, 1\}^{\text{PV}_2}$ such that $\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1$. The soundness of the SP (see Equation 8.1) gives that for every (distinct) query $\alpha_1 \in \{0, 1\}^{\text{PV}_1}$ it holds that

$$\Pr \left[\begin{array}{c} \exists \alpha_2 \in \{0, 1\}^{\text{PV}_2} \text{ s.t.} \\ \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \rho \in \{0, 1\}^r \right] \leq \epsilon_{\text{SP}}(\mathbf{x}).$$

Since $\tilde{\mathcal{P}}$ performs at most t queries, we get that

$$\Pr \left[\begin{array}{c} \mathbf{V}_{\text{SP}}(\mathbf{x}, f(\alpha_1), \rho, \alpha_2) = 1 \\ \wedge (\alpha_1, f(\alpha_1)) \in \text{tr} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\alpha_1, \alpha_2) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \end{array} \right] \leq t \cdot \epsilon_{\text{SP}}(\mathbf{x}).$$

- *Case 2: α_1 is not a query in the trace.* We argue that

$$\Pr \left[\begin{array}{l} \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, f(\alpha_1), \alpha_2) = 1 \\ \wedge (\alpha_1, f(\alpha_1)) \notin \text{tr} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\alpha_1, \alpha_2) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \end{array} \right] \leq \epsilon_{\text{SP}}(\mathbf{x}).$$

Suppose that the malicious argument prover $\tilde{\mathcal{P}}$ outputs α_1 that is not a query in the trace tr . Then, the response $f(\alpha_1)$ is random, and thus by the soundness of the SP, the probability that the SP verifier accepts in this case is at most $\epsilon_{\text{SP}}(\mathbf{x})$.

We conclude that

$$\begin{aligned} & \Pr \left[\mathcal{V}^f(\mathbf{x}, \pi) = 1 \middle| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ \pi \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ &= \Pr \left[\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, f(\alpha_1), \alpha_2) = 1 \middle| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\alpha_1, \alpha_2) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \end{array} \right] \\ &\leq t \cdot \epsilon_{\text{SP}}(\mathbf{x}) + \epsilon_{\text{SP}}(\mathbf{x}) \\ &= (t+1) \cdot \epsilon_{\text{SP}}(\mathbf{x}). \end{aligned}$$

□

9.3.2 An efficient security reduction

Lemma 9.3.2. *Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP for a relation \mathcal{R} , and let $\text{NARG} = (\mathcal{P}, \mathcal{V})$ be the non-interactive argument defined in Construction 9.1.1. There exists an SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ such that for every instance \mathbf{x} , query bound $t \in \mathbb{N}$, and malicious t -query argument prover $\tilde{\mathcal{P}}$,*

$$\begin{aligned} & \Pr \left[\mathcal{V}^f(\mathbf{x}, \pi) = 1 \middle| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ \pi \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ &\leq (t+1) \cdot \Pr \left[\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \middle| \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \end{array} \right]. \end{aligned}$$

Moreover, if the running time of $\tilde{\mathcal{P}}$ is $\mathbf{t}_{\mathcal{P}}$ then the running time of $\tilde{\mathbf{P}}_{\text{SP}}$ is $\mathbf{t}_{\mathcal{P}} + O(r \cdot t)$.

Construction 9.3.3. The SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ is parametrized by the query bound t and receives an argument prover $\tilde{\mathcal{P}}$ as a black box, and works as follows.

- First SP prover message: $\tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \rightarrow (\alpha_1, \text{aux})$.
 1. Lazily sample an oracle $\dot{f} \leftarrow \mathcal{U}(r)$.
 2. Sample an index $i \in [t+1]$ at random.
 3. Simulate $\tilde{\mathcal{P}}^{\dot{f}}$ up to before the i -th unique query x to f :
 - a) If $\tilde{\mathcal{P}}$ already stopped with output (α'_1, α'_2) , then set $\alpha_1 := \alpha'_1$.
 - b) Else if the query x can be parsed as $\alpha_{1,i}$, then set $\alpha_1 := \alpha_{1,i}$.
 - c) Else set $\alpha_1 := \perp$.
 4. Let $\text{aux}_{\tilde{\mathcal{P}}}$ be the state of $\tilde{\mathcal{P}}$.
 5. Output the first SP prover message α_1 and the auxiliary information $\text{aux} := (\text{aux}_{\tilde{\mathcal{P}}}, \alpha_1, \dot{f})$.
- Second SP prover message: $\tilde{\mathbf{P}}_{\text{SP}}(\text{aux} = (\text{aux}_{\tilde{\mathcal{P}}}, \alpha_1, \dot{f}), \rho \in \{0, 1\}^r) \rightarrow \alpha_2$.

1. Set $\mu := \{(\alpha_1, \rho)\}$.
2. Use $\text{aux}_{\tilde{\mathcal{P}}}$ to continue the simulation of $\tilde{\mathcal{P}}^{f[\mu]}$ until it outputs (α'_1, α'_2) .
3. If $\alpha'_1 \neq \alpha_1$ set $\alpha_2 := \perp$; otherwise set $\alpha_2 := \alpha'_2$.
4. Output the second SP prover message α_2 .

Note that the running of $\tilde{\mathbf{P}}_{\text{SP}}$ is $\mathbf{t}_{\mathcal{P}} + O(r \cdot t)$ since each of the t queries takes time at most $O(r)$ to simulate.

Proof of Lemma 9.3.2. The SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ simulates $\tilde{\mathcal{P}}$ with a random function (sampled in a lazy way): $\tilde{\mathbf{P}}_{\text{SP}}$ runs $\tilde{\mathcal{P}}$ with the oracle $f[\mu]$, where $\mu = \{(\alpha_1, \rho)\}$ programs f with the random answer ρ for the query α_1 . Moreover, the choice of $i \in [t+1]$ is independent of the execution of $\tilde{\mathcal{P}}$. If $\alpha'_1 = \alpha_{1,j}$ for some $j \in [t]$ then $\alpha'_1 = \alpha_1$ assuming $\tilde{\mathbf{P}}_{\text{SP}}$ sampled $i = j$. Otherwise, if α'_1 is not a query in the query-answer trace, then $\alpha'_1 = \alpha_1$ if $\tilde{\mathbf{P}}_{\text{SP}}$ sampled $i = t+1$. We deduce that the probability that $\alpha'_1 = \alpha_1$, and thus that $\alpha_2 \neq \perp$, is at least $\frac{1}{t+1}$. Therefore the following two distributions are equivalent:

$$\left\{ (\alpha_1, \rho, \alpha_2) \middle| \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \\ \text{conditioned on} \\ \alpha_2 \neq \perp \end{array} \right\} \equiv \left\{ (\alpha_1, \rho, \alpha_2) \middle| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\alpha_1, \alpha_2) \leftarrow \tilde{\mathcal{P}}^f \\ \rho := f(\alpha_1) \end{array} \right\}. \quad (9.2)$$

Moreover, we can lower bound the probability that $\alpha_2 \neq \perp$ as follows:

$$\Pr \left[\alpha_2 \neq \perp \middle| \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \end{array} \right] \geq \frac{1}{t+1}. \quad (9.3)$$

Using the above, we conclude that

$$\begin{aligned} & \Pr \left[\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \middle| \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \end{array} \right] \\ & \geq \Pr \left[\alpha_2 \neq \perp \middle| \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \end{array} \right] \cdot \Pr \left[\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \middle| \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \neq \perp \\ \text{conditioned on} \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \end{array} \right] \\ & \geq \frac{1}{t+1} \cdot \Pr \left[\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, f(\alpha_1), \alpha_2) = 1 \middle| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\alpha_1, \alpha_2) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ & = \frac{1}{t+1} \cdot \Pr \left[\mathcal{V}^f(\mathbf{x}, \pi) = 1 \middle| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ \pi \leftarrow \tilde{\mathcal{P}}^f \end{array} \right]. \end{aligned}$$

□

9.4 Non-adaptive knowledge soundness

We prove that Construction 9.1.1 satisfies the weak notion of *non-adaptive* knowledge soundness.

Suppose that $\mathbf{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ has rewinding knowledge soundness error κ_{SP} with extraction time \mathbf{et}_{SP} relative to an extractor \mathbf{E}_{SP} (see Definition 8.1.4). Below we show that $\mathbf{NARG} = (\mathcal{P}, \mathcal{V})$

in Construction 9.1.1 satisfies non-adaptive rewinding knowledge soundness (the non-adaptive restriction of Definition 7.1.7) with knowledge soundness error κ_{ARG} with extraction time \mathbf{et}_{ARG} relative to an extractor \mathcal{E} (that we construct) such that

- $\kappa_{\text{ARG}}(\lambda, t, \mathbf{x}, \delta_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x})) \leq (t+1) \cdot \kappa_{\text{SP}}\left(\mathbf{x}, \delta'_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x})\right)$, and
- $\mathbf{et}_{\text{ARG}}(\lambda, t, \mathbf{x}, \delta_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x}), \tau_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x})) \leq \mathbf{et}_{\text{SP}}\left(\mathbf{x}, \delta'_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x}), \tau'_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x})\right)$.

Above, $\delta'_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x}) := 1 - \frac{1-\delta_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x})}{t+1}$ and $\tau'_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x}) := \tau_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x}) + O(r \cdot t)$.

Moreover, if \mathbf{E}_{SP} is a straightline extractor (\mathbf{E}_{SP} receives as input only the instance \mathbf{x} and SP prover messages (α_1, α_2)) then κ_{SP} and \mathbf{et}_{SP} do not depend on the prover failure probability (or prover running time); therefore, \mathcal{E} is also a straightline extractor, so the knowledge soundness error and extraction time satisfy simplified upper bounds:

$$\kappa_{\text{ARG}}(\lambda, t, \mathbf{x}) \leq (t+1) \cdot \kappa_{\text{SP}}(\mathbf{x}) \quad \text{and} \quad \mathbf{et}_{\text{ARG}}(\lambda, t, \mathbf{x}) \leq \mathbf{et}_{\text{SP}}(\mathbf{x}).$$

The following lemma directly implies the above bounds. The lemma states that the non-adaptive knowledge soundness error of the non-interactive argument for a given argument prover $\tilde{\mathcal{P}}$ is upper bounded by $t+1$ times the knowledge soundness error of the underlying SP for the corresponding SP prover $\tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}})$ from Construction 9.3.3.

Lemma 9.4.1. *Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP for a relation \mathcal{R} , and let $\text{NARG} = (\mathcal{P}, \mathcal{V})$ be the non-interactive argument defined in Construction 9.1.1. There exists an argument extractor \mathcal{E} such that for every instance \mathbf{x} , query bound $t \in \mathbb{N}$, and malicious t -query argument prover $\tilde{\mathcal{P}}$,*

$$\begin{aligned} & \Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ \pi \xleftarrow{\text{tr}} \tilde{\mathcal{P}}f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}^f(\mathbf{x}, \pi) \\ \mathbf{w} \leftarrow \mathcal{E}(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}}) \end{array} \right] \\ & \leq (t+1) \cdot \Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}})) \end{array} \right]. \end{aligned}$$

Moreover, the running time of \mathcal{E} is $\mathbf{et}_{\text{SP}}\left(\mathbf{x}, 1 - \frac{1-\delta_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x})}{t+1}, \tau_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x}) + O(r \cdot t)\right)$.

Construction 9.4.2. The argument extractor \mathcal{E} receives as input an instance \mathbf{x} , argument string π , query-answer trace tr of the argument prover, query-answer trace tr_v of the argument verifier, and black-box access to the argument prover $\tilde{\mathcal{P}}$, and works as follows.

$\mathcal{E}(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}})$:

1. Parse the argument string π as a tuple (α_1, α_2) .
2. Parse the trace tr_v as a single query-answer pair $((\alpha_1, \rho))$.
3. Let $\tilde{\mathbf{P}}_{\text{SP}}$ be the SP prover in Construction 9.3.3 obtained from $\tilde{\mathcal{P}}$.
4. Compute the witness: $\mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}))$.
5. Output \mathbf{w} .

Proof. We upper bound the knowledge soundness error of the non-interactive argument. Observe that \mathcal{E} relies on tr_v to set ρ equal to $f(\alpha_1)$. Hence we can write

$$\begin{aligned}
& \Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \pi \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}^f(\mathbf{x}, \pi) \\ \mathbf{w} \leftarrow \mathcal{E}(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}}) \end{array} \right] \\
&= \Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\alpha_1, \alpha_2) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ \rho := f(\alpha_1) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}})) \end{array} \right] \quad (\text{by definition of } \mathcal{E}) \\
&= \Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \\ \text{conditioned on} \\ \alpha_2 \neq \perp \end{array} \middle| \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}) \end{array} \right] \quad (\text{by Equation 9.2}) \\
&= \Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \\ \wedge \alpha_2 \neq \perp \end{array} \middle| \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}) \end{array} \right] \\
&\quad / \Pr \left[\alpha_2 \neq \perp \middle| \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \end{array} \right] \\
&\leq (t+1) \cdot \Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}) \end{array} \right].
\end{aligned}$$

The last inequality comes from Equation 9.3 and the fact that $\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1$ implies that $\alpha_2 \neq \perp$.

Next, we discuss the running time of \mathcal{E} . This depends on the running time of \mathbf{E}_{SP} , which in turn depends on the failure probability and running time of $\tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}})$. By Lemma 9.3.2, the failure probability of $\tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}})$ is $\delta'_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x}) := 1 - \frac{1 - \delta_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x})}{t+1}$, and the running time of $\tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}})$ is $\tau'_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x}) := \tau_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x}) + O(r \cdot t)$. Therefore, the running time of the SP extractor \mathbf{E}_{SP} , as invoked by \mathcal{E} , is $\text{et}_{\text{SP}}(\mathbf{x}, \delta'_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x}), \tau'_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x}))$. All other operations in \mathcal{E} are merely syntactic. \square

9.5 Non-adaptive zero knowledge

We prove that Construction 9.1.1 satisfies the weak notion of non-adaptive zero knowledge in Definition 5.2.1. Later in Section 9.6.2 we explain why Construction 9.1.1 does *not* satisfy the stronger notion of adaptive zero knowledge in Definition 7.1.8.

Lemma 9.5.1. *Let SP be an SP for a relation \mathcal{R} with honest-verifier zero-knowledge error z_{SP} . For every security parameter $\lambda \in \mathbb{N}$, Construction 9.1.1 is a non-interactive argument for \mathcal{R}*

with non-adaptive zero-knowledge error z_{ARG} (see Definition 5.2.1) such that

$$z_{\text{ARG}}(\lambda, t, \mathbf{x}) \leq z_{\text{SP}}(\mathbf{x}).$$

Construction 9.5.2. The simulator is an algorithm $\mathcal{S}^f(\mathbf{x})$ that works as follows. Below we denote by \mathbf{S}_{SP} the honest-verifier zero-knowledge simulator of the SP (see Definition 8.1.6).

- $\mathcal{S}^f(\mathbf{x})$:

1. Sample a simulated view of the SP verifier: $(\mathbf{x}, \alpha_1, \rho, \alpha_2) \leftarrow \mathbf{S}_{\text{SP}}(\mathbf{x})$.
2. Set the argument string $\pi := (\alpha_1, \alpha_2)$.
3. Set the list of query-answer pairs $\mu := \{(\alpha_1, \rho)\}$.
4. Output (π, μ) .

Note that \mathcal{S} programs the oracle f at one point via the output μ (and does not query f).

Proof. Fix an algorithm \mathcal{A} and an instance-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$. (The number of queries of \mathcal{A} to the random oracle does not matter in this analysis.) We wish to upper bound the statistical distance between the following two distributions:

$$\mathcal{D}_{\text{real}} := \left\{ \text{out} \mid \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ \pi \leftarrow \mathcal{P}^f(\mathbf{x}, \mathbf{w}) \\ \text{out} \leftarrow \mathcal{A}^f(\pi) \end{array} \right\} \quad \text{and} \quad \mathcal{D}_{\text{sim}} := \left\{ \text{out} \mid \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\pi, \mu) \leftarrow \mathcal{S}^f(\mathbf{x}) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\pi) \end{array} \right\}.$$

By construction of the argument prover \mathcal{P} and the simulator \mathcal{S} , the above two distributions correspond to the following two distributions:

$$\left\{ \text{out} \mid \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\alpha_1, \text{aux}) \leftarrow \mathbf{P}_{\text{SP}}(\mathbf{x}, \mathbf{w}) \\ \rho := f(\alpha_1) \\ \alpha_2 \leftarrow \mathbf{P}_{\text{SP}}(\text{aux}, \rho) \\ \pi := (\alpha_1, \alpha_2) \\ \text{out} \leftarrow \mathcal{A}^f(\pi) \end{array} \right\} \quad \text{and} \quad \left\{ \text{out} \mid \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \alpha_1, \rho, \alpha_2) \leftarrow \mathbf{S}_{\text{SP}}(\mathbf{x}) \\ \pi := (\alpha_1, \alpha_2) \\ \mu := \{(\alpha_1, \rho)\} \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\pi) \end{array} \right\}.$$

In both cases, the output of \mathcal{A} is a function of $\pi = (\alpha_1, \alpha_2)$ and of the answers to \mathcal{A} 's queries to the oracle. In fact, in this analysis we can afford for \mathcal{A} to query the oracle everywhere, and it suffices to upper bound the statistical distance between the following two distributions:

$$\left\{ (\alpha_1, \rho, \alpha_2, f) \mid \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\alpha_1, \text{aux}) \leftarrow \mathbf{P}_{\text{SP}}(\mathbf{x}, \mathbf{w}) \\ \rho := f(\alpha_1) \\ \alpha_2 \leftarrow \mathbf{P}_{\text{SP}}(\text{aux}, \rho) \end{array} \right\} \quad \text{and} \quad \left\{ (\alpha_1, \rho, \alpha_2, f[\mu]) \mid \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \alpha_1, \rho, \alpha_2) \leftarrow \mathbf{S}_{\text{SP}}(\mathbf{x}) \\ \mu := \{(\alpha_1, \rho)\} \end{array} \right\}.$$

In the left distribution the tuple $(\alpha_1, \rho, \alpha_2)$ with $\rho := f(\alpha_1)$ is distributed as in a real SP view (ρ is random in $\{0, 1\}^r$); and in the right distribution the tuple $(\alpha_1, \rho, \alpha_2)$ is sampled by the SP simulator and the oracle f is programmed to answer ρ for the query α_1 . The statistical distance between these two tuples is upper bounded by the honest-verifier zero-knowledge error $z_{\text{SP}}(\mathbf{x})$.

Moreover, in both distributions, the answers to all queries other than α_1 are uniformly and independently random of $(\alpha_1, \rho, \alpha_2)$, and hence do not contribute any additional statistical distance.

In sum, the only difference between the two distributions comes from whether the tuple $(\alpha_1, \rho, \alpha_2)$ is a real SP view or a simulated SP view, and is thus upper bounded by $z_{\text{SP}}(\mathbf{x})$. \square

9.6 Lack of adaptive security

Construction 9.1.1 does not satisfy *adaptive* notions of security. We explain why, and outline modifications of the construction that enable achieving adaptive security (as we will prove later).

9.6.1 Lack of adaptive soundness

Consider the SP for the empty relation (no instance is in the corresponding language) where the SP verifier accepts if and only if the SP verifier challenge equals the instance (that is, $\mathbf{x} = \rho$). For every \mathbf{x} the probability that the SP verifier accepts is at most $\frac{1}{2^r}$ because this is an upper bound on the probability that the SP verifier random challenge $\rho \in \{0, 1\}^r$ equals the instance \mathbf{x} .

We consider the non-interactive argument obtained by applying Construction 9.1.1 to this SP. The discussion in Section 9.3 tells us that, for every instance \mathbf{x} , any t -query malicious argument prover makes the argument verifier accept \mathbf{x} with probability at most $t + 1$ times $\frac{1}{2^r}$.

On the other hand, a malicious argument prover can choose an instance \mathbf{x} based on the random oracle to convince the argument verifier: set $\mathbf{x} := f(\alpha_1)$, because this is how in Construction 9.1.1 the SP verifier challenge ρ is derived. Hence this malicious argument prover convinces the argument verifier, with probability 1 over the choice of random oracle, to accept an adaptively chosen instance that is not in the language.

We deduce that Construction 9.1.1 does *not* have adaptive soundness.

Solution A modification to Construction 9.1.1 that should prevent the above attack is to include the instance \mathbf{x} in the query to the random oracle for deriving the SP verifier challenge ρ . That is, we modify the construction so that ρ is derived as follows:

$$\rho := f(\mathbf{x}, \alpha_1).$$

Intuitively, this modification ensures that the instance \mathbf{x} cannot be chosen after deriving the SP verifier challenge ρ . In Section 10.1 we describe the modified construction, and in Section 10.2 we prove that it satisfies the notion of adaptive soundness (Definition 7.1.4): we prove that the adaptive soundness error of the (so modified) non-interactive argument is at most $t + 1$ times the SP soundness error. (This upper bound on the soundness error is essentially tight due to the lower bound on the soundness error in Section 9.2, which carries over to the modified construction because the “resampling attack” is unaffected by the modification.)

9.6.2 Lack of adaptive zero knowledge

The discussion in Section 9.5 tells us that Construction 9.1.1 satisfies the weak notion of one-time *non-adaptive* zero knowledge (Definition 5.2.1), if the underlying SP is honest-verifier zero-knowledge. Below we explain why the construction does not straightforwardly satisfy the stronger notion of *adaptive* zero knowledge (Definition 7.1.8). This is regardless of whether the instance \mathbf{x} is included in the query to achieve adaptive soundness as discussed in Section 9.6.1.

Consider the simulator \mathcal{S} in Construction 9.5.2: given as input an instance \mathbf{x} , the simulator \mathcal{S} samples an SP verifier view $(\mathbf{x}, \alpha_1, \rho, \alpha_2) \leftarrow \mathbf{S}_{\text{SP}}(\mathbf{x})$ and then programs the random oracle at the query α_1 to answer ρ . Suppose that the SP for the given relation \mathcal{R} is “trivial”: $\alpha_1 = 0$, ρ is a random challenge, and $\alpha_2 = 0$. This could happen, e.g., if the relation \mathcal{R} is efficiently decidable

(as the SP verifier can efficiently decide the relation without the help of the SP prover). Now consider the adversary \mathcal{A} for Definition 7.1.8 that is defined as follows:

- \mathcal{A} , given query access to the random oracle f , outputs an arbitrary instance \mathbf{x} and witness \mathbf{w} such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ as well as the auxiliary state $\mathbf{aux} := y$ where $y := f(0)$.
- \mathcal{A} , given query access to the random oracle f (if in the real world) or the programmed random oracle $f[\mu]$ (if in the simulated world) and given input (\mathbf{aux}, π) , outputs **real** if querying the oracle returns y and outputs **sim** otherwise. (In particular, \mathcal{A} ignores the argument string π .)

In the real world \mathcal{A} outputs **real** with probability 1, because \mathcal{A} outputs **real** if and only if the answer to the query $x = 0$ is the same in both phases (before and after programming), which is the case because in the real world \mathcal{A} has access to the same oracle in the two phases.

In the simulated world the oracle in the second phase is programmed. The simulator above programs the random oracle to be a freshly sampled challenge, which is different from y stored in \mathbf{aux} with all but a probability of $\frac{1}{2^r}$.

Overall the statistical distance between the outputs in the two cases is at least $1 - \frac{1}{2^r}$ (that is, the adversary distinguishes almost always between the real world and the simulated world).

The above discussion does not qualify as a counter example because we did not establish that for every simulator there exists a distinguishing adversary. In fact, one could argue that this example could be resolved by having the simulator depend on the relation, and not program the random oracle for trivial examples like the one above.

Nevertheless, the above discussion does show that the simple simulator in Construction 9.5.2 does *not* work. Moreover, it is desirable to have a simple simulator that works for all relations.

Solution A modification to Construction 9.1.1 that, intuitively, should prevent the above problem is to include a random salt τ in the query to the random oracle for deriving the SP verifier challenge. That is, we modify the construction so that ρ is derived as follows:

$$\rho := f(\alpha_1, \tau).$$

Intuitively, this modification ensures that the argument simulator \mathcal{S} programs the location (α_1, τ) , which is unpredictable unless the adversary guesses the random salt τ in the first phase (which is unlikely). In Section 10.1 we describe the modified construction, and in Section 11.2 we prove that it satisfies the notion of adaptive zero-knowledge (Definition 7.1.8).

10 The Fiat–Shamir transformation for SPs

We describe how to transform any sigma protocol (SP) into a non-interactive argument that has adaptive security, addressing the limitations of the warmup construction in Chapter 9.

Organization In Section 10.1 we describe the improved construction, and in Section 10.2 we show that it has adaptive soundness. In Chapter 11 we study additional security definitions for this construction.

10.1 Construction

We add to Construction 9.1.1 the features discussed in Section 9.6 to additionally achieve adaptive security: (i) including the instance \mathbf{x} in the query to the random oracle; (ii) including a random salt τ in the query to the random oracle. Below is the resulting construction.

Construction 10.1.1: Fiat–Shamir transformation for SPs

Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP. Let $\lambda \in \mathbb{N}$ be a security parameter and $s \in \mathbb{N}$ be a privacy parameter.

We define $\text{NARG} := \mathbf{FS}_{\text{SP}}[\text{SP}, \lambda, s]$ to be the non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ constructed as follows. The argument prover \mathcal{P} receives as input an instance \mathbf{x} and witness \mathbf{w} , and the argument verifier \mathcal{V} receives as input the instance \mathbf{x} and an argument string π . Both receive query access to a random oracle $f \in \mathcal{U}(\mathbf{r})$; this implies that the oracle configuration cnf is defined as $\text{cnf}(\lambda, n) := \mathbf{r}$ (see Definition 7.1.1).

- $\mathcal{P}^f(\mathbf{x}, \mathbf{w})$:
 1. Run the SP prover to obtain the first message and auxiliary state: $(\alpha_1, \text{aux}) \leftarrow \mathbf{P}_{\text{SP}}(\mathbf{x}, \mathbf{w})$.
 2. Sample a random salt $\tau \in \{0, 1\}^s$.
 3. Derive the SP verifier challenge as $\rho := f(\mathbf{x}, \alpha_1, \tau)$.
 4. Run the SP prover to obtain the second message: $\alpha_2 \leftarrow \mathbf{P}_{\text{SP}}(\text{aux}, \rho)$.
 5. Output the argument string $\pi := (\alpha_1, \alpha_2, \tau)$.
- $\mathcal{V}^f(\mathbf{x}, \pi)$:
 1. Parse the argument string π as a tuple $(\alpha_1, \alpha_2, \tau)$.
 2. Derive the SP verifier challenge ρ as in Item 3 of the argument prover \mathcal{P} .
 3. Check that the SP verifier accepts: $\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1$.

Efficiency We discuss efficiency properties of the above construction.

- *Argument size.* The argument string π contains the two prover messages and a salt, but not the verifier message. Hence its size equals the prover-to-verifier communication complexity of the SP plus the salt size:

$$\text{len}(\alpha_1) + \text{len}(\alpha_2) + s = \text{pv}_1 + \text{pv}_2 + s.$$

In particular, the transformation *does not compress prover messages*, but rather merely removes the interaction between the prover and verifier.

- *Prover complexity.* The cost of the argument prover \mathcal{P} is essentially the same as that of the underlying SP prover \mathbf{P}_{SP} . The only difference is that the argument prover \mathcal{P} additionally makes 1 query of size $\text{len}(x) + \text{len}(\alpha_1) + s$ to the random oracle.
- *Verifier complexity.* The cost of the argument verifier \mathcal{V} is essentially the same as that of the underlying SP verifier \mathbf{V}_{SP} . The only difference is that the argument verifier \mathcal{V} additionally makes 1 query, the same one as the argument prover \mathcal{P} .

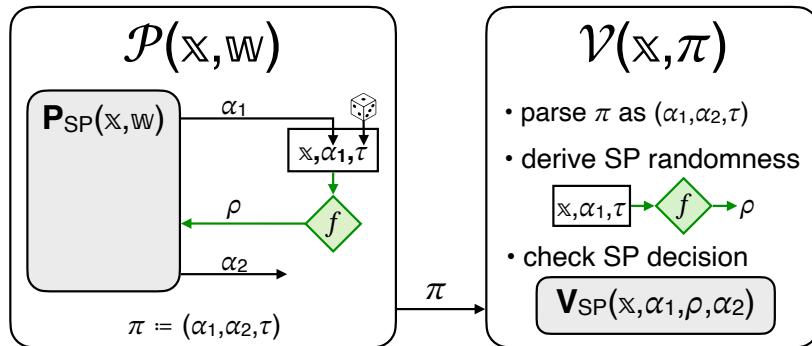


Figure 10.1: Diagram of the Fiat–Shamir transformation for SPs (\mathbf{FS}_{SP} in Construction 10.1.1).

10.2 Adaptive soundness

We prove that the adaptive soundness error of the non-interactive argument in Construction 10.1.1 is at most $t + 1$ times the soundness error of the underlying SP. This directly follows from a lemma that (efficiently) transforms a malicious argument prover into a malicious SP prover, with a loss of at most a factor of $t + 1$ in convincing probability.

Theorem 10.2.1

Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP for a relation \mathcal{R} with soundness error ϵ_{SP} . For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{NARG} := \mathbf{FS}_{\text{SP}}[\text{SP}, \lambda, s]$ in Construction 10.1.1 is a non-interactive argument for \mathcal{R} with adaptive soundness error ϵ_{ARG} (see Definition 7.1.4) such that

$$\epsilon_{\text{ARG}}(\lambda, t, n) \leq (t + 1) \cdot \epsilon_{\text{SP}}(n).$$

Lemma 10.2.2. *There exists an SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ such that for every instance size bound $n \in \mathbb{N}$, query bound $t \in \mathbb{N}$, and malicious t -query argument prover $\tilde{\mathcal{P}}$,*

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}_{\mathbf{x}}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ \leq (t+1) \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \end{array} \right].$$

Moreover, if the running time of $\tilde{\mathcal{P}}$ is $\mathbf{t}_{\mathcal{P}}$ then the running time of $\tilde{\mathbf{P}}_{\text{SP}}$ is $\mathbf{t}_{\mathcal{P}} + O(r \cdot t)$.

Construction 10.2.3. The SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ is parameterized by the query bound t and receives an argument prover $\tilde{\mathcal{P}}$ as a black box, and works as follows.

- First SP prover message: $\tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \rightarrow (\mathbf{x}, \alpha_1, \text{aux})$.
 1. Lazily sample an oracle $\dot{f} \leftarrow \mathcal{U}(\mathbf{r})$.
 2. Sample an index $i \in [t+1]$ at random.
 3. Simulate $\tilde{\mathcal{P}}^{\dot{f}}$ up to before the i -th unique query x to f :
 - a) If $\tilde{\mathcal{P}}$ already stopped with output $(\mathbf{x}', (\alpha'_1, \alpha'_2, \tau'))$, then set $\mathbf{x} := \mathbf{x}'$, $\tau := \tau'$, $\alpha_1 := \alpha'_1$.
 - b) Else if the query x can be parsed as $(\mathbf{x}_i, \alpha_{1,i}, \tau_i)$, then set $\mathbf{x} := \mathbf{x}_i$, $\tau := \tau_i$, $\alpha_1 := \alpha_{1,i}$.
 - c) Else set $\mathbf{x} := \perp$, $\tau := \perp$, $\alpha_1 := \perp$.
 4. Let $\text{aux}_{\tilde{\mathcal{P}}}$ be the state of $\tilde{\mathcal{P}}$.
 5. Output the instance \mathbf{x} , first SP prover message α_1 , and the auxiliary information $\text{aux} := (\dot{f}, \mathbf{x}, \alpha_1, \tau, \text{aux}_{\tilde{\mathcal{P}}})$.
- Second SP prover message: $\tilde{\mathbf{P}}_{\text{SP}}(\text{aux} = (\dot{f}, \mathbf{x}, \alpha_1, \tau, \text{aux}_{\tilde{\mathcal{P}}})), \rho \in \{0, 1\}^r \rightarrow \alpha_2$.
 1. Set $\mu := \{((\mathbf{x}, \alpha_1, \tau), \rho)\}$.
 2. Use $\text{aux}_{\tilde{\mathcal{P}}}$ to continue the simulation of $\tilde{\mathcal{P}}^{\dot{f}[\mu]}$ until it outputs $(\mathbf{x}', (\alpha'_1, \alpha'_2, \tau'))$.
 3. If $\mathbf{x}' \neq \mathbf{x} \vee \alpha'_1 \neq \alpha_1 \vee \tau' \neq \tau$ set $\alpha_2 := \perp$; otherwise set $\alpha_2 := \alpha'_2$.
 4. Output the second SP prover message α_2 .

Proof. The SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ simulates $\tilde{\mathcal{P}}$ with a random function (sampled in a lazy way): $\tilde{\mathbf{P}}_{\text{SP}}$ runs $\tilde{\mathcal{P}}$ with the oracle $f[\mu]$, where $\mu = \{((\mathbf{x}, \alpha_1, \tau), \rho)\}$ programs f with the random answer ρ for the query $(\mathbf{x}, \alpha_1, \tau)$. Moreover, the choice of $i \in [t+1]$ is independent of the execution of $\tilde{\mathcal{P}}$. If $(\mathbf{x}', \alpha'_1, \tau') = (\mathbf{x}_j, \alpha_{1,j}, \tau_j)$ for some $j \in [t]$ then $(\mathbf{x}', \alpha'_1, \tau') = (\mathbf{x}, \alpha_1, \tau)$ if $\tilde{\mathbf{P}}_{\text{SP}}$ sampled $i = j$ (note that in this case the query is well-formed and can be parsed). Otherwise, if $(\mathbf{x}', \alpha'_1, \tau')$ is not a query in the query-answer trace, then $(\mathbf{x}', \alpha'_1, \tau') = (\mathbf{x}, \alpha_1, \tau)$ if $\tilde{\mathbf{P}}_{\text{SP}}$ sampled $i = t+1$. We deduce that the probability that $(\mathbf{x}', \alpha'_1, \tau') = (\mathbf{x}, \alpha_1, \tau)$, and thus that $\alpha_2 \neq \perp$, is at least $\frac{1}{t+1}$. Therefore, the following two distributions are equivalent:

$$\left\{ (\mathbf{x}, \alpha_1, \rho, \alpha_2) \middle| \begin{array}{l} (\mathbf{x}, \alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \\ \text{conditioned on } \alpha_2 \neq \perp \end{array} \right\} \equiv \left\{ (\mathbf{x}, \alpha_1, \rho, \alpha_2) \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, (\alpha_1, \alpha_2, \tau)) \leftarrow \tilde{\mathcal{P}}^f \\ \rho := f(\mathbf{x}, \alpha_1, \tau) \end{array} \right\}. \quad (10.1)$$

Moreover, we can lower bound the probability that $\alpha_2 \neq \perp$ as follows:

$$\Pr \left[\alpha_2 \neq \perp \mid \begin{array}{l} (\mathbf{x}, \alpha_1, \mathbf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathbf{aux}, \rho) \end{array} \right] \geq \frac{1}{t+1}. \quad (10.2)$$

Using the above, we conclude that

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \mid \begin{array}{l} (\mathbf{x}, \alpha_1, \mathbf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathbf{aux}, \rho) \end{array} \right] \\ & \geq \Pr \left[\alpha_2 \neq \perp \mid \begin{array}{l} (\mathbf{x}, \alpha_1, \mathbf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathbf{aux}, \rho) \end{array} \right] \\ & \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \\ \text{conditioned on} \\ \alpha_2 \neq \perp \end{array} \mid \begin{array}{l} (\mathbf{x}, \alpha_1, \mathbf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathbf{aux}, \rho) \end{array} \right] \\ & \geq \frac{1}{t+1} \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, f(\mathbf{x}, \alpha_1, \tau), \alpha_2) = 1 \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, (\alpha_1, \alpha_2, \tau)) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ & = \frac{1}{t+1} \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right]. \end{aligned}$$

□

11 Additional security definitions

We prove that Construction 10.1.1 satisfies additional security definitions.

11.1 Knowledge soundness

In Construction 10.1.1, if the SP satisfies knowledge soundness then the resulting non-interactive argument satisfies adaptive knowledge soundness.

Theorem 11.1.1

Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP for a relation \mathcal{R} with rewinding knowledge soundness error κ_{SP} with extraction time \mathbf{et}_{SP} (see Definition 8.1.4). For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{NARG} := \mathbf{FS}_{\text{SP}}[\text{SP}, \lambda, s]$ in Construction 10.1.1 is a non-interactive argument for \mathcal{R} with rewinding knowledge soundness error κ_{ARG} with extraction time \mathbf{et}_{ARG} (see Definition 7.1.7) such that

- $\kappa_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}}(\lambda, n)) \leq (t+1) \cdot \kappa_{\text{SP}}(n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n))$, and
- $\mathbf{et}_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}}(\lambda, n), \tau_{\tilde{\mathcal{P}}}(\lambda, n)) \leq \mathbf{et}_{\text{SP}}(n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n), \tau'_{\tilde{\mathcal{P}}}(\lambda, n))$.

Above, $\delta'_{\tilde{\mathcal{P}}}(\lambda, n) := 1 - \frac{1 - \delta_{\tilde{\mathcal{P}}}(\lambda, n)}{t+1}$ and $\tau'_{\tilde{\mathcal{P}}}(\lambda, n) := \tau_{\tilde{\mathcal{P}}}(\lambda, n) + O(r \cdot t)$.

Moreover, if the SP extractor is straightline then the NARG extractor is also straightline (see Definition 7.1.5). In this case:

- the (straightline) knowledge soundness error is $\kappa_{\text{ARG}}(\lambda, t, n) \leq (t+1) \cdot \kappa_{\text{SP}}(n)$; and
- the (straightline) extraction time is $\mathbf{et}_{\text{ARG}}(\lambda, t, n) \leq \mathbf{et}_{\text{SP}}(n)$.

The theorem directly follows from the lemma below.

Lemma 11.1.2. Let \mathbf{E}_{SP} be the SP extractor for $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$. There exists an argument extractor \mathcal{E} such that for every query bound $t \in \mathbb{N}$, malicious t -query argument prover $\tilde{\mathcal{P}}$, and instance size bound $n \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}^f(\mathbf{x}, \pi) \\ \mathbf{w} \leftarrow \mathcal{E}(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}}) \end{array} \right] \\ \leq (t+1) \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}) \end{array} \right].$$

Moreover, the running time of \mathcal{E} is $\text{et}_{\text{SP}}\left(n, 1 - \frac{1-\delta_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x})}{t+1}, \tau_{\tilde{\mathcal{P}}}(\lambda, \mathbf{x}) + O(r \cdot t)\right)$.

Construction 11.1.3. The argument extractor \mathcal{E} receives as input an instance \mathbf{x} , argument string π , query-answer trace tr of the argument prover, query-answer trace tr_v of the argument verifier, and black-box access to the argument prover $\tilde{\mathcal{P}}$, and works as follows.

$\mathcal{E}(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}})$:

1. Parse the argument string π as a tuple $(\alpha_1, \alpha_2, \tau)$.
2. Parse the query-answer trace tr_v of the argument verifier as a single pair $((\mathbf{x}, \alpha_1, \tau), \rho)$.
3. Let $\tilde{\mathbf{P}}_{\text{SP}} := \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}})$ be the SP prover in Construction 9.3.3.
4. Compute the witness: $w \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}))$.
5. Output w .

Proof. We upper bound the rewinding knowledge soundness error. Note that \mathcal{V} invokes the random oracle f once only: the query $(\mathbf{x}, \alpha_1, \tau)$, which is answered via $\rho := f(\mathbf{x}, \alpha_1, \tau)$. We can write

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, w) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}^f(\mathbf{x}, \pi) \\ w \leftarrow \mathcal{E}(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}}) \end{array} \right] \\ &= \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, w) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, (\alpha_1, \alpha_2, \tau)) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ \rho := f(\mathbf{x}, \alpha_1, \tau) \\ w \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}})) \end{array} \right] \quad (\text{by definition of } \mathcal{E}) \\ &= \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, w) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \\ \text{conditioned on} \\ \alpha_2 \neq \perp \end{array} \middle| \begin{array}{l} (\mathbf{x}, \alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \\ w \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}) \end{array} \right] \quad (\text{by Equation 10.1}) \\ &= \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, w) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \\ \wedge \alpha_2 \neq \perp \end{array} \middle| \begin{array}{l} (\mathbf{x}, \alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \\ w \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}) \end{array} \right] \\ &\quad / \Pr \left[\alpha_2 \neq \perp \middle| \begin{array}{l} (\mathbf{x}, \alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \end{array} \right] \\ &\leq (t+1) \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, w) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \\ w \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}) \end{array} \right]. \end{aligned}$$

The last inequality comes from Equation 10.2 and the fact that $\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1$ implies that $\alpha_2 \neq \perp$.

Next, we discuss the running time of \mathcal{E} . This depends on the running time of \mathbf{E}_{SP} , which in turn depends on the failure probability and running time of $\tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}})$. Observe that Lemma 10.2.2

does not suffice to upper bound the failure probability of $\tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}})$, because the probability events in the lemma include the condition $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$, while the failure probability does not include this condition. Nevertheless, using Equations 10.1 and 10.2 in the proof of Lemma 10.2.2, we can straightforwardly upper bound the failure probability (by lower bounding the success probability):

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \alpha_1, \mathbf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathbf{aux}, \rho) \end{array} \right] \\ & \geq \Pr \left[\alpha_2 \neq \perp \middle| \begin{array}{l} (\mathbf{x}, \alpha_1, \mathbf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathbf{aux}, \rho) \end{array} \right] \\ & \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \\ \text{conditioned on} \\ \alpha_2 \neq \perp \end{array} \middle| \begin{array}{l} (\mathbf{x}, \alpha_1, \mathbf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathbf{aux}, \rho) \end{array} \right] \\ & \geq \frac{1}{t+1} \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, f(\mathbf{x}, \alpha_1, \tau), \alpha_2) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, (\alpha_1, \alpha_2, \tau)) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ & = \frac{1}{t+1} \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f \right]. \end{aligned}$$

Taking the complement events and rearranging:

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| > n \\ \vee \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 0 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \alpha_1, \mathbf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathbf{aux}, \rho) \end{array} \right] \\ & \leq 1 - \frac{1}{t+1} \cdot \left(1 - \Pr \left[\begin{array}{l} |\mathbf{x}| > n \\ \vee \mathcal{V}^f(\mathbf{x}, \pi) = 0 \end{array} \middle| (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f \right] \right) \\ & = 1 - \frac{1}{t+1} \cdot (1 - \delta_{\tilde{\mathcal{P}}}(\lambda, n)). \end{aligned}$$

We deduce that the failure probability of $\tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}})$ is $\delta'_{\tilde{\mathcal{P}}}(\lambda, n) := 1 - \frac{1-\delta_{\tilde{\mathcal{P}}}(\lambda, n)}{t+1}$. Moreover, by inspection of Construction 9.3.3, the running time of $\tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}})$ is $\tau'_{\tilde{\mathcal{P}}}(\lambda, n) := \tau_{\tilde{\mathcal{P}}}(\lambda, n) + O(r \cdot t)$. Therefore, the running time of the SP extractor \mathbf{E}_{SP} , as invoked by \mathcal{E} , is $\mathbf{et}_{\text{SP}}(n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n), \tau'_{\tilde{\mathcal{P}}}(\lambda, n))$. All other operations in \mathcal{E} are merely syntactic. \square

We conclude by discussing the special case of straightline extraction. Suppose that \mathbf{E}_{SP} is a straightline extractor, which means that \mathbf{E}_{SP} is a deterministic algorithm that does not receive as input the SP prover $\tilde{\mathbf{P}}_{\text{SP}}$. Then \mathcal{E} in Construction 11.1.3 is a straightline extractor because \mathcal{E} uses $\tilde{\mathcal{P}}$ only to construct $\tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathcal{P}})$, which \mathbf{E}_{SP} no longer needs. In this case, κ_{SP} does not depend on the failure probability of $\tilde{\mathbf{P}}_{\text{SP}}$, so the knowledge soundness error bound simplifies to $\kappa_{\text{ARG}}(\lambda, t, n) \leq (t+1) \cdot \kappa_{\text{SP}}(n)$. Similarly, \mathbf{et}_{SP} does not depend on the failure probability or running time of $\tilde{\mathbf{P}}_{\text{SP}}$, so the extraction time bound simplifies to $\mathbf{et}_{\text{ARG}}(\lambda, t, n) \leq \mathbf{et}_{\text{SP}}(n)$.

11.2 Zero knowledge

In Construction 10.1.1, if the SP satisfies honest-verifier zero knowledge (and the privacy parameter s of the construction is large enough), then the resulting non-interactive argument satisfies adaptive zero knowledge.

Theorem 11.2.1

Let SP be an SP for a relation \mathcal{R} with honest-verifier zero-knowledge error z_{SP} (see Definition 8.1.6). For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, Construction 10.1.1 is a non-interactive argument for \mathcal{R} with adaptive zero-knowledge error z_{ARG} (see Definition 7.1.8) such that

$$z_{\text{ARG}}(\lambda, t, n) \leq z_{\text{SP}}(n) + \frac{t}{2^s}.$$

Construction 11.2.2. The simulator is an algorithm $\mathcal{S}^f(\mathbf{x})$ that works as follows. Below we denote by \mathbf{S}_{SP} the honest-verifier zero-knowledge simulator of the SP (see Definition 8.1.6).

- $\mathcal{S}^f(\mathbf{x})$:
 1. Sample a simulated view of the SP verifier: $(\mathbf{x}, \alpha_1, \rho, \alpha_2) \leftarrow \mathbf{S}_{\text{SP}}(\mathbf{x})$.
 2. Sample a random salt $\tau \in \{0, 1\}^s$.
 3. Set the argument string $\pi := (\alpha_1, \alpha_2, \tau)$.
 4. Set the list of query-answer pairs $\mu := \{((\mathbf{x}, \alpha_1, \tau), \rho)\}$.
 5. Output (π, μ) .

Note that \mathcal{S} programs the oracle f at one point via the output μ (and does not query f).

Proof. Fix a query bound $t \in \mathbb{N}$, t -query admissible adversary \mathcal{A} , and instance bound $n \in \mathbb{N}$. We wish to upper bound the statistical distance between the output of \mathcal{A} in the real world and in the simulated world. For that, we introduce a hybrid simulator (that takes the witness as input and programs the oracle): $\mathcal{S}_H^f(\mathbf{x}, \mathbf{w})$ acts as $\mathcal{S}^f(\mathbf{x})$ except that the view in Item 1 is sampled as a real view $(\mathbf{x}, \alpha_1, \rho, \alpha_2) \leftarrow \text{View}_{\text{SP}}(\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}}, \mathbf{x}, \mathbf{w})$. We list the real-world, hybrid-world, and simulated-world distributions, making explicit the operations underlying the relevant algorithms.

1. The real-world distribution:

$$\mathcal{D}_{\text{real}} := \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}^f \\ \pi \leftarrow \mathcal{P}^f(\mathbf{x}, \mathbf{w}) \\ \text{out} \leftarrow \mathcal{A}^f(\text{aux}, \pi) \end{array} \right. \right\} \equiv \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}^f \\ (\alpha_1, \text{aux}) \leftarrow \mathbf{P}_{\text{SP}}(\mathbf{x}, \mathbf{w}) \\ \tau \leftarrow \{0, 1\}^s \\ \rho := f(\mathbf{x}, \alpha_1, \tau) \\ \alpha_2 \leftarrow \mathbf{P}_{\text{SP}}(\text{aux}, \rho) \\ \pi := (\alpha_1, \alpha_2, \tau) \\ \text{out} \leftarrow \mathcal{A}^f(\text{aux}, \pi) \end{array} \right. \right\}.$$

2. The hybrid-world distribution:

$$\mathcal{D}_H := \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ (\pi, \mu) \leftarrow \mathcal{S}_H^f(\mathbf{x}, \mathbf{w}) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right. \right\} \equiv \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ (\alpha_1, \mathbf{aux}) \leftarrow \mathbf{P}_{SP}(\mathbf{x}, \mathbf{w}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \mathbf{P}_{SP}(\mathbf{aux}, \rho) \\ \tau \leftarrow \{0, 1\}^s \\ \pi := (\alpha_1, \alpha_2, \tau) \\ \mu := \{((\mathbf{x}, \alpha_1, \tau), \rho)\} \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right. \right\} .$$

3. The simulated-world distribution:

$$\mathcal{D}_{\text{sim}} := \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ (\pi, \mu) \leftarrow \mathcal{S}^f(\mathbf{x}) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right. \right\} \equiv \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ (\mathbf{x}, \alpha_1, \rho, \alpha_2) \leftarrow \mathbf{S}_{SP}(\mathbf{x}) \\ \tau \leftarrow \{0, 1\}^s \\ \pi := (\alpha_1, \alpha_2, \tau) \\ \mu := \{((\mathbf{x}, \alpha_1, \tau), \rho)\} \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right. \right\} .$$

Next we analyze the statistical difference between these distributions.

- *Real versus hybrid.* We analyze the statistical distance between $\mathcal{D}_{\text{real}}$ and \mathcal{D}_H .

The two distributions only differ in that the hybrid simulator \mathcal{S}_H programs the oracle f by setting the answer to the query $(\mathbf{x}, \alpha_1, \tau)$ to be a freshly sampled SP challenge ρ (independent of f), whereas the argument prover \mathcal{P} does not program the oracle (it sets ρ to be the answer of f to the query $(\mathbf{x}, \alpha_1, \tau)$).

The distribution of $((\mathbf{x}, \alpha_1, \tau), \rho)$ is the same in both cases. However, \mathcal{A} has access to f before f is programmed. Thus, \mathcal{A} can distinguish between the two distributions *only* if \mathcal{A} queries f at $(\mathbf{x}, \alpha_1, \tau)$ before f is programmed (and also after f is programmed).

The programmed query $(\mathbf{x}, \alpha_1, \tau)$ is such that $\tau \in \{0, 1\}^s$ is sampled independently and uniformly at random. The probability that \mathcal{A} queries $(\mathbf{x}, \alpha_1, \tau)$ before τ is sampled is at most $\frac{t}{2^s}$ (since \mathcal{A} makes at most t queries to f). In particular, this is an upper bound on the statistical distance between $\mathcal{D}_{\text{real}}$ and \mathcal{D}_H .

- *Hybrid versus simulated.* We analyze the statistical distance between \mathcal{D}_H and \mathcal{D}_{sim} .

The two distributions only differ in that \mathcal{S}_H samples a real SP view, and \mathcal{S} samples a simulated SP view. By the zero-knowledge property of the SP, the statistical distance between the SP views for $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ is at most $z_{SP}(\mathbf{x})$. Since \mathcal{A} outputs $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ with $|\mathbf{x}| \leq n$ (as \mathcal{A} is admissible), the statistical distance between \mathcal{D}_H and \mathcal{D}_{sim} is upper bounded by $z_{SP}(n)$.

Adding these error terms yields the statistical difference claimed in the lemma. \square

12 State restoration

We describe *state-restoration soundness*, a notion of soundness for sigma protocols (SPs). This notion requires security against a class of malicious provers called *state-restoration provers*, which are more powerful than the provers considered for standard soundness for SPs (Definition 8.1.2).

State-restoration soundness is a notion that plays a central role in this book. Briefly, we study non-interactive arguments constructed from different types of probabilistic proofs (SPs, IPs, PCPs, IOPs), and the security of these non-interactive arguments is characterized by state-restoration soundness of the probabilistic proof rather than by standard soundness of the probabilistic proof. In general, state-restoration soundness is a far stronger notion than standard soundness, so one cannot merely rely on the standard soundness of the probabilistic proof; instead, one must explicitly assume that the given probabilistic proof has good state-restoration soundness.

However, in special cases, good standard soundness *does* imply good state-restoration soundness. These special cases include SPs (studied in this part) and PCPs (studied in Part V). Therefore in the special cases of SPs and PCPs it is possible to directly relate the security of the non-interactive argument constructed from an SP or PCP to the standard soundness of the given probabilistic proof, without having to deal with state-restoration soundness of the probabilistic proof.

In this chapter we nevertheless study state-restoration soundness for SPs in detail. This clarifies how state-restoration soundness has played an implicit role in the security analysis of Construction 10.1.1, and also serves as a helpful warm up towards settings where we *must* consider state-restoration soundness (specifically, when starting from IPs in Part III or from IOPs in Part VI). Specifically, this chapter is organized as follows.

- In Section 12.1 we define state-restoration soundness for SPs.
- In Section 12.2 we express the security of the Fiat–Shamir transformation for SPs (Construction 10.1.1) in terms of state-restoration soundness.
- In Section 12.3 we compare state-restoration soundness and standard soundness.

Throughout we also consider state-restoration *knowledge* soundness.

12.1 Definition

The standard notion of soundness for an SP considers the setting where a malicious SP prover attempts to convince the SP verifier in a single interaction (see Definition 8.1.2): the malicious SP prover sends a first message, the SP verifier responds with some randomness, and the malicious SP prover sends a second message; then the SP verifier accepts or rejects based on (the instance and) this transcript of interaction. In contrast, state-restoration soundness considers the setting where a malicious SP prover attempts to convince the SP verifier multiple times across related interactions, and the malicious SP prover wins if it finds an interaction that convinces the SP verifier.

In more detail, we consider an *SP state-restoration game* that informally works as follows. The game samples a random function $\text{rnd} \in \mathcal{U}(r)$ to be used as SP verifier randomness. In this game, the goal of the malicious SP prover is to output an instance \mathbf{x} , SP prover messages (α_1, α_2) , and salt σ that convinces the SP verifier according to the following condition:

$$\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \quad \text{where} \quad \rho := \text{rnd}(\mathbf{x}, \alpha_1, \sigma).$$

The salt σ is a string of s bits, for some salt parameter $s \in \mathbb{N}$.

The malicious SP prover has a budget of $t \in \mathbb{N}$ moves to invest in choosing its output. A move consists of outputting a tuple $(\mathbf{x}, \alpha_1, \sigma)$ (possibly unrelated to the eventual output), and the game responds with corresponding randomness $\rho := \text{rnd}(\mathbf{x}, \alpha_1, \sigma)$.

On one extreme, if the malicious SP prover makes no moves, it has no information about the random function rnd . This means that it chooses an output $(\mathbf{x}, \alpha_1, \sigma, \alpha_2)$ without seeing any SP verifier random messages. This is a poor way to play the game.

A minimal strategy for the malicious SP prover is to recreate a single interaction with the SP verifier: output a move $(\mathbf{x}, \alpha_1, \sigma)$; then obtain, as a response from the game, the SP verifier challenge $\rho := \text{rnd}(\mathbf{x}, \alpha_1, \sigma)$; and finally output $(\mathbf{x}, \alpha_1, \sigma, \alpha_2)$ for some choices of second SP prover message α_2 . This enables the malicious SP prover to win with probability that is at least that of winning in a single interaction with the SP verifier.

However, the malicious SP prover can do more than this: the malicious SP prover can use moves to see multiple “random continuations” of an interaction with the SP verifier. For example, the malicious SP prover can first output a move $(\mathbf{x}, \alpha_1, \sigma)$ to obtain the SP verifier challenge $\rho := \text{rnd}(\mathbf{x}, \alpha_1, \sigma)$, and then also output the move $(\mathbf{x}, \alpha_1, \sigma')$ for some salt σ' different from σ to obtain another SP verifier challenge $\rho' := \text{rnd}(\mathbf{x}, \alpha_1, \sigma')$. This enables the malicious SP prover to choose, in hindsight, which of the two transcripts to complete, which increases the probability of convincing the SP verifier.

The salt parameter s limits how many times the malicious SP prover can extend the same partial transcript. For example, the malicious SP prover can see at most 2^s different choices of ρ for the same instance \mathbf{x} and first SP prover message α_1 . (Each s -bit salt leads to a fresh randomness choice.)

Definition 12.1.1. *The **SP state-restoration game** for $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ with salt size $s \in \mathbb{N}$, function $\text{rnd} \in \mathcal{U}(r)$, and SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ is defined below.*

$\text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$:

1. Repeat the following until $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ decides to exit the loop.
 - a) $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ outputs $(\mathbf{x}, \alpha_1, \sigma)$, where \mathbf{x} is an instance, $\alpha_1 \in \{0, 1\}^{\text{pv}_1}$ is an SP prover commitment, and $\sigma \in \{0, 1\}^s$ is a salt string.
 - b) Set $\rho := \text{rnd}(\mathbf{x}, \alpha_1, \sigma)$.
 - c) Send ρ to $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$.
2. $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ outputs $(\mathbf{x}, \alpha_1, \sigma, \alpha_2)$, where \mathbf{x} is an instance, $\alpha_1 \in \{0, 1\}^{\text{pv}_1}$ is an SP prover commitment, $\sigma \in \{0, 1\}^s$ is a salt string, and $\alpha_2 \in \{0, 1\}^{\text{pv}_2}$ is an SP prover response.
3. Set $\rho := \text{rnd}(\mathbf{x}, \alpha_1, \sigma)$.
4. Output $(\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2)$.

We denote by tr^{sr} the list of move-response pairs of the form $((\mathbf{x}, \alpha_1, \sigma), \rho)$ performed in the loop. We show tr^{sr} in an execution of the SP state-restoration game using the following notation:

$$(\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \xleftarrow{\text{tr}^{\text{sr}}} \text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}).$$

We say that $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ is **t -move** if $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ exits the loop after at most t iterations.

The above game directly leads to the notion of state-restoration soundness error: it is an upper bound on the probability that any SP prover in the SP state-restoration game can find an instance not in the language and an accepting transcript for it.

Definition 12.1.2. $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ has **state-restoration soundness error** $\epsilon_{\text{SP}}^{\text{sr}}$ if for every salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move malicious SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$, and instance size bound $n \in \mathbb{N}$:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \leftarrow \text{Game}_{\text{SP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right] \leq \epsilon_{\text{SP}}^{\text{sr}}(s, t, n).$$

We also define the notion of state-restoration *knowledge soundness* error: it is an upper bound on the probability that any SP prover in the SP state-restoration game can find an instance and an accepting transcript for it such that an extractor algorithm cannot find a witness for the instance (given the relevant information about the SP prover and its moves in the game).

In more detail, we consider two flavors of state-restoration knowledge soundness: **straightline** knowledge soundness, and a relaxation known as **rewinding** knowledge soundness. The former notion is primarily of pedagogical value because most SPs of interest only satisfy the latter notion.

The straightline variant of state-restoration knowledge soundness considers a (deterministic) knowledge extractor $\mathbf{E}_{\text{SP}}^{\text{sr}}$ tasked with finding a witness \mathbf{w} when given the final output $(\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2)$ of the state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ and its move-response trace tr^{sr} . The knowledge extractor $\mathbf{E}_{\text{SP}}^{\text{sr}}$ does not receive the randomness ρ used by the SP verifier.

Definition 12.1.3. $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ has **straightline state-restoration knowledge soundness error** $\kappa_{\text{SP}}^{\text{sr}}$ if there exists a polynomial-time deterministic algorithm $\mathbf{E}_{\text{SP}}^{\text{sr}}$ (the extractor) such that for every salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move deterministic SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$, and instance size bound n :

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

The rewinding variant of state-restoration knowledge soundness relaxes the prior notion by considering a knowledge extractor that additionally receives the randomness ρ used by the SP verifier and black-box access to the state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$. In this case, the error may additionally depend on the failure probability of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ (an upper bound on the probability that the final output of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ does *not* convince the SP verifier). Intuitively, as the failure probability increases, the error of extraction increases.

Definition 12.1.4. Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP. A deterministic SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ has **failure probability** $\delta_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}$ if for every salt size $s \in \mathbb{N}$ and instance size bound n :

$$\Pr \left[\begin{array}{l} |\mathbf{x}| > n \\ \vee \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 0 \end{array} \middle| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \leftarrow \text{Game}_{\text{SP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right] \leq \delta_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n).$$

Definition 12.1.5. We say that $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ has **rewinding state-restoration knowledge soundness error** $\kappa_{\text{SP}}^{\text{sr}}$ with **extraction time** $\text{et}_{\text{SP}}^{\text{sr}}$ if there exists a probabilistic algorithm $\mathbf{E}_{\text{SP}}^{\text{sr}}$ (the extractor) such that for every salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move deterministic SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ with failure probability $\delta_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}$ and running time $\tau_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}$, and instance size bound n :

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \xleftarrow{\text{tr}^{\text{sr}}} \text{Game}_{\text{SP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}^{\text{sr}}(\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2, \text{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right] \\ \leq \kappa_{\text{SP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n)). \end{math>$$

Moreover, $\mathbf{E}_{\text{SP}}^{\text{sr}}$ runs in expected time $\text{et}_{\text{SP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n), \tau_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n))$ (over the given inputs and internal randomness).

12.2 Security from state restoration

The state-restoration game for an SP is closely related to the security of the non-interactive argument obtained by applying the Fiat–Shamir transformation to the SP.

Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP and let $\text{NARG} = (\mathcal{P}, \mathcal{V})$ be the non-interactive argument $\text{NARG} := \text{FS}_{\text{SP}}[\text{SP}, \lambda, s]$ (see Construction 10.1.1). There is a straightforward equivalence between two types of provers: (i) provers $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ in the SP state-restoration game $\text{Game}_{\text{SP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$ for SP; and (ii) provers $\tilde{\mathcal{P}}$ against the argument verifier \mathcal{V} .

- An argument prover $\tilde{\mathcal{P}}$ against \mathcal{V} can be viewed as an SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}} := \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}(\tilde{\mathcal{P}})$ in the game $\text{Game}_{\text{SP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$. Queries of $\tilde{\mathcal{P}}$ to the random oracle are interpreted as move in the SP state-restoration game; this excludes “garbage” queries that cannot be interpreted as moves, which $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ answers using internal randomness.

$\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}(\tilde{\mathcal{P}})$:

1. Lazily sample an oracle $\dot{g} \leftarrow \mathcal{U}(\mathbf{r})$ (to answer malformed queries).
2. Simulate $\tilde{\mathcal{P}}$ while answering a query x to f as follows:
 - a) If x can be parsed as a tuple $(\mathbf{x}, \alpha_1, \sigma)$ (where \mathbf{x} is an instance, $\alpha_1 \in \{0, 1\}^{p_{V_1}}$ is an SP prover commitment, and $\sigma \in \{0, 1\}^s$ is a salt string):
 - i. Output the move $(\mathbf{x}, \alpha_1, \sigma)$ to the SP state-restoration game.
 - ii. Receive the response ρ from the SP state-restoration game.
 - b) Otherwise set $\rho := \dot{g}(x)$.
 - c) Return ρ as the answer to the query.
3. The simulation of $\tilde{\mathcal{P}}$ ends with an output (\mathbf{x}, π) , where $\pi = (\alpha_1, \alpha_2, \tau)$.
4. Set the salt string $\sigma := \tau$.
5. The final output is $(\mathbf{x}, \alpha_1, \sigma, \alpha_2)$.

- Conversely, a prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ in the SP state-restoration game $\text{Game}_{\text{SP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$ can be viewed as an argument prover $\tilde{\mathcal{P}} := \tilde{\mathcal{P}}(\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$ against \mathcal{V} . Each move of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ in the SP state-restoration game is mapped to a query to the random oracle. (There are no “garbage” moves in this case.)

This equivalence directly characterizes the adaptive soundness of Construction 10.1.1 in terms of the state-restoration soundness of the underlying SP.

Lemma 12.2.1. Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP for a relation \mathcal{R} with state-restoration soundness error $\epsilon_{\text{SP}}^{\text{sr}}$. For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{NARG} := \mathbf{FS}_{\text{SP}}[\text{SP}, \lambda, s]$ in Construction 10.1.1 is a non-interactive argument for \mathcal{R} with adaptive soundness error ϵ_{ARG} (see Definition 7.1.4) such that

$$\epsilon_{\text{ARG}}(\lambda, t, n) \leq \epsilon_{\text{SP}}^{\text{sr}}(s, t, n).$$

In fact, if $\epsilon_{\text{SP}}^{\text{sr}}$ is a tight upper bound then $\epsilon_{\text{ARG}}(\lambda, t, n) \geq \epsilon_{\text{SP}}^{\text{sr}}(s, t, n)$.

The above correspondence also tells us about the *knowledge* soundness of Construction 10.1.1 in terms of the state-restoration *knowledge* soundness of the underlying SP. Given an SP state-restoration extractor $\mathbf{E}_{\text{SP}}^{\text{sr}}$, we construct an argument extractor \mathcal{E} for $\text{NARG} = (\mathcal{P}, \mathcal{V})$ (intended to fulfill Definition 7.1.7). The argument extractor \mathcal{E} receives as input an instance \mathbf{x} , argument string π , query-answer trace \mathbf{tr} of the argument prover, query-answer trace \mathbf{tr}_v of the argument verifier, and black-box access to the argument prover $\tilde{\mathcal{P}}$, and works as follows.

$\mathcal{E}(\mathbf{x}, \pi, \mathbf{tr}, \mathbf{tr}_v, \tilde{\mathcal{P}}):$

1. Parse the argument string π as a tuple $(\alpha_1, \alpha_2, \tau)$.
2. Set the salt string $\sigma := \tau$.
3. Set the move-response trace \mathbf{tr}^{sr} equal to the query-answer trace \mathbf{tr} , ignoring any entries in \mathbf{tr} where queries cannot be parsed as moves of the SP state-restoration game.
4. Parse the trace \mathbf{tr}_v as a single query-answer pair $((\mathbf{x}, \alpha_1, \tau), \rho)$.
5. Set $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}} := \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}(\tilde{\mathcal{P}})$ to be the SP state-restoration prover defined above.
6. Compute the witness: $\mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}^{\text{sr}}(\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2, \mathbf{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$.
7. Output \mathbf{w} .

The operations above are syntactically allowed as we now explain.

- Salts in the non-interactive argument and in the SP state restoration game are of the same type, so it is possible to set $\sigma := \tau$.
- The moves of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}(\tilde{\mathcal{P}})$ are the queries of $\tilde{\mathcal{P}}$ that can be parsed as moves, so if \mathbf{tr} is the query-answer trace of $\tilde{\mathcal{P}}$ then \mathbf{tr}^{sr} as defined above is the move-response trace of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}(\tilde{\mathcal{P}})$.

The construction of \mathcal{E} from $\mathbf{E}_{\text{SP}}^{\text{sr}}$ directly implies the following lemma. Indeed, $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}(\tilde{\mathcal{P}})$ has the same failure probability as $\tilde{\mathcal{P}}$, and the running time of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}(\tilde{\mathcal{P}})$ is the running time of $\tilde{\mathcal{P}}$ plus $O(r \cdot t)$ (to identify and drop queries that cannot be interpreted as moves).

Lemma 12.2.2. Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP for a relation \mathcal{R} with rewinding state-restoration knowledge soundness error $\kappa_{\text{SP}}^{\text{sr}}$ with extraction time $\mathbf{et}_{\text{SP}}^{\text{sr}}$ (see Definition 12.1.5). For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{NARG} := \mathbf{FS}_{\text{SP}}[\text{SP}, \lambda, s]$ in Construction 10.1.1 is a non-interactive argument for \mathcal{R} with rewinding knowledge soundness error κ_{ARG} with extraction time \mathbf{et}_{ARG} (Definition 7.1.7) such that

- $\kappa_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}}) \leq \kappa_{\text{SP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathcal{P}}})$, and
- $\mathbf{et}_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}}, \tau_{\tilde{\mathcal{P}}}) \leq \mathbf{et}_{\text{SP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathcal{P}}}, \tau_{\tilde{\mathcal{P}}} + O(r \cdot t))$.

Moreover, if the SP state-restoration extractor is straightline (Definition 12.1.3) then the NARG extractor is also straightline (see Definition 7.1.5). In this case:

- the (straightline) knowledge soundness error is $\kappa_{\text{ARG}}(\lambda, t, n) \leq \kappa_{\text{SP}}^{\text{sr}}(s, t, n)$; and
- the (straightline) extraction time is $\mathbf{et}_{\text{ARG}}(\lambda, t, n) \leq \mathbf{et}_{\text{SP}}^{\text{sr}}(s, t, n)$.

12.3 Comparison with standard soundness notions

We compare state-restoration soundness and knowledge soundness for an SP to the standard notions of soundness and knowledge soundness.

The SP state-restoration game gives the prover $t + 1$ attempts at convincing the SP verifier (once per loop and once after all loops), each time possibly with a different instance and transcript of interaction. Intuitively, the maximum winning probability of any cheating prover, for instances not in the language, is at most $t + 1$ times the SP soundness error. Moreover, this upper bound is essentially tight because, in common parameter regimes, the SP state restoration soundness error is at least $\Omega(t)$ times the SP soundness error. We prove this next.

Lemma 12.3.1. *Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP. If SP has soundness error ϵ_{SP} then SP has state-restoration soundness error $\epsilon_{\text{SP}}^{\text{sr}}$ such that*

$$\epsilon_{\text{SP}}^{\text{sr}}(s, t, n) \leq (t + 1) \cdot \epsilon_{\text{SP}}(n).$$

Note that the salt size s does not affect the upper bound. Moreover,

$$\epsilon_{\text{SP}}^{\text{sr}}(s, t, n) \geq \min\{t, 2^s\} \cdot \epsilon_{\text{SP}}(n) - \binom{\min\{t, 2^s\}}{2} \cdot \epsilon_{\text{SP}}(n)^2.$$

Proof of upper bound. We show that

$$\epsilon_{\text{SP}}^{\text{sr}}(s, t, n) \leq (t + 1) \cdot \epsilon_{\text{SP}}(n).$$

We describe a SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ such that for every salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move malicious SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$, and instance size bound $n \in \mathbb{N}$ satisfies:

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \leftarrow \text{Game}_{\text{SP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right] \\ & \leq (t + 1) \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \alpha_1, \mathbf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathbf{aux}, \rho) \end{array} \right]. \end{aligned}$$

This suffices to prove the lemma because, by soundness of the SP (see Definition 8.1.2), the probability that an SP prover convinces the SP verifier on an instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ is at most $\epsilon_{\text{SP}}(\mathbf{x})$; therefore, since the experiment above only considers instances $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ with $|\mathbf{x}| \leq n$, the above quantity is upper bounded by $(t + 1) \cdot \epsilon_{\text{SP}}(n)$.

We construct the SP prover and argue the inequality.

Construction 12.3.2. The SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ receives as input a salt size $s \in \mathbb{N}$ and move budget $t \in \mathbb{N}$ and black-box access to a SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$, and works as follows.

$\tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \rightarrow (\mathbf{x}, \alpha_1, \mathbf{aux})$:

1. Lazily sample an oracle $\mathbf{rnd} \leftarrow \mathcal{U}(r)$.
2. Sample a random index $j \in [t + 1]$.
3. If $j \in [t]$ then emulate $\text{Game}_{\text{SP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$ up to the j -th output $(\mathbf{x}', \alpha'_1, \sigma')$ of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ (there are at most t outputs in the loop in Item 1), and set $\alpha'_2 := \perp$.

4. Otherwise ($j = t + 1$), emulate $\text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$ up to its final output $(\mathbf{x}', \alpha'_1, \sigma', \alpha'_2)$. In this case, $\alpha'_2 \neq \perp$.
5. Let aux^{sr} be the state of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$.
6. Output $(\mathbf{x}, \alpha_1, \text{aux})$ where $\mathbf{x} := \mathbf{x}'$, $\alpha'_1 := \alpha_1$, and $\text{aux} := (\text{aux}^{\text{sr}}, \text{rnd}, \alpha'_2)$.

$\tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \rightarrow \alpha_2$:

1. Parse aux as a tuple $(\text{aux}^{\text{sr}}, \text{rnd}, \alpha'_2)$.
2. If $\alpha'_2 \neq \perp$ then output $\alpha_2 := \alpha'_2$.
3. Otherwise, use aux^{sr} and rnd to continue emulating $\text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$ while answering the j -th move (and any other moves equal to this move) with ρ .
4. Let $(\mathbf{x}', \alpha'_1, \sigma', \alpha'_2)$ be the final output of the game.
5. Output $\alpha_2 := \alpha'_2$.

The SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ produces at most $t + 1$ outputs: one output per iteration of the loop (there are at most t iterations), and a final output after the loop. The final output either equals an earlier output or not. We denote by I the set of indices of any appearance of the final output: I contains the index of any iteration in which the final output appears, or (if the final output first appears after the loop) I is a singleton containing the number of iterations plus one. Note that I is a random variable over $\{1, \dots, t + 1\}$ that depends on rnd . Whenever $j \in I$, the j -th output of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ equals the final output of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$, so in this case $\tilde{\mathbf{P}}_{\text{SP}}$ and $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ output the same instance \mathbf{x} , SP commitment α_1 , and SP response α_2 . Moreover, the SP verifier randomness ρ is sampled from $\{0, 1\}^r$, the same distribution as $\rho := \text{rnd}(\mathbf{x}, \alpha_1, \sigma)$ for a random choice of $\text{rnd} \in \mathcal{U}(r)$. This allows us to conclude the following about the probability that \mathbf{V}_{SP} accepts.

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \end{array} \right] \\ & \geq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \\ \wedge j \in I \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \leftarrow \text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \\ j \leftarrow [t + 1] \end{array} \right]. \end{aligned}$$

Since j and I are independent, $\Pr[j \in I] \geq \frac{1}{t+1}$. This lets us lower bound the previous probability:

$$\geq \frac{1}{t+1} \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \leftarrow \text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right].$$

We conclude that:

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \leftarrow \text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right] \\ & \leq (t+1) \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \end{array} \right] \end{aligned}$$

$$\leq (t+1) \cdot \epsilon_{\text{SP}}(n).$$

Salts play a meaningful role in the SP state-restoration game (in that they lead to a more general class of games) but do not affect the above argument. \square

Proof of lower bound. We show that

$$\epsilon_{\text{SP}}^{\text{sr}}(s, t, n) \geq \min\{t, 2^s\} \cdot \epsilon_{\text{SP}}(n) - \binom{\min\{t, 2^s\}}{2} \cdot \epsilon_{\text{SP}}(n)^2.$$

We describe a “universal” state-restoration attack. Let $\tilde{\mathbf{P}}_{\text{SP}}$ be a malicious SP prover that convinces the SP verifier \mathbf{V}_{SP} with probability exactly ϵ_{SP} ; without loss of generality, $\tilde{\mathbf{P}}_{\text{SP}}$ is deterministic. Consider the SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ that, in the SP state-restoration game, runs $\tilde{\mathbf{P}}_{\text{SP}}$ as many times as possible each time with a unique salt (thereby resulting in a fresh SP verifier challenge), in an attempt to convince the SP verifier.

$\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$:

1. Compute $(\mathbf{x}, \alpha_1, \mathbf{aux}) := \tilde{\mathbf{P}}_{\text{SP}}$ (the instance and SP commitment output by $\tilde{\mathbf{P}}_{\text{SP}}$).
2. For each $\sigma \in \{0, 1\}^s$ (or until the move budget t is exhausted):
 - a) Output the move $(\mathbf{x}, \alpha_1, \sigma)$, in order to receive the response $\rho := \text{rnd}(\mathbf{x}, \alpha_1, \sigma)$.
 - b) Compute $\alpha_2 := \tilde{\mathbf{P}}_{\text{SP}}(\mathbf{aux}, \rho)$ (the SP response output by $\tilde{\mathbf{P}}_{\text{SP}}$ given ρ).
 - c) If $\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1$ then output $(\mathbf{x}, \alpha_1, \sigma, \alpha_2)$.

Note that $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ performs at most $\min\{t, 2^s\}$ moves. The probability that the output of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ convinces \mathbf{V}_{SP} is the probability that at least one of the $\min\{t, 2^s\}$ moves convinces \mathbf{V}_{SP} . Each move causes \mathbf{V}_{SP} to accept with probability ϵ_{SP} (independently of any other executions) because the SP state-restoration game responds with a fresh SP verifier challenge to each unique move (moves are unique since the salts are unique). By the simple inclusion-exclusion principle (Lemma 1.2.3), the probability that at least one move convinces \mathbf{V}_{SP} is at least the lower bound stated above. \square

Next, we compare knowledge soundness and state-restoration knowledge soundness. Intuitively, similarly to the case of soundness, we expect a multiplicative loss of $t+1$ in the knowledge soundness error, due to the fact that, in the SP state-restoration game, the prover has $t+1$ attempts at convincing the SP verifier. Less obvious is an additional loss, this time in the failure probability: the transformation from an SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ to an SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ incurs a multiplicative loss of $t+1$ in convincing probability, and this loss (potentially) affects the knowledge soundness error and extraction time. Nevertheless, this second loss is sometimes immaterial because there are settings in which the failure probability does not affect the knowledge soundness error or extraction time (e.g., the setting of special soundness discussed in Section 30.2). More generally, the impact of the loss in failure probability depends on the specifics of the functions that determine the knowledge soundness error and extraction time. Overall, we prove the following lemma.

Lemma 12.3.3. *Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP. If SP has rewinding knowledge soundness error κ_{SP} with extraction time et_{SP} then SP has state-restoration knowledge soundness error $\kappa_{\text{SP}}^{\text{sr}}$ with extraction time $\text{et}_{\text{SP}}^{\text{sr}}$ such that*

- $\kappa_{\text{SP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n)) \leq (t+1) \cdot \kappa_{\text{SP}}(n, \delta'_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n))$, and

$$\bullet \quad \mathbf{et}_{\text{SP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n), \tau_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n)) \leq \mathbf{et}_{\text{SP}}(n, \delta'_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n), \tau'_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n)).$$

Above, $\delta'_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n) := 1 - \frac{1 - \delta_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n)}{t+1}$ and $\tau'_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n) := \tau_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n) + O(t \cdot r)$.

Moreover, if the SP extractor is straightline then the SP state-restoration extractor is also straightline; in that case, the upper bound on the knowledge soundness error is $\kappa_{\text{SP}}^{\text{sr}}(s, t, n) \leq (t+1) \cdot \kappa_{\text{SP}}(n)$ and the upper bound on the extraction time is $\mathbf{et}_{\text{SP}}^{\text{sr}}(s, t, n) \leq \mathbf{et}_{\text{SP}}(n)$.

Proof. The proof is similar to the proof of Lemma 12.3.1 (state-restoration soundness). Let $\tilde{\mathbf{P}}_{\text{SP}}$ be the SP prover in Construction 12.3.2 obtained from an SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$. Let \mathbf{E}_{SP} be the SP extractor. We construct an SP state-restoration extractor $\mathbf{E}_{\text{SP}}^{\text{sr}}$ as follows:

$$\mathbf{E}_{\text{SP}}^{\text{sr}}(\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2, \mathbf{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) := \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})).$$

It is straightforward to argue that the failure probability of $\tilde{\mathbf{P}}_{\text{SP}}$ satisfies $\delta'_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n) := 1 - \frac{1 - \delta_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n)}{t+1}$. Thus, the running time of $\mathbf{E}_{\text{SP}}^{\text{sr}}$ is bounded by $\mathbf{et}_{\text{SP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n), \tau_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n)) \leq \mathbf{et}_{\text{SP}}(n, \delta'_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n), \tau'_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n))$ since the running time of $\tilde{\mathbf{P}}_{\text{SP}}$ is $\tau'_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n) := \tau_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}(s, n) + O(t \cdot r)$ ($\tilde{\mathbf{P}}_{\text{SP}}$ simulates $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ in an SP state-restoration game).

We are left to argue the success probability of this extractor.

The SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ produces at most $t+1$ outputs: one output per iteration of the loop (there are at most t iterations), and a final output after the loop. The final output either equals an earlier output or not. We denote by I the set of indices of any appearance of the final output: I contains the index of any iteration in which the final output appears, or (if the final output first appears after the loop) I is a singleton containing the number of iterations plus one. Note that I is a random variable over $\{1, \dots, t+1\}$ that depends on \mathbf{rnd} . Whenever $j \in I$, the j -th output of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ equals the final output of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$, so in this case $\tilde{\mathbf{P}}_{\text{SP}}$ and $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ output the same instance \mathbf{x} , SP commitment α_1 , and SP response α_2 . Moreover, the SP verifier randomness ρ is sampled from $\{0, 1\}^r$, the same distribution as $\rho := \mathbf{rnd}(\mathbf{x}, \alpha_1, \sigma)$ for a random choice of $\mathbf{rnd} \in \mathcal{U}(r)$. This allows us to conclude the following about the success probability of the extractor.

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \right] \left| \begin{array}{l} (\alpha_1, \mathbf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathbf{aux}, \rho) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})) \end{array} \right. \right] \\ & \geq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \\ \wedge j \in I \end{array} \right] \left| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \xleftarrow{\mathbf{tr}^{\text{sr}}} \mathbf{Game}_{\text{SP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \\ j \leftarrow [t+1] \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}^{\text{sr}}(\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2, \mathbf{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right. \right]. \end{aligned}$$

Since j and I are independent, $\Pr[j \in I] \geq \frac{1}{t+1}$. This lets us lower bound the previous probability:

$$\geq \frac{1}{t+1} \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \right] \left| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \xleftarrow{\mathbf{tr}^{\text{sr}}} \mathbf{Game}_{\text{SP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}^{\text{sr}}(\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2, \mathbf{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right. \right].$$

This lets us conclude that:

$$\begin{aligned}
 & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \xleftarrow{\text{tr}} \text{Game}_{\text{SP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}^{\text{sr}}(\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2, \text{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right] \\
 & \leq (t+1) \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \alpha_1, \mathbf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\mathbf{aux}, \rho) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}(\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})) \end{array} \right] \\
 & \leq (t+1) \cdot \kappa_{\text{SP}} \left(n, \delta'_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}} (s, n) \right).
 \end{aligned}$$

□

Part III

NARGs Based on IPs

Overview

We introduce public-coin interactive proofs (public-coin IPs), a type of probabilistic proof that generalizes sigma protocols (SPs) to multiple rounds. We show how to use a hash function (the random oracle) to transform any public-coin IP into a corresponding non-interactive argument (NARG), thereby “removing” the interaction from the public-coin IP. This builds on ideas from the special case of SPs in Part II. A key difference is that, in order for the transformation from a public-coin IP to a NARG to be secure, the public-coin IP must satisfy a stronger notion of soundness, known as state-restoration soundness.

13 Interactive proofs	111
13.1 Definition	111
13.2 State restoration	114
14 The Fiat–Shamir transformation for IPs	121
14.1 Construction	121
14.2 Lower bound on the soundness error	123
14.3 Soundness	126
14.4 Knowledge soundness	128
15 Optimization: a faster variant	131
15.1 Construction	131
15.2 Soundness	134
16 Additional security definitions	144
16.1 Knowledge soundness	144
16.2 Zero knowledge	146
16.3 A security reduction from state-restoration	148

13 Interactive proofs

We introduce notations and definitions for *interactive proofs* (IPs), which are multi-round protocols between a prover and a verifier. This generalizes the notion of SPs introduced in Chapter 8.

In subsequent chapters we study how to construct non-interactive arguments from public-coin IPs. This is known as the *Fiat–Shamir transformation* [FS86], and builds and extends ideas introduced for the case of SPs.

13.1 Definition

An interactive proof (IP) is a tuple of algorithms

$$\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$$

that works as follows. The IP prover \mathbf{P}_{IP} receives as input an instance \mathbf{x} and a witness w , and the IP verifier \mathbf{V}_{IP} receives as input the instance \mathbf{x} . They interact over some number k of rounds, where in each round $i \in [k]$ the IP prover \mathbf{P}_{IP} sends a message α_i and then the IP verifier \mathbf{V}_{IP} sends a message ρ_i .¹ After the interaction, the IP verifier \mathbf{V}_{IP} outputs a bit denoting whether to accept or reject, computed based on the instance \mathbf{x} , the received prover messages $(\alpha_i)_{i \in [k]}$, and the IP verifier's own randomness (these form the entire view of the IP verifier). Both algorithms may be probabilistic. See Figure 13.1 for a diagram of an IP.

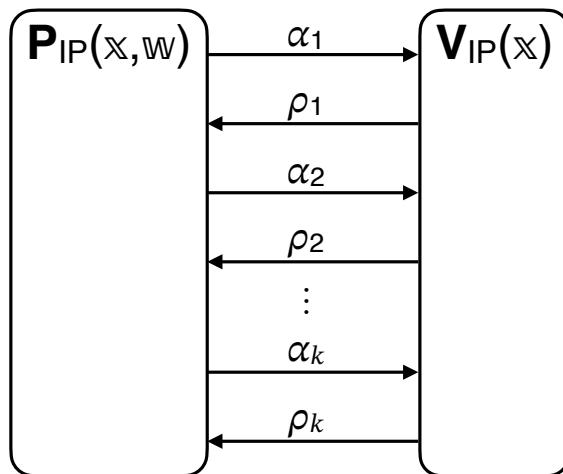


Figure 13.1: Diagram of an IP.

¹Alternatively, an IP can be defined so that in each round the IP verifier sends a message and then the IP prover sends a message. The convention that we use (the IP prover moves first) is consistent with the special case of SPs (where the SP prover moves first) and offers other notational benefits.

The tuple $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ is an IP for a relation \mathcal{R} with (*perfect*) *completeness* and *soundness error* ϵ_{IP} if it satisfies the two properties stated below.

Definition 13.1.1. $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ for a relation \mathcal{R} has **perfect completeness** if for every $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$

$$\Pr[\langle \mathbf{P}_{\text{IP}}(\mathbf{x}, \mathbf{w}), \mathbf{V}_{\text{IP}}(\mathbf{x}) \rangle_{\text{IP}} = 1] = 1.$$

Definition 13.1.2. $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ for a relation \mathcal{R} has **soundness error** ϵ_{IP} if for every $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ and malicious IP prover $\tilde{\mathbf{P}}_{\text{IP}}$

$$\Pr[\langle \tilde{\mathbf{P}}_{\text{IP}}, \mathbf{V}_{\text{IP}}(\mathbf{x}) \rangle_{\text{IP}} = 1] \leq \epsilon_{\text{IP}}(\mathbf{x}).$$

We additionally define $\epsilon_{\text{IP}}(n) := \max \{ \epsilon_{\text{IP}}(\mathbf{x}) \mid \mathbf{x} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \text{ and } \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \}$.

Above, given two interactive algorithms A and B , we use the notation $\langle A, B \rangle_{\text{IP}}$ to indicate the random variable that equals the output of B after interacting with A , and where the probability is taken over any randomness used by A or B .

See Figure 13.1 for a diagram.

Efficiency measures We are interested in several efficiency measures of an IP.

- *Round complexity*, denoted k , is the number of back-and-forth interactions between the IP prover and the IP verifier. Each round contains (at most) two messages: an IP prover message and an IP verifier message. If we wish to be more precise in describing the interaction (e.g., when an IP has small round complexity), we may instead specify the number of *messages* and whether the IP prover or IP verifier sends the first message in the interaction.
- *Communication complexity* refers to the total number of bits exchanged between the IP prover and IP verifier. In more detail, for every $i \in [k]$, we denote by pv_i and vp_i the size of the i -th IP prover message and the i -th IP verifier message. Therefore the size of the prover-to-verifier transcript is $\text{pv} = \sum_{i \in [k]} \text{pv}_i$ and the size of the verifier-to-prover transcript is $\text{vp} = \sum_{i \in [k]} \text{vp}_i$. In particular, the communication complexity is $\text{pv} + \text{vp}$.
- *Prover time and verifier time*, denoted pt and vt , are the running times of the IP prover and the IP verifier (across the whole interaction).

The above efficiency measures may be functions of the given instance \mathbf{x} (and possibly other parameters associated to the construction of an IP).

Public-coin IPs In this book we only consider IPs that are *public-coin*, which is a property of the IP verifier \mathbf{V}_{IP} .

Definition 13.1.3. $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ is **public-coin** if every message ρ_i sent by the IP verifier \mathbf{V}_{IP} is a random binary string of some prescribed size r_i (that is statistically independent of everything else); moreover, the IP verifier \mathbf{V}_{IP} has no other randomness. In this case, the decision bit of the IP verifier \mathbf{V}_{IP} is a function only of the instance \mathbf{x} and the transcript of interaction $(\alpha_1, \rho_1, \dots, \alpha_k, \rho_k)$. We denote this bit by

$$\mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}).$$

In this case the total randomness complexity is $r := \sum_{i \in [k]} r_i$, and the maximum randomness complexity in any round (which we sometimes use in an analysis) is $r_{\max} := \max_{i \in [k]} r_i$. Moreover, the verifier-to-prover communication complexity coincides with randomness complexity: for every $i \in [k]$, $\text{vp}_i = r_i$; and thus $\text{vp} = \sum_{i \in [k]} \text{vp}_i = \sum_{i \in [k]} r_i = r$.

Knowledge soundness The soundness notion (Definition 13.1.2) can be strengthened to a knowledge soundness notion, which informally means that whenever an IP prover convinces the IP verifier, then an efficient extractor finds a witness (up to some error). In more detail, we require the existence of a probabilistic algorithm \mathbf{E}_{IP} that works as follows. Fix any instance \mathbf{x} and IP prover $\tilde{\mathbf{P}}_{\text{IP}}$. The IP prover $\tilde{\mathbf{P}}_{\text{IP}}$ and IP verifier \mathbf{V}_{IP} (on input \mathbf{x}) interact; denote by $((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ the transcript of this interaction and by b the decision bit of the IP verifier \mathbf{V}_{IP} . The knowledge extractor \mathbf{E}_{IP} is tasked to find a witness \mathbf{w} when given as input the instance \mathbf{x} , the interaction transcript $((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$, and black-box access to the IP prover $\tilde{\mathbf{P}}_{\text{IP}}$. This means that the knowledge extractor \mathbf{E}_{IP} is *rewinding*: it can re-run the IP prover $\tilde{\mathbf{P}}_{\text{IP}}$ multiple times, possibly with correlated inputs. The knowledge soundness error is an upper bound on the probability that $b = 1$ ($\tilde{\mathbf{P}}_{\text{IP}}$ convinces \mathbf{V}_{IP}) and $(\mathbf{x}, \mathbf{w}) \notin \mathcal{R}$ (\mathbf{E}_{IP} does not find a witness). In general, the knowledge soundness error may be a function of the failure probability of $\tilde{\mathbf{P}}_{\text{IP}}$, which we define below.

Definition 13.1.4. Let $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ be an IP. A deterministic IP prover $\tilde{\mathbf{P}}_{\text{IP}}$ has **failure probability** $\delta_{\tilde{\mathbf{P}}_{\text{IP}}}$ if for every instance \mathbf{x} :

$$\Pr[\langle \tilde{\mathbf{P}}_{\text{IP}}, \mathbf{V}_{\text{IP}}(\mathbf{x}) \rangle_{\text{IP}} = 0] \leq \delta_{\tilde{\mathbf{P}}_{\text{IP}}}(\mathbf{x}).$$

Definition 13.1.5. $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ for a relation \mathcal{R} has **rewinding knowledge soundness error** κ_{IP} with **extraction time** et_{IP} if there exists a probabilistic algorithm \mathbf{E}_{IP} (the extractor) such that for every instance \mathbf{x} and deterministic IP prover $\tilde{\mathbf{P}}_{\text{IP}}$ with running time $\tau_{\tilde{\mathbf{P}}_{\text{IP}}}$ the following holds:

$$\Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} b \xleftarrow{((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})} \langle \tilde{\mathbf{P}}_{\text{IP}}, \mathbf{V}_{\text{IP}}(\mathbf{x}) \rangle_{\text{IP}} \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}}) \end{array} \right] \leq \kappa_{\text{IP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{IP}}}(\mathbf{x})).$$

Moreover, $\mathbf{E}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}})$ runs in expected time $\text{et}_{\text{IP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{IP}}}(\mathbf{x}), \tau_{\tilde{\mathbf{P}}_{\text{IP}}}(\mathbf{x}))$. We additionally define

$$\begin{aligned} \kappa_{\text{IP}}(n, \delta_{\tilde{\mathbf{P}}_{\text{IP}}}) &:= \max \{ \kappa_{\text{IP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{IP}}}(\mathbf{x})) \mid \mathbf{x} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \}, \quad \text{and} \\ \text{et}_{\text{IP}}(n, \delta_{\tilde{\mathbf{P}}_{\text{IP}}}, \tau_{\tilde{\mathbf{P}}_{\text{IP}}}) &:= \max \{ \text{et}_{\text{IP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{IP}}}(\mathbf{x}), \tau_{\tilde{\mathbf{P}}_{\text{IP}}}(\mathbf{x})) \mid \mathbf{x} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \}. \end{aligned}$$

If $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ has knowledge soundness error κ_{IP} (with any extraction time et_{IP}) then it has soundness error ϵ_{IP} such that $\epsilon_{\text{IP}}(\mathbf{x}) \leq \min_{\delta \in [0, 1]} \kappa_{\text{IP}}(\mathbf{x}, \delta)$: if $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ then the extractor cannot output a witness \mathbf{w} such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ (as none exists), in which case the event bounded by the knowledge soundness condition equals the event bounded by the soundness condition.

A notable special case of Definition 13.1.5 is *straightline extraction*: \mathbf{E}_{IP} is a polynomial-time deterministic algorithm that receives as input the instance \mathbf{x} and IP prover messages $(\alpha_i)_{i \in [k]}$ (but not the IP verifier messages $(\rho_i)_{i \in [k]}$ or access to $\tilde{\mathbf{P}}_{\text{IP}}$).

Zero knowledge An IP is *zero knowledge* if the IP prover, when interacting with the IP verifier, reveals no information beyond the fact that the proved statement is true. If the IP verifier is allowed to behave arbitrarily, then the property is called *malicious-verifier* zero knowledge; otherwise, if the IP verifier must follow the honest strategy, then the property is called *honest-verifier* zero knowledge. This latter (and weaker) property suffices for the applications that we consider, and so our definitions focus on that property only. The formal definition consists of two steps: (a) we formalize the *view* of the IP verifier, which is all the information that the IP verifier learns by interacting with the IP prover; (b) we say that the interactive proof is zero knowledge if the view of the IP verifier can be (approximately) simulated by a polynomial-time probabilistic algorithm, known as the *simulator*. The intuition here is that anything that is computable efficiently with randomness is “for free”, and so does not count as revealed knowledge. Crucially, the property is only required for instance-witness pairs in the relation, and the simulator is only given as input the instance (but not the witness).

Definition 13.1.6. *The IP verifier’s view in $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ on the instance-witness pair (\mathbf{x}, \mathbf{w}) , denoted $\text{View}_{\text{IP}}(\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}}, \mathbf{x}, \mathbf{w})$, is the random variable $(\mathbf{x}, \rho, (\alpha_i)_{i \in [k]})$ where:*

- ρ is a random choice of randomness for the IP verifier \mathbf{V}_{IP} ; and
- $(\alpha_i)_{i \in [k]}$ are the prover messages received in an interaction between $\mathbf{P}_{\text{IP}}(\mathbf{x}, \mathbf{w})$ and $\mathbf{V}_{\text{IP}}(\mathbf{x}, \rho)$.

Note that the honest IP prover $\mathbf{P}_{\text{IP}}(\mathbf{x}, \mathbf{w})$ may use its own private randomness (and is not part of the IP verifier’s view). If the IP is public-coin then the view shows each round’s randomness: $(\mathbf{x}, (\rho_i)_{i \in [k]}, (\alpha_i)_{i \in [k]})$.

Definition 13.1.7. $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ for a relation \mathcal{R} has **honest-verifier zero-knowledge error** z_{IP} if there exists a polynomial-time probabilistic algorithm \mathbf{S}_{IP} such that for every instance-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ the following random variables are $z_{\text{IP}}(\mathbf{x})$ -close in statistical distance:

$$\text{View}_{\text{IP}}(\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}}, \mathbf{x}, \mathbf{w}) \text{ and } \mathbf{S}_{\text{IP}}(\mathbf{x}).$$

We additionally define $z_{\text{IP}}(n) := \max \{z_{\text{IP}}(\mathbf{x}) \mid \mathbf{x}, \mathbf{w} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \text{ and } (\mathbf{x}, \mathbf{w}) \in \mathcal{R}\}$.

13.2 State restoration

We describe *state-restoration soundness*, a notion of soundness for public-coin IPs. This notion requires security against a more powerful class of malicious provers called *state-restoration provers*.

In Section 13.2.1 we define state-restoration soundness for IPs. In Section 13.2.2 we compare state-restoration soundness and standard soundness for IPs.

The security of a non-interactive argument obtained from a given IP via the Fiat–Shamir transformation is characterized by the state-restoration soundness of the IP, as we discuss in Chapter 14. However, in stark contrast to the case of state-restoration soundness for SPs (discussed in Chapter 12), good standard soundness for an IP does not, in general, imply good state-restoration soundness. (An IP can have small standard soundness error but huge state-restoration soundness error; see Remark 13.2.9.) Therefore, to ensure security of the Fiat–Shamir transformation, one must assume that the given IP has small state-restoration soundness error.

Small state-restoration soundness error is implied by natural strong notions of soundness, such as special soundness (see Chapter 30) and round-by-round soundness (see Chapter 31).

Analogous considerations hold for state-restoration *knowledge* soundness versus standard *knowledge* soundness.

13.2.1 Definition

The standard notion of soundness for an IP considers the setting where a malicious IP prover attempts to convince the IP verifier in a single interaction (see Definition 13.1.2): in each round the malicious IP prover sends a message and the IP verifier responds with some randomness, and at the end of the interaction the IP verifier accepts or rejects based on (the instance and) the transcript of interaction. In contrast, state-restoration soundness considers the setting where a malicious IP prover attempts to convince the IP verifier multiple times across related interactions, and the malicious IP prover wins if it finds an interaction that convinces the IP verifier.

In more detail, we consider an *IP state-restoration game* that informally works as follows. The game samples k random functions to be used as IP verifier randomness: for every $i \in [k]$, the game samples a random function $\text{rnd}_i \in \mathcal{U}(\mathbf{r}_i)$ to choose IP verifier randomness corresponding to the i -th round. In this game the goal of the malicious IP prover is to output an instance \mathbf{x} , IP prover messages $(\alpha_1, \dots, \alpha_k)$, and salts $(\sigma_1, \dots, \sigma_k)$ that convince the IP verifier according to the following condition:

$$\mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 1 \quad \text{where} \quad \rho_i := \text{rnd}_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i)).$$

Each salt σ_i is a string of s bits, for some salt parameter $s \in \mathbb{N}$.

The malicious IP prover has a budget of $t \in \mathbb{N}$ moves to invest in choosing its output. A move consists of outputting a tuple $(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i))$ (possibly unrelated to the eventual output), and the game responds with the randomness $\rho_i := \text{rnd}_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i))$.

On one extreme, if the malicious IP prover makes no moves then it has no information about the random functions $(\text{rnd}_i)_{i \in [k]}$. This means that it chooses an output $(\mathbf{x}, (\alpha_1, \dots, \alpha_k), (\sigma_1, \dots, \sigma_k))$ without seeing any IP verifier random messages. This is a poor way to play the game.

The malicious IP prover can at minimum recreate a single interaction with the IP verifier: output a move $(\mathbf{x}, (\alpha_1), (\sigma_1))$ to obtain, as a response from the game, the IP verifier challenge $\rho_1 := \text{rnd}_1(\mathbf{x}, (\alpha_1), (\sigma_1))$; and then, for each $i \in \{2, \dots, k\}$, output a move $(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i))$ based on the prior randomness $\rho_1, \dots, \rho_{i-1}$ and then obtain, as a response from the game, the next round randomness $\rho_i := \text{rnd}_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i))$. This enables the malicious IP prover to win with probability that is at least that of winning in a single interaction with the IP verifier.

However, the malicious IP prover can do more than this: the malicious IP prover can use moves to see multiple “random continuations” of an interaction with the IP verifier. For example, the malicious IP prover can first output a move $(\mathbf{x}, (\alpha_1), (\sigma_1))$ to obtain the IP verifier challenge $\rho_1 := \text{rnd}_1(\mathbf{x}, (\alpha_1), (\sigma_1))$, and subsequently explore two different extensions of this transcript: output the moves $(\mathbf{x}, (\alpha_1, \alpha_2), (\sigma_1, \sigma_2))$ and $(\mathbf{x}, (\alpha_1, \alpha'_2), (\sigma_1, \sigma_2))$ for two different IP prover messages α_2 and α'_2 in order to obtain two different IP verifier challenges ρ_2 and ρ'_2 . And so on.

The salt parameter s limits how many times the malicious IP prover can extend the same partial transcript. For example, the malicious IP prover can see at most 2^s different choices of ρ_1 for the same instance \mathbf{x} and IP prover message α_1 . (Each s -bit salt leads to a fresh randomness choice.)

Definition 13.2.1. *The IP state-restoration game for $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ with salt size $s \in \mathbb{N}$, functions $\text{rnd} = (\text{rnd}_i)_{i \in [k]} \in \mathcal{U}((r_i)_{i \in [k]})$, and IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ is defined below.*

$\text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$:

1. Repeat the following until $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ decides to exit the loop.
 - a) $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ outputs $(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i))$, where \mathbf{x} is an instance, $(\alpha_1, \dots, \alpha_i)$ are IP prover messages, and $(\sigma_1, \dots, \sigma_i)$ are salt strings in $\{0, 1\}^s$.
 - b) Set $\rho_i := \text{rnd}_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i))$.
 - c) Send ρ_i to $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$.
2. $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ outputs $(\mathbf{x}, (\alpha_1, \dots, \alpha_k), (\sigma_1, \dots, \sigma_k))$, where \mathbf{x} is an instance, $(\alpha_1, \dots, \alpha_k)$ are IP prover messages, and $(\sigma_1, \dots, \sigma_k)$ are salt strings in $\{0, 1\}^s$.
3. For every $i \in [k]$, set $\rho_i := \text{rnd}_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i))$.
4. Output $(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$.

We denote by tr^{sr} the list of move-response pairs of the form $((\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i)), \rho_i)$ performed in the loop. We show tr^{sr} in an execution of the IP state-restoration game using the following notation:

$$(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}).$$

We say that $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ is **t-move** if $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ exits the loop after at most t iterations.

The above game directly leads to the notion of state-restoration soundness error: it is an upper bound on the probability that any IP prover in the IP state-restoration game can find an instance not in the language and an accepting transcript for it.

Definition 13.2.2. $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ has **state-restoration soundness error** $\epsilon_{\text{IP}}^{\text{sr}}$ if for every salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move malicious IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$, and instance size bound $n \in \mathbb{N}$:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 1 \end{array} \middle| \begin{array}{l} \text{rnd} = (\text{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \leftarrow \\ \text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \end{array} \right] \leq \epsilon_{\text{IP}}^{\text{sr}}(s, t, n).$$

We also define the notion of state-restoration *knowledge soundness* error: it is an upper bound on the probability that any IP prover in the IP state-restoration game can find an instance and an accepting transcript for it such that an extractor algorithm cannot find a witness for the instance (given the relevant information about the malicious prover and its moves in the game).

In more detail, we consider two flavors of state-restoration knowledge soundness: **straightline** knowledge soundness, and a relaxation known as **rewinding** knowledge soundness.²

The straightline variant of state-restoration knowledge soundness considers a (deterministic) knowledge extractor $\mathbf{E}_{\text{IP}}^{\text{sr}}$ that is tasked with finding a witness w while given the final output $(\mathbf{x}, (\alpha_1, \dots, \alpha_k), (\sigma_1, \dots, \sigma_k))$ of the state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ and its move-response trace tr^{sr} . The knowledge extractor $\mathbf{E}_{\text{IP}}^{\text{sr}}$ does not receive the randomness $(\rho_i)_{i \in [k]}$ used by the IP verifier.

²Many IPs of interest satisfy the relaxed notion of rewinding knowledge soundness rather than the stronger notion of straightline knowledge soundness. Nevertheless, we consider both for two reasons. First, the straightline flavor is simpler to understand than the rewinding flavor, so it plays a meaningful pedagogical unit. Second, in Section 26.1 we rely on (a relaxation of) the straightline variant as an intermediate notion when proving the knowledge soundness of Construction 25.1.1 as a composition of two transformations.

Definition 13.2.3. $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ has **straightline state-restoration knowledge soundness error** $\kappa_{\text{IP}}^{\text{sr}}$ if there exists a polynomial-time deterministic algorithm $\mathbf{E}_{\text{IP}}^{\text{sr}}$ (the extractor) such that for every salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move deterministic IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$, and instance size bound n :

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 1 \end{array} \middle| \begin{array}{l} \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \\ \text{Game}_{\text{IP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IP}}^{\text{sr}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, \mathbf{tr}^{\text{sr}}) \end{array} \right] \leq \kappa_{\text{IP}}^{\text{sr}}(s, t, n).$$

The rewinding variant of state-restoration knowledge soundness relaxes the prior notion by considering a knowledge extractor that additionally receives the randomness $(\rho_i)_{i \in [k]}$ used by the IP verifier and black-box access to the state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$. In this case, the error may additionally depend on the failure probability of $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ (an upper bound on the probability that the final output of $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ does *not* convince the IP verifier). Intuitively, as the failure probability increases, the error of extraction increases.

The motivation to consider the rewinding variant is that there are notable classes of IPs that satisfy this property. For example, in Chapter 30, we discuss how IPs that satisfy *special soundness* also satisfy rewinding state-restoration knowledge soundness.

Definition 13.2.4. Let $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ be an IP. A deterministic IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ has **failure probability** $\delta_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}}$ if for every salt size $s \in \mathbb{N}$ and instance size bound n :

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 0 \end{array} \middle| \begin{array}{l} \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \leftarrow \\ \text{Game}_{\text{IP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \end{array} \right] \leq \delta_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}}(s, n).$$

Definition 13.2.5. $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ has **rewinding state-restoration knowledge soundness error** $\kappa_{\text{IP}}^{\text{sr}}$ with **extraction time** $\mathbf{et}_{\text{IP}}^{\text{sr}}$ if there exists a probabilistic algorithm $\mathbf{E}_{\text{IP}}^{\text{sr}}$ (the extractor) such that for every salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move deterministic IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ with failure probability $\delta_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}}$ and running time $\tau_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}}$, and instance size bound n :

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 1 \end{array} \middle| \begin{array}{l} \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \\ \text{Game}_{\text{IP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IP}}^{\text{sr}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \mathbf{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \end{array} \right] \leq \kappa_{\text{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}}(s, n)).$$

Moreover, $\mathbf{E}_{\text{IP}}^{\text{sr}}$ runs in expected time $\mathbf{et}_{\text{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}}(s, n), \tau_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}}(s, n))$ (over the given inputs and internal randomness).

Remark 13.2.6 (non-adaptive choice of instance). The IP state-restoration soundness and knowledge soundness games naturally restrict to the setting where the instance \mathbf{x} is fixed in advance, rather than being chosen by the IP prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$. Here we consider an adaptive choice of instance \mathbf{x} to match the adaptive soundness and knowledge soundness notions that we consider for non-interactive arguments.

13.2.2 Comparison with standard soundness

The lemma below compares state-restoration soundness to (standard) soundness for public-coin IPs. The lower bound tells us that state-restoration soundness error is at least a multiplicative factor of $\Omega(\frac{t}{k})$ larger than standard soundness error. But it can be much larger than that, especially if the round complexity k of the IP is large. Both the upper bound and the lower bound in the lemma are right up to small-order terms, as explained in the corresponding remarks below.

Lemma 13.2.7. *Let $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ be a public-coin IP with round complexity k and soundness error ϵ_{IP} . For every $s, t, n \in \mathbb{N}$ with $s \geq \log \frac{t}{k}$, IP has state-restoration soundness error $\epsilon_{\text{IP}}^{\text{sr}}$ such that*

$$\left\lfloor \frac{t}{k} \right\rfloor \cdot \epsilon_{\text{IP}}(n) - \binom{\lfloor \frac{t}{k} \rfloor}{2} \cdot \epsilon_{\text{IP}}(n)^2 \leq \epsilon_{\text{IP}}^{\text{sr}}(s, t, n) \leq \binom{t+k}{k} \cdot \epsilon_{\text{IP}}(n).$$

Proof of lower bound. We show that

$$\epsilon_{\text{IP}}^{\text{sr}}(s, t, n) \geq \left\lfloor \frac{t}{k} \right\rfloor \cdot \epsilon_{\text{IP}}(n) - \binom{\lfloor \frac{t}{k} \rfloor}{2} \cdot \epsilon_{\text{IP}}(n)^2.$$

We describe a “universal” state-restoration attack. Let $\tilde{\mathbf{P}}_{\text{IP}}$ be a malicious IP prover that convinces the IP verifier \mathbf{V}_{IP} with probability exactly ϵ_{IP} . Consider the IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ that, in the IP state-restoration game, runs $\tilde{\mathbf{P}}_{\text{IP}}$ for $\lfloor \frac{t}{k} \rfloor$ times, each time with a unique salt. (This is possible because $s \geq \log \frac{t}{k}$.) Then $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ outputs an accepting execution, if any exists. Each execution costs k moves, and convinces the IP verifier \mathbf{V}_{IP} with probability ϵ_{IP} (independently of any other executions). By the simple inclusion-exclusion principle (Lemma 1.2.3), the probability that at least one execution is accepting is at least the lower bound stated above. \square

Remark 13.2.8 (tightness of the lower bound). The lower bound above is tight up to small-order terms. In particular, there is a public-coin IP with round complexity k , soundness error ϵ_{IP} , and state-restoration soundness error $\epsilon_{\text{IP}}^{\text{sr}}$ such that $\epsilon_{\text{IP}}^{\text{sr}}(s, t, n) \leq t \cdot \epsilon_{\text{IP}}(n)$.

Consider the following IP for a trivial language \mathcal{L} : the IP verifier sends $\log 1/\epsilon_{\text{IP}}$ random bits as its first message, and accepts if and only if all bits are 0 or $\mathbf{x} \in \mathcal{L}$; pad the rest of the interaction with dummy rounds so to obtain a k -round public-coin protocol. The IP verifier accepts instances $\mathbf{x} \in \mathcal{L}$ with probability 1, and accepts instances $\mathbf{x} \notin \mathcal{L}$ with probability ϵ_{IP} . For this public-coin IP, the state-restoration soundness error is at most $t \cdot \epsilon_{\text{IP}}(n)$ (because an IP state-restoration prover has at most t attempts to obtain an all-zero message from the IP verifier).

Proof of upper bound. We show that

$$\epsilon_{\text{IP}}^{\text{sr}}(s, t, n) \leq \binom{t+k}{k} \cdot \epsilon_{\text{IP}}(n).$$

Let $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ be an IP state-restoration prover that wins the IP state-restoration game $\text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$ with probability δ . Assume, without loss of generality, that $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ does not make duplicate moves. Moreover, assume that the final output $(\mathbf{x}, (\alpha_1, \dots, \alpha_k), (\sigma_1, \dots, \sigma_k))$ of $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ is such that the moves $(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i))$ for all $i \in [k]$ have been previously performed; this can be ensured by increasing the move budget from t to $t + k$. We construct a malicious IP prover $\tilde{\mathbf{P}}_{\text{IP}}$ that convinces the IP verifier \mathbf{V}_{IP} with probability at least $\binom{t+k}{k}^{-1} \cdot \delta$ (without any state restoration).

$\tilde{\mathbf{P}}_{\text{IP}}$:

1. Sample a random subset $S \subseteq [t+k]$ of cardinality k .
2. For every $i \in [k]$, denote by $S[i]$ the i -th smallest value in S .
3. For every $i \in [k]$, lazily sample an oracle $\text{rnd}_i \leftarrow \mathcal{U}(r_i)$.
4. Simulate an execution of the IP state-restoration game with $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$, where the j -th move is answered as follows:
 - a) Let $i \in [k]$ be such that the move has the form $(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i))$.
 - b) If $j = S[i]$: send the prover message α_i to the IP verifier, receive the random message ρ_i , and answer the move with ρ_i .
 - c) If $j \neq S[i]$: answer the move with $\text{rnd}_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i))$.

The IP state-restoration game allows $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ to attempt to convince the IP verifier across multiple related interactions, with each round of the game corresponds to a random extension for a chosen partial transcript (that might or might not exist as a previous move). When $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ ultimately chooses a transcript of interaction, we can associate distinct indices $i_1, \dots, i_k \subseteq [t+k]$ such that the randomness used by the IP verifier in round j is the randomness returned to $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ in round i_j of the game. In other words, the transcript of interaction chosen by $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ appears as a subset of its $t+k$ moves. Hence we can construct an IP prover $\tilde{\mathbf{P}}_{\text{IP}}$ from $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ by guessing this subset, playing against the real IP verifier on this subset, and simulating the rest of the game. \square

Remark 13.2.9 (tightness of the upper bound). The upper bound above is tight up to small-order terms. In particular, there is a public-coin IP with round complexity k , soundness error ϵ_{IP} , and state-restoration soundness error $\epsilon_{\text{IP}}^{\text{sr}}$ such that $\epsilon_{\text{IP}}^{\text{sr}}(s, t, n) \geq \binom{t/6}{k} \cdot \epsilon_{\text{IP}}(n)$.

Let \mathcal{L} be a language decidable in polynomial time, and consider the following k -round IP for \mathcal{L} . The IP verifier sends $\ell := \frac{\log 1/\epsilon_{\text{IP}}}{k}$ random bits in each round, and accepts an instance \mathbf{x} if and only if $\mathbf{x} \in \mathcal{L}$ (by using the language's decider) or all random bits in every round are zero. The soundness error of this IP is ϵ_{IP} .

Consider an IP state-restoration prover that makes t/k moves per round, attempting to obtain random bits for this round that are all zero. If $t/k \geq 2^\ell$ then this attack succeeds with probability one. If instead $t/k < 2^\ell$, then we can use the inclusion-exclusion principle (Lemma 1.2.3). For any $i \in [t/k]$, let E_i be the event that the i -th move is answered with ℓ zeros. Then, the probability of succeeding in a particular round is at least

$$\begin{aligned} \Pr[\vee_{i \in [t/k]} E_i] &\geq \sum_{i \in [t/k]} \Pr[E_i] - \sum_{\substack{i, j \in [t/k] \\ \text{with } i \neq j}} \Pr[E_i \wedge E_j] \\ &= \frac{t}{k} \cdot \frac{1}{2^\ell} - \binom{t/k}{2} \cdot \frac{1}{2^{2\ell}} \\ &\geq \frac{t}{k} \cdot \frac{1}{2^\ell} - \frac{(t/k)^2}{2} \cdot \frac{1}{2^{2\ell}} \\ &\geq \frac{t}{k} \cdot \frac{1}{2^\ell} - \frac{t/k}{2} \cdot \frac{1}{2^\ell} \quad (\text{since } t/k < 2^\ell) \\ &= \frac{t}{2k} \cdot \frac{1}{2^\ell}. \end{aligned}$$

We conclude that

$$\epsilon_{\text{IP}}^{\text{sr}}(s, t, n) \geq \left(\frac{t}{2k} \cdot \frac{1}{2^\ell} \right)^k$$

$$\begin{aligned} &\geq \left(\frac{e \cdot t/6}{k} \right)^k \cdot \frac{1}{2^{k \cdot \ell}} \quad (\text{since } 1/2 > e/6) \\ &\geq \binom{t/6}{k} \cdot \epsilon_{IP}(n). \quad (\text{by Lemma 1.4.1}) \end{aligned}$$

14 The Fiat–Shamir transformation for IPs

We describe how to transform any public-coin interactive proof (IP) into a corresponding non-interactive argument, via a construction that is known as the *Fiat–Shamir transformation* [FS86].

The key idea of the construction is to use the random oracle to emulate an interaction between the IP prover and IP verifier, and set the argument string to include the IP prover messages of the interaction. This builds on the case of SPs (a special case of public-coin IPs) discussed in Part II. Similarly to the case of SPs, the construction’s only goal is to “remove the interaction” and, in particular, results in an argument string that is at least as large as the IP prover-to-verifier communication complexity. Hence the resulting non-interactive argument is not succinct.

Organization In Section 14.1 we describe the construction. In Section 14.2 we provide a lower bound on the soundness error. In Section 14.3 we provide an upper bound on the soundness error. In Section 14.4 we provide an upper bound on the knowledge soundness error.

14.1 Construction

We describe how to transform any public-coin IP into a non-interactive argument. The high-level idea is that the argument prover can emulate an interaction of the IP prover and the IP verifier by using the random oracle to derive the IP verifier challenges, and subsequently send an argument string that contains the IP prover messages corresponding to this interaction (which the argument verifier can check by re-deriving the IP verifier challenges). Turning this high-level idea into a secure construction involves delicate technical details, which we discuss below.

Intuition Let $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ be a public-coin IP with round complexity k . We wish to construct a non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ where the argument prover \mathcal{P} uses the random oracle to emulate an interaction between the IP prover and IP verifier, and the argument verifier \mathcal{V} uses the random oracle to reconstruct this interaction and uses the IP verifier to check the interaction.

The argument prover \mathcal{P} derives the first IP verifier challenge ρ_1 analogously to the case of an SP in Construction 10.1.1: \mathcal{P} uses the IP prover \mathbf{P}_{IP} to compute the first IP prover message α_1 , and then sets the first IP verifier challenge to be $\rho_1 := f(\mathbf{x}, \alpha_1)$.¹ Recall from Section 9.6.1 that the instance \mathbf{x} appears in the query (\mathbf{x}, α_1) in order to ensure adaptive security, intuitively ensuring that a malicious argument prover cannot choose the instance \mathbf{x} after seeing the query answer ρ_1 .

But how should the argument prover \mathcal{P} derive subsequent IP verifier challenges ρ_i with $i > 1$? (This question did not arise in the case of an SP, where there is a single IP verifier challenge.)

¹We discuss the adjustment of output sizes to match the length of IP verifier challenges later on.

An idea is to set $\rho_i := f(\mathbf{x}, \alpha_i)$ where α_i is the i -th IP prover message.

However this does not “enforce” the chronological order of interaction: a malicious argument prover could compute the second IP verifier challenge $\rho_2 := f(\mathbf{x}, \alpha_2)$ for a specially-chosen message α_2 , and only after that compute the first IP verifier challenge $\rho_1 := f(\mathbf{x}, \alpha_1)$ for a specially-chosen message α_1 that depends on ρ_2 , which could lead to an attack against the non-interactive argument. For example, suppose that the IP verifier accepts if and only if $\alpha_1 = \rho_2$ (the IP prover first message equals the IP verifier second challenge). While this event occurs with a small probability in an interaction between an IP prover and the IP verifier (because the IP prover sends its first message before the IP verifier second challenge is sampled and sent), a malicious argument prover can make this event occur always if IP verifier challenges are derived as $\rho_i := f(\mathbf{x}, \alpha_i)$.

A natural solution, which we will show is secure, is to derive IP verifier challenges by including *all* IP prover messages so far in the query to the random oracle:

$$\rho_i := f(\mathbf{x}, \alpha_1, \dots, \alpha_i).$$

Intuitively, this ensures that the argument prover “commits” to the IP prover messages $\alpha_1, \dots, \alpha_i$ before seeing the i -th IP verifier challenge ρ_i , which chronologically comes after these messages.

Additional considerations Analogously to the case of SPs in Construction 10.1.1, deriving IP verifier challenges requires additional care.

- *Salts.* Each query to the random oracle should be unpredictable (so that we can prove adaptive zero knowledge), and so each query includes a fresh salt. In more detail, the argument prover includes a random salt τ_i in the query to derive the i -th IP verifier challenge. Again to enforce a precise chronological order, each query additionally includes all prior salts. Overall, for every $i \in [k]$, the query to derive the i -th IP verifier challenge is as follows:

$$\rho_i := f(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i)).$$

- *Output sizes.* The i -th IP verifier challenge consists of r_i bits, so we use a random oracle $f_i \in \mathcal{U}(r_i)$ to derive this challenge. Overall, given random oracles $(f_i)_{i \in [k]} \in \mathcal{U}((r_i)_{i \in [k]})$, for every $i \in [k]$, the i -th IP verifier challenge is derived as follows:

$$\rho_i := \begin{cases} f_1(\mathbf{x}, \alpha_1, \tau_1) & \text{if } i = 1 \\ f_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i)) & \text{if } i > 1 \end{cases}.$$

Final construction The above considerations lead to the following construction.

Construction 14.1.1: Fiat–Shamir transformation for public-coin IPs

Let $\mathbf{IP} = (\mathbf{P}_{\mathbf{IP}}, \mathbf{V}_{\mathbf{IP}})$ be a public-coin IP with round complexity k . Let $\lambda \in \mathbb{N}$ be a security parameter and $s \in \mathbb{N}$ be a privacy parameter.

We define $\mathbf{NARG} := \mathbf{FS}_{\mathbf{IP}}[\mathbf{IP}, \lambda, s]$ to be the non-interactive argument $\mathbf{NARG} = (\mathcal{P}, \mathcal{V})$ constructed as follows. The argument prover \mathcal{P} receives as input an instance \mathbf{x} and witness \mathbf{w} , and the argument verifier \mathcal{V} receives as input the instance \mathbf{x} and an argument

string π . Both receive query access to random oracles $(f_i)_{i \in [k]} \in \mathcal{U}((r_i)_{i \in [k]})$; this implies that the oracle configuration cnf is defined as $\text{cnf}(\lambda, n) := (r_i)_{i \in [k]}$ (see Definition 7.1.1).

- $\mathcal{P}^f(\mathbf{x}, \mathbf{w})$:

1. For $i = 1, \dots, k$:
 - Compute the i -th message (and auxiliary state) of the IP prover:

$$(\alpha_i, \text{aux}_i) := \begin{cases} \mathbf{P}_{\text{IP}}(\mathbf{x}, \mathbf{w}) & \text{if } i = 1 \\ \mathbf{P}_{\text{IP}}(\text{aux}_{i-1}, \rho_{i-1}) & \text{if } i > 1 \end{cases}.$$

2. Sample a random salt $\tau_i \in \{0, 1\}^s$.
- c) Derive the i -th random message of the IP verifier:

$$\rho_i := \begin{cases} f_1(\mathbf{x}, \alpha_1, \tau_1) & \text{if } i = 1 \\ f_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i)) & \text{if } i > 1 \end{cases}.$$

2. Output the argument string $\pi := ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]})$.

- $\mathcal{V}^f(\mathbf{x}, \pi)$:

1. Parse the argument string π as a tuple $((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]})$.
2. For $i = 1, \dots, k$:
 - derive the i -th IP verifier message ρ_i as in Item 1c of the argument prover \mathcal{P} .
3. Check that the IP verifier accepts: $\mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 1$.

14.2 Lower bound on the soundness error

We show that every state-restoration attack on a public-coin IP translates into a corresponding attack on the resulting non-interactive argument (obtained via Construction 14.1.1). Hence, a small state-restoration soundness error for the public-coin IP is *necessary* for a small soundness error for the resulting non-interactive argument.² Later in Section 14.3 we prove that a small state-restoration soundness error is also *sufficient*.

The intuition is similar to the lower bound in Section 9.2 for the Fiat–Shamir transformation applied to SPs. Namely, while a malicious IP prover has a single chance to convince the IP verifier, a malicious argument prover for Construction 14.1.1 has *multiple* chances to convince the IP verifier that underlies the argument verifier. Hence the soundness error of the non-interactive argument is greater than the soundness error of the IP.

In more detail, the malicious argument prover can “explore” different partial transcripts of interaction by querying the random oracle with different IP prover messages and salts, and eventually output an argument string $\pi = ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]})$ based on this information. For example, the malicious argument prover can first query the random oracle at $(\mathbf{x}, \alpha_1, \tau_1)$ to

²Having small soundness error does not imply having small state-restoration soundness error; see Remark 13.2.9.

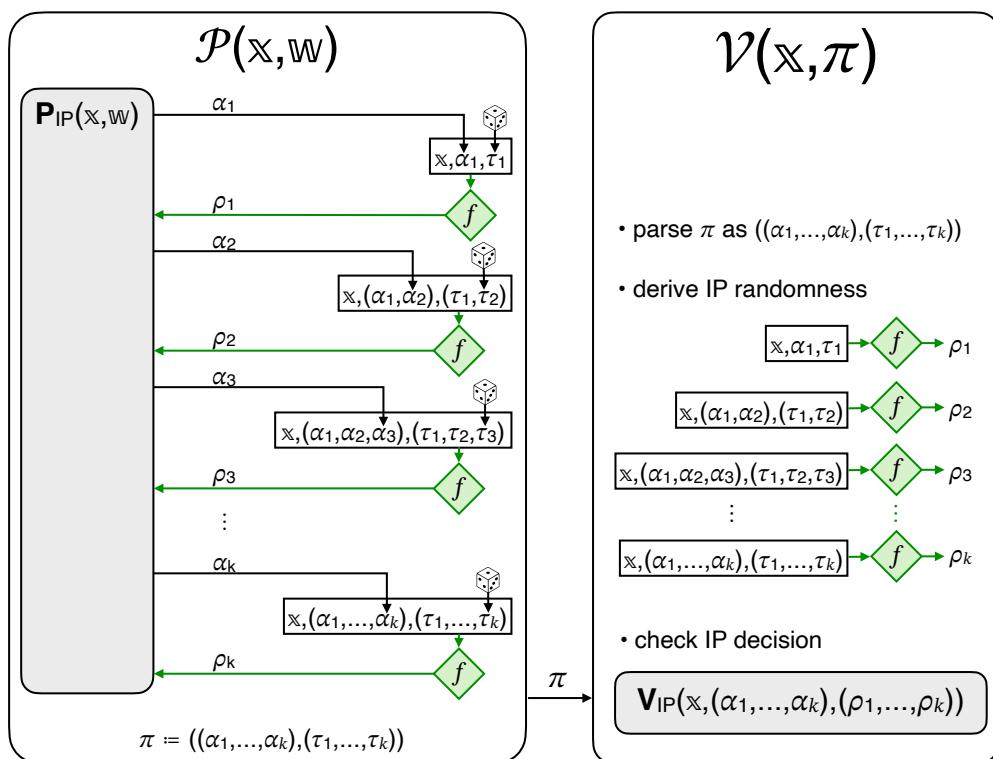


Figure 14.1: Diagram of the Fiat–Shamir transformation for public-coin IPs (\mathbf{FS}_{IP} in Construction 14.1.1).

obtain the IP verifier challenge ρ_1 , and subsequently explore two different extensions of this transcript: query the random oracle at $(\mathbf{x}, (\alpha_1, \alpha_2), (\tau_1, \tau_2))$ and $(\mathbf{x}, (\alpha_1, \alpha'_2), (\tau_1, \tau_2))$ for two different IP prover messages α_2 and α'_2 in order to obtain two different IP verifier challenges ρ_2 and ρ'_2 . And so on. The malicious argument prover can also try its luck by outputting an argument string $\pi = ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]})$ for which it has not seen some of the corresponding randomness.

This type of exploration is exactly what is modeled by the IP state-restoration game in Definition 13.2.1. Indeed, we show that every IP state-restoration strategy can be directly translated into a strategy to convince the argument verifier in Construction 14.1.1 with (at least) the same convincing probability. In particular, the soundness error of the non-interactive argument is lower bounded by the underlying IP's state-restoration soundness error.

Lemma 14.2.1. *Let $\mathbf{IP} = (\mathbf{P}_{\mathbf{IP}}, \mathbf{V}_{\mathbf{IP}})$ be a public-coin IP for a relation \mathcal{R} with state-restoration soundness error exactly $\epsilon_{\mathbf{IP}}^{\text{sr}}$. For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{NARG} := \mathbf{FS}_{\mathbf{IP}}[\mathbf{IP}, \lambda, s]$ in Construction 14.1.1 is a non-interactive argument for \mathcal{R} with adaptive soundness error ϵ_{ARG} (see Definition 7.1.4) such that for every query bound $t \in \mathbb{N}$ and instance size bound $n \in \mathbb{N}$,*

$$\epsilon_{\text{ARG}}(\lambda, t, n) \geq \epsilon_{\mathbf{IP}}^{\text{sr}}(s, t, n).$$

Construction 14.2.2. Let $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$ be a t -move IP state-restoration prover for $\mathbf{IP} = (\mathbf{P}_{\mathbf{IP}}, \mathbf{V}_{\mathbf{IP}})$ that succeeds with probability $\epsilon_{\mathbf{IP}}^{\text{sr}}(s, t, n)$. We construct a t -query argument prover $\tilde{\mathcal{P}}$ for $\text{NARG} := \mathbf{FS}_{\mathbf{IP}}[\mathbf{IP}, \lambda, s]$ that, given query access to random oracles $(f_i)_{i \in [k]} \in \mathcal{U}((\mathbf{r}_i)_{i \in [k]})$ and black-box access to $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$, works as follows.

1. Repeat the following t times (or until $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$ decides to exit the loop):
 - a) $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$ outputs a move $(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i))$;
 - b) answer with $f_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i))$.
2. $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$ outputs $(\mathbf{x}, (\alpha_1, \dots, \alpha_k), (\sigma_1, \dots, \sigma_k))$.
3. For every $i \in [k]$, set $\tau_i := \sigma_i$.
4. Set the argument string $\pi := ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]})$.
5. Output (\mathbf{x}, π) .

In other words, the argument prover $\tilde{\mathcal{P}}$ emulates the IP state-restoration attack performed by $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$, choosing randomness of the IP state-restoration game according to the random oracles $(f_i)_{i \in [k]}$.

Proof. The lemma directly follows from the observation that $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$ wins in the IP state-restoration game if and only if $\tilde{\mathcal{P}}$ convinces the argument verifier, whenever the randomness choices are the same in both cases. (Randomness consists of a sample from $\mathcal{U}((\mathbf{r}_i)_{i \in [k]})$.) We conclude that

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| \begin{array}{l} f = (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f(\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}) \end{array} \right] \\ &= \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\mathbf{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 1 \end{array} \middle| \begin{array}{l} \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \leftarrow \text{Game}_{\mathbf{IP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}) \end{array} \right]. \end{aligned}$$

□

14.3 Soundness

We prove that Construction 14.1.1 is adaptively sound. The following theorem states that the adaptive soundness error of the non-interactive argument is upper bounded by the IP state-restoration soundness error of the underlying IP. In particular, a small state-restoration soundness error for the IP is *sufficient* for a small adaptive soundness error for the resulting non-interactive argument.

Theorem 14.3.1

Let IP be a public-coin IP for a relation \mathcal{R} with state-restoration soundness error $\epsilon_{\text{IP}}^{\text{sr}}$. For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{NARG} := \mathbf{FS}_{\text{IP}}[\text{IP}, \lambda, s]$ in Construction 14.1.1 is a non-interactive argument for \mathcal{R} with adaptive soundness error ϵ_{ARG} (see Definition 7.1.4) such that

$$\epsilon_{\text{ARG}}(\lambda, t, n) \leq \epsilon_{\text{IP}}^{\text{sr}}(s, t, n).$$

Proof. Fix a query bound $t \in \mathbb{N}$, t -query argument prover $\tilde{\mathcal{P}}$, and instance size bound $n \in \mathbb{N}$. In Construction 14.3.2 we describe an IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ that, using $\tilde{\mathcal{P}}$ as a black box, wins the IP state-restoration soundness game with (at least) the same probability as $\tilde{\mathcal{P}}$ convinces the argument verifier \mathcal{V} . Specifically, Lemma 14.3.4 upper bounds the winning probability of the argument prover $\tilde{\mathcal{P}}$ by the winning probability of the IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$. \square

Construction 14.3.2. The IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ is parametrized by the security parameter λ and instance size bound n and receives an argument prover $\tilde{\mathcal{P}}$ as a black box.

1. For every $i \in [k]$, lazily sample an oracle $\dot{g}_i \leftarrow \mathcal{U}(r_i)$ (to answer malformed queries).
2. Simulate $\tilde{\mathcal{P}}$ while answering a query x to f_i as follows:
 - a) If x can be parsed as a tuple $(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i))$:
 - i. Output the move $(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i))$ to the IP state-restoration game.
 - ii. Receive the response ρ_i from the IP state-restoration game.
 - b) Otherwise set $\rho_i := \dot{g}_i(x)$.
 - c) Return ρ_i as the answer to the query.
3. The simulation of $\tilde{\mathcal{P}}$ ends with an output (\mathbf{x}, π) , where $\pi = ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]})$.
4. For every $i \in [k]$, set the salt string $\sigma_i := \tau_i$.
5. The final output is $(\mathbf{x}, (\alpha_1, \dots, \alpha_k), (\sigma_1, \dots, \sigma_k))$.

The IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ makes at most t moves in the IP state-restoration game, since $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ makes at most one move for each query of $\tilde{\mathcal{P}}$ (which makes at most t queries). Moreover the running time of $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ is at most the running time of $\tilde{\mathcal{P}}$ plus $O(r_{\max} \cdot t)$.

Construction 14.3.3. The function FStoSR receives as input a query-answer trace tr and outputs a move-response trace tr^{sr} computed as follows.

$\text{FStoSR}(\text{tr})$:

1. Initialize an empty list of move-response pairs tr^{sr} .
2. For each query-answer (x, y) in tr (taken in order): if x is a query to f_i and can be parsed as a tuple $(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i))$, append (x, y) to tr^{sr} .
3. Output tr^{sr} .

If $\tilde{\mathcal{P}}$ has query-answer trace tr then $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}})$ has a move-response trace tr^{sr} (in the IP state-restoration game) that is output by the procedure FStoSR . The running time of FStoSR is $O(r_{\max} \cdot t)$, as processing each of at most t query-answer pairs takes time $O(r_{\max})$.

Lemma 14.3.4. $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ in Construction 14.3.2 is such that

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| \begin{array}{l} f = (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ \leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 1 \end{array} \middle| \begin{array}{l} \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \leftarrow \\ \text{Game}_{\text{IP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}})) \end{array} \right].$$

Moreover, if the running time of $\tilde{\mathcal{P}}$ is $\mathbf{t}_{\mathcal{P}}$ then the running time of $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ is $\mathbf{t}_{\mathcal{P}} + O(r_{\max} \cdot t)$.

Proof. It suffices to prove the inequality for an arbitrary choice of randomness for $\tilde{\mathcal{P}}$ (in case $\tilde{\mathcal{P}}$ is not deterministic), so fix such a choice of randomness. The proof of the lemma follows immediately from the following equivalence of distributions:

$$\left\{ \begin{array}{l} (tr^{\text{sr}}, (\rho_i)_{i \in [k]}, \mathbf{x}, \pi, b) \\ \left| \begin{array}{l} f = (f_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}^f(\mathbf{x}, \pi) \\ \text{tr}^{\text{sr}} := \text{FStoSR}(\text{tr}) \\ \text{parse } \text{tr}_v \text{ as } (x_i, \rho_i)_{i \in [k]} \end{array} \right. \end{array} \right\} \quad (14.1) \\ \equiv \left\{ \begin{array}{l} (tr^{\text{sr}}, (\rho_i)_{i \in [k]}, \mathbf{x}, \pi, b) \\ \left| \begin{array}{l} \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \text{Game}_{\text{IP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}})) \\ (\tau_i)_{i \in [k]} := (\sigma_i)_{i \in [k]} \\ \pi := ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]}) \\ b \leftarrow \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \end{array} \right. \end{array} \right\}.$$

We argue that $(tr^{\text{sr}}, (\rho_i)_{i \in [k]}, \mathbf{x}, \pi, b)$ is identically distributed in both experiments.

The cheating argument prover $\tilde{\mathcal{P}}$ receives the same distribution of query answers in both sides: in the left-side experiment $\tilde{\mathcal{P}}$ receives answers from the random oracles $(f_i)_{i \in [k]}$; and in the right-side experiment $\tilde{\mathcal{P}}$ receives answers either from the IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ (which in turn receives answers from the IP state restoration game with randomness $(\mathbf{rnd}_i)_{i \in [k]}$) or from the auxiliary oracles $(g_i)_{i \in [k]}$ (which are lazily sampled). In both cases, unique queries get independent random answers, and duplicate queries are answered consistently.

Hence the output (\mathbf{x}, π) of $\tilde{\mathcal{P}}$ in both experiments is equally distributed. (In the right-side experiment π is disassembled within the IP state-restoration game and reassembled outside of it.)

Next we discuss the decision bit b and the randomness $(\rho_i)_{i \in [k]}$ in both experiments.

- *Left-side experiment.* The bit b is $\mathcal{V}^f(\mathbf{x}, \pi)$. By construction of the argument verifier \mathcal{V} and recalling that $\pi = ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]})$, the bit b is $\mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ where $\rho_i := f_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i))$ for every $i \in [k]$.
- *Right-side experiment.* By definition of the IP state-restoration game, the bit b is equal to $\mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ where \mathbf{x} is the instance output by $\tilde{\mathcal{P}}$, $\pi = ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]})$ is the argument string output by $\tilde{\mathcal{P}}$, and $\rho_i := \text{rnd}_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i))$ for every $i \in [k]$.

Recalling that $f = (f_i)_{i \in [k]}$ and $\text{rnd} = (\text{rnd}_i)_{i \in [k]}$ have the same distribution in both experiments, we deduce that the decision bit b , and the random values $(\rho_i)_{i \in [k]}$ have the same distribution in both experiments.

Finally, $\text{FStoSR}(\text{tr})$ removes queries in tr that are not well-formed and $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ forwards (and only forwards) well-formed queries of $\tilde{\mathcal{P}}$ to the IP state-restoration game, so $\text{FStoSR}(\text{tr})$ in the left-side experiment and tr^{sr} in the right-side experiment have the same distribution. \square

Remark 14.3.5 (the case of SPs). An SP is a special case of a public-coin IP where the verifier sends a single random challenge and, in particular, the notions of state restoration for SPs in Section 12.1 are special cases of the notions of state restoration for IPs in Section 13.2.1. Moreover, Construction 10.1.1 (NARG from an SP) is a special case of Construction 14.1.1 (NARG from an IP) and, correspondingly, Lemma 12.2.1 ($\epsilon_{\text{ARG}}(\lambda, t, n) \leq \epsilon_{\text{SP}}^{\text{sr}}(s, t, n)$) is a special case of Theorem 14.3.1 ($\epsilon_{\text{ARG}}(\lambda, t, n) \leq \epsilon_{\text{IP}}^{\text{sr}}(s, t, n)$). Finally, Lemma 12.3.1 tells us that the SP state-restoration soundness error for an SP is at most $t + 1$ times the soundness error of the SP: $\epsilon_{\text{SP}}^{\text{sr}}(s, t, n) \leq (t + 1) \cdot \epsilon_{\text{SP}}(n)$. Hence, if we apply Theorem 14.3.1 to an SP then we get that the adaptive soundness error of the resulting non-interactive argument is upper bounded as follows:

$$\epsilon_{\text{ARG}}(\lambda, t, n) \leq \epsilon_{\text{IP}}^{\text{sr}}(s, t, n) = \epsilon_{\text{SP}}^{\text{sr}}(s, t, n) \leq (t + 1) \cdot \epsilon_{\text{SP}}(n).$$

Thus we recover the same upper bound as in Theorem 10.2.1 for Construction 10.1.1.

A similar discussion holds for knowledge soundness rather than soundness.

Remark 14.3.6 (on adaptive security). In Construction 14.1.1 every query to the random oracle (by the argument prover and by the argument verifier) includes the instance \mathbf{x} . This is consistent with the generic transformation to achieve adaptive security in Section 6.2. However, unlike Section 6.2, the upper bound on the adaptive soundness error in Theorem 14.3.1 does not have a multiplicative factor of t (when compared to the non-adaptive soundness error). This is because we give a direct proof of adaptive soundness for this construction.

14.4 Knowledge soundness

We prove that Construction 14.1.1 satisfies (adaptive) knowledge soundness. The following theorem states that the adaptive knowledge soundness error of the non-interactive argument is upper bounded by the IP state-restoration knowledge soundness error of the underlying IP.

Theorem 14.4.1

Let IP be a public-coin IP for a relation \mathcal{R} with rewinding state-restoration knowledge soundness error $\kappa_{\text{IP}}^{\text{sr}}$ with extraction time et_{IP} (see Definition 13.2.5). For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{NARG} := \text{FS}_{\text{IP}}[\text{IP}, \lambda, s]$ in Construction 14.1.1 is a non-interactive argument for \mathcal{R} with rewinding knowledge soundness error κ_{ARG} with extraction time et_{ARG} (see Definition 7.1.7) such that

- $\kappa_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}}(\lambda, n)) \leq \kappa_{\text{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathcal{P}}}(\lambda, n))$, and
- $\text{et}_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}}(\lambda, n), \tau_{\tilde{\mathcal{P}}}(\lambda, n)) \leq \text{et}_{\text{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathcal{P}}}(\lambda, n), \tau_{\tilde{\mathcal{P}}}(\lambda, n) + O(r_{\max} \cdot t)) + O(r_{\max} \cdot t)$.

Moreover, if the IP state-restoration extractor is straightline (see Definition 13.2.3) then the NARG extractor is also straightline (see Definition 7.1.5). In this case:

- the (straightline) knowledge soundness error is $\kappa_{\text{ARG}}(\lambda, t, n) \leq \kappa_{\text{IP}}^{\text{sr}}(s, t, n)$; and
- the (straightline) extraction time is $\text{et}_{\text{ARG}}(\lambda, t, n) \leq \text{et}_{\text{IP}}^{\text{sr}}(s, t, n) + O(r_{\max} \cdot t)$.

Construction 14.4.2. Let $\mathbf{E}_{\text{IP}}^{\text{sr}}$ be the IP state-restoration extractor for IP . Let FStoSR be the trace translation procedure in Construction 14.3.3. The extractor \mathcal{E} for NARG receives as input an instance \mathbf{x} , argument string π , query-answer trace tr of the argument prover, query-answer trace tr_v of the argument verifier, and black-box access to the argument prover $\tilde{\mathcal{P}}$, and works as follows.

$\mathcal{E}(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}})$:

1. Parse π as a tuple $((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]})$.
2. Compute the move-response trace from the query-answer trace: $\text{tr}^{\text{sr}} := \text{FStoSR}(\text{tr})$.
3. Parse the trace tr_v as k query-answer pairs $((\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i)), \rho_i)_{i \in [k]}$.
4. Use $\tilde{\mathcal{P}}$ to construct the IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}} := \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}})$ in Construction 14.3.2.
5. Compute the witness: $\mathbf{w} \leftarrow \mathbf{E}_{\text{IP}}^{\text{sr}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \text{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$.
6. Output \mathbf{w} .

Proof. Fix a t -query malicious argument prover $\tilde{\mathcal{P}}$. We upper bound the rewinding knowledge soundness error as follows:

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}^f(\mathbf{x}, \pi) \\ \mathbf{w} \leftarrow \mathcal{E}(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}}) \end{array} \right] \\ &= \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]})) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}^f(\mathbf{x}, ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]})) \\ \text{tr}^{\text{sr}} := \text{FStoSR}(\text{tr}) \\ \text{tr}_v = (((\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i)), \rho_i))_{i \in [k]} \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IP}}^{\text{sr}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \text{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}})) \end{array} \right] \\ & \quad \text{(by definition of } \mathcal{E} \text{)} \end{aligned}$$

$$\begin{aligned}
&\leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \end{array} \middle| \begin{array}{l} \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \\ \mathbf{Game}_{\text{IP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IP}}^{\text{sr}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \text{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}})) \end{array} \right] \\
&\quad (\text{by Equation 14.1}) \\
&\leq \kappa_{\text{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}}) \quad (\text{by definition of } \kappa_{\text{IP}}^{\text{sr}}) \\
&\leq \kappa_{\text{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathcal{P}}}) \quad (\text{since } \delta_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}} \leq \delta_{\tilde{\mathcal{P}}} \text{ by Equation 14.1}).
\end{aligned}$$

The running time of \mathcal{E} on $\tilde{\mathcal{P}}$ equals the running time of FStoSR plus the running time of $\mathbf{E}_{\text{IP}}^{\text{sr}}$ on $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}})$:

$$\mathbf{et}_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}}, \tau_{\tilde{\mathcal{P}}}) \leq O(r_{\max} \cdot t) + \mathbf{et}_{\text{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}}, \tau_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}}).$$

Since $\delta_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}} \leq \delta_{\tilde{\mathcal{P}}}$ (by Equation 14.1), $\tau_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}} \leq \tau_{\tilde{\mathcal{P}}} + O(r_{\max} \cdot t)$ (as discussed in Construction 14.3.2), and $\mathbf{et}_{\text{IP}}^{\text{sr}}$ is non-decreasing in the failure probability and running time of the prover, we deduce that

$$\mathbf{et}_{\text{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}}, \tau_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}}) \leq \mathbf{et}_{\text{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathcal{P}}}, \tau_{\tilde{\mathcal{P}}} + O(r_{\max} \cdot t)),$$

which yields the extraction time bound in the lemma statement.

The straightline case The above analysis directly specializes to the straightline case. Suppose that $\mathbf{E}_{\text{IP}}^{\text{sr}}$ is a straightline extractor (Definition 13.2.3): it is a deterministic algorithm that does not need access to the IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$. Then \mathcal{E} in Construction 14.4.2 is a straightline extractor (Definition 7.1.5): there is no need to construct $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ from $\tilde{\mathcal{P}}$, and in particular no need to receive $\tilde{\mathcal{P}}$. In this case $\kappa_{\text{IP}}^{\text{sr}}$ does not depend on the failure probability of $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}})$, so the (straightline) knowledge soundness error bound simplifies to $\kappa_{\text{ARG}}(\lambda, t, n) \leq \kappa_{\text{IP}}^{\text{sr}}(s, t, n)$. Moreover, the running time $\mathbf{et}_{\text{IP}}^{\text{sr}}$ of $\mathbf{E}_{\text{IP}}^{\text{sr}}$ does not depend on the failure probability or running time of $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}})$, so the extraction time bound simplifies to $\mathbf{et}_{\text{ARG}}(\lambda, t, n) \leq \mathbf{et}_{\text{IP}}^{\text{sr}}(s, t, n) + O(r_{\max} \cdot t)$ (the running time of $\mathbf{E}_{\text{IP}}^{\text{sr}}$ plus the running time of FStoSR). \square

15 Optimization: a faster variant

In Chapter 14 we discussed how to transform any public-coin interactive proof (IP) into a corresponding non-interactive argument, via Construction 14.1.1. Here we study a faster variant of that construction: in Section 15.1 we highlight an inefficiency and describe how to resolve it via a modified construction; and in Section 15.2 we analyze the soundness of the modified construction. Later in Chapter 16 we study additional security definitions for this modified construction.

15.1 Construction

We describe an inefficiency of Construction 14.1.1 and then describe a modified construction that addresses this inefficiency.

Problem 1: quadratic blowup in hashing In Construction 14.1.1, for every $i > 1$, the i -th query to the random oracle $x_i := (\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i))$ contains within it the $(i - 1)$ -th query $x_{i-1} := (\mathbf{x}, (\alpha_1, \dots, \alpha_{i-1}), (\tau_1, \dots, \tau_{i-1}))$ to the random oracle. Therefore: (i) the instance \mathbf{x} appears in k queries; and (ii) each IP prover message α_i and each salt τ_i appears in $k - i + 1$ queries. This redundant structure creates a “quadratic cost” in the total size of the queries (that is visually evident in Figure 14.1):

$$\sum_{i \in [k]} \text{len}(x_i) = k \cdot \text{len}(\mathbf{x}) + \sum_{i \in [k]} (k - i + 1) \cdot (\text{len}(\alpha_i) + s) = \Omega(k^2).$$

While this structure in the queries is precisely what ensures a chronological order, the structure also leads to the quadratic cost.

Solution 1: hash chain The aforementioned quadratic cost can be reduced to linear by modifying the construction to use a “hash chain”. The intuition is that each query includes a hash of the previous query, rather than the previous query itself. This creates a “hash chain” structure that enforces the desired chronological order of the k queries. In more detail, the first query remains $x_1 := (\mathbf{x}, \alpha_1, \tau_1)$ as before, and its answer serves as the first IP verifier challenge ρ_1 . The answer ρ_1 additionally serves as the hash of the prior query, which we include in the second query: we define the second query to be $x_2 := (\rho_1, \alpha_2, \tau_2)$. The answer to this second query is ρ_2 , which serves as the second IP verifier challenge as well as the hash of the second query. In general, for every $i \in \{2, \dots, k\}$, the i -th query is $(\rho_{i-1}, \alpha_i, \tau_i)$. This hashing pattern is known as the *Merkle–Damgård construction*, and it is commonly used to hash messages whose size is bigger than a hash function’s input size. Here we use the fact that the hashing is incremental, which (intuitively) ensures that the IP verifier challenges are appropriately interleaved between IP prover messages.

Problem 2: attacks to the hash chain The hash chain must be secure against attacks, in particular ensuring that, for every $i \in \{2, \dots, k\}$, if an adversary makes a query $(\rho_{i-1}, \alpha_i, \tau_i)$ to the oracle f_i then the adversary is committed to (at most) one tuple $(\mathbf{x}, (\alpha_1, \dots, \alpha_{i-1}), (\tau_1, \dots, \tau_{i-1}))$ that includes a choice of instance, prior messages, and prior salts. If the adversary were to find a collision or invert an element in the hash chain, then the adversary may *not* be committed to (at most) one such tuple. The success probability of such attacks for round i depends on the output size of f_i , which is r_i (the number of random bits sent by IP verifier in round i). However the values $(r_i)_{i \in [k]}$ are determined by the underlying IP, and the hash chain is insecure if at least one of them is not large enough.

Solution 2: padding the randomness One solution is to set the output size of each oracle f_i to be $\max\{r_i, \lambda\}$. On the one hand this ensures that there are enough random bits to simulate the underlying IP. On the other hand, this also ensures that the hash chain is secure (even if each r_i equals 1), up to a small additive error that represents the success probability of collision/inversion attacks. For notational simplicity we take the simpler approach of assuming that each r_i is at least λ . This is essentially equivalent to the above because any IP can be straightforwardly modified to satisfy the condition $\min_{i \in [k]} r_i \geq \lambda$ (as the IP verifier can ignore any unused random bits).

Final construction The above considerations lead to the following construction.

Construction 15.1.1: fast Fiat–Shamir transformation for public-coin IPs

Let $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ be a public-coin IP with round complexity k and randomness complexity r . Let $\lambda \in \mathbb{N}$ be a security parameter. Let $s \in \mathbb{N}$ be a privacy parameter. Suppose that $\min_{i \in [k]} r_i \geq \lambda$.

We define $\text{NARG} := \mathbf{FS}_{\text{IP}}^{\star}[\text{IP}, \lambda, s]$ to be the non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ constructed as follows. The argument prover \mathcal{P} receives as input an instance \mathbf{x} and witness \mathbf{w} , and the argument verifier \mathcal{V} receives as input the instance \mathbf{x} and an argument string π . Both receive query access to random oracles $(f_i)_{i \in [k]} \in \mathcal{U}((r_i)_{i \in [k]})$; this implies that the oracle configuration cnf is defined as $\text{cnf}(\lambda, n) := (r_i)_{i \in [k]}$ (see Definition 7.1.1).

- $\mathcal{P}^f(\mathbf{x}, \mathbf{w})$:
 1. For $i = 1, \dots, k$:
 - a) Compute the i -th message (and auxiliary state) of the IP prover:

$$(\alpha_i, \text{aux}_i) := \begin{cases} \mathbf{P}_{\text{IP}}(\mathbf{x}, \mathbf{w}) & \text{if } i = 1 \\ \mathbf{P}_{\text{IP}}(\text{aux}_{i-1}, \rho_{i-1}) & \text{if } i > 1 \end{cases}.$$

- b) Sample a random salt $\tau_i \in \{0, 1\}^s$.
- c) Derive the i -th random message of the IP verifier:

$$\rho_i := \begin{cases} f_1(\mathbf{x}, \alpha_1, \tau_1) \in \{0, 1\}^{r_1} & \text{if } i = 1 \\ f_i(\rho_{i-1}, \alpha_i, \tau_i) \in \{0, 1\}^{r_i} & \text{if } i > 1 \end{cases}.$$

2. Output the argument string $\pi := ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]}).$
- $\mathcal{V}^f(\mathbf{x}, \pi):$
 1. Parse the argument string π as a tuple $((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]}).$
 2. For $i = 1, \dots, k$:
 - a) derive the i -th IP verifier message ρ_i as in Item 1c of the argument prover \mathcal{P} .
 3. Check that the IP verifier accepts: $\mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 1.$

Efficiency We discuss efficiency properties of the above construction.

- *Argument size.* The argument string π contains all IP prover messages (and none of the IP verifier messages) and k salts. Hence the size of π is the prover-to-verifier communication complexity of the IP plus $k \cdot s$:

$$\sum_{i \in [k]} \text{len}(\alpha_i) + k \cdot s = \sum_{i \in [k]} \text{pv}_i + k \cdot s = \text{pv} + k \cdot s.$$

Analogously to the case for an SP in Chapter 10, the transformation *does not compress prover messages*, but rather merely removes the interaction between the prover and verifier.

- *Prover complexity.* The cost of the argument prover \mathcal{P} is essentially the same as that of the underlying IP prover \mathbf{P}_{IP} . The only difference is that the argument prover additionally makes 1 query to the random oracle for each one of the k rounds. The query size in round 1 is $\text{len}(\mathbf{x}) + \text{len}(\alpha_1) + \text{len}(\tau_1) = \text{len}(\mathbf{x}) + \text{pv}_1 + s$, while for every $i \in \{2, \dots, k\}$ the query size in round i is $\text{len}(\rho_{i-1}) + \text{len}(\alpha_i) + \text{len}(\tau_i) + r_{i-1} + \text{pv}_i + s$.
- *Verifier complexity.* The cost of the argument verifier \mathcal{V} is essentially the same as that of the underlying IP verifier \mathbf{V}_{IP} . The only difference is that the argument verifier additionally makes 1 query to the random oracle for each one of the k rounds, and the queries are the same as the argument prover \mathcal{P} .

Remark 15.1.2 (single-oracle variant). A single-oracle variant of Construction 14.1.1 sometimes appears in the literature: for every $i \in [k]$, the i -th random message of the IP verifier is derived as $f(x_i)$ rather than as $f_i(x_i)$ (where $(x_i)_{i \in [k]}$ are the queries defined in Construction 14.1.1). This variants incurs ambiguities that demand a different analysis, and incur a larger soundness error.

In more detail, a security analysis of the single-oracle variant may be unable to determine the round for which a query was intended. In particular, the queries $(x_i)_{i \in [k]}$ may have the same format if the instance and all IP verifier messages have the same length and all IP prover messages have the same length. An adversary can exploit this ambiguity: the adversary can create a long hash chain and after that decide which segment of k hashes to use for constructing an argument string. While it does increase the adversary's probability of convincing the argument verifier, this capability is not devastating. Specifically, the single-oracle variant has soundness error that is (roughly) k times larger than the soundness error of Construction 14.1.1.

Nevertheless this multiplicative cost is noticeable, and it is desirable to avoid it.

Construction 14.1.1 does not have this ambiguity because the i -th query x_i is intended for the i -th oracle f_i . Thus each query-answer pair in the query-answer trace of a malicious argument prover can be uniquely associated to an IP round; in particular, query-answer pairs associated to the first IP round determine the start of a hash chain, preventing the problem described above.¹

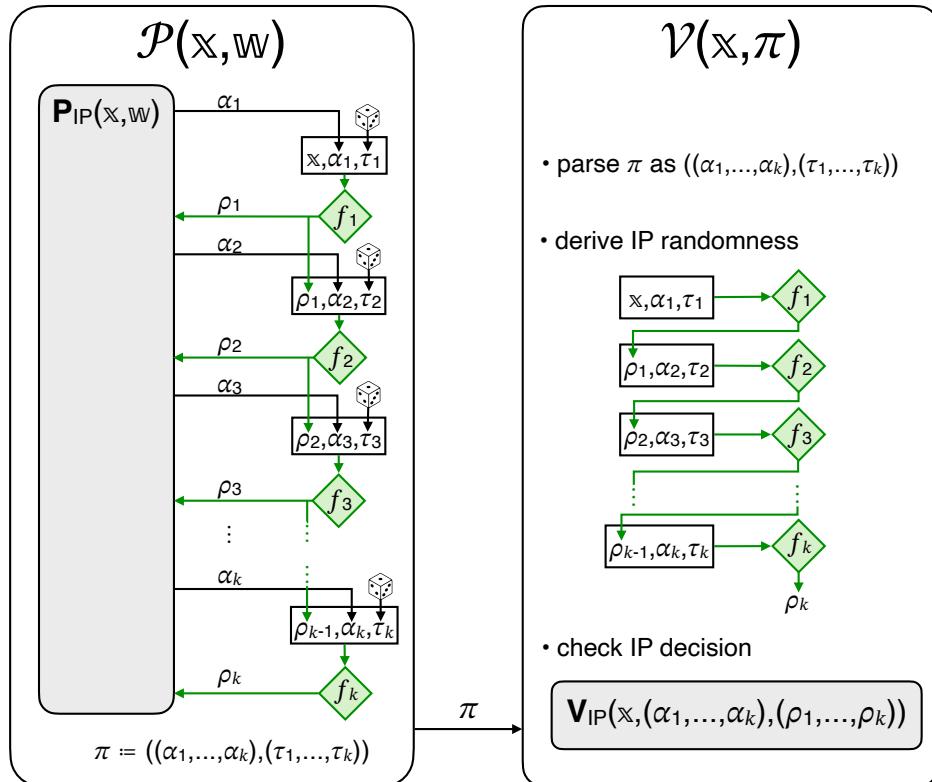


Figure 15.1: Diagram of the fast Fiat-Shamir transformation for public-coin IPs (FS_{IP}^* in Construction 15.1.1).

15.2 Soundness

We prove that Construction 15.1.1 is adaptively sound. The following theorem states that the adaptive soundness error of the non-interactive argument is upper bounded by the IP state-restoration soundness error of the underlying IP plus a small error term (which represents the probability of “breaking” the hash chain). In particular, a small state-restoration soundness error for the IP is *sufficient* for a small soundness error for the resulting non-interactive argument.

¹It would suffice to domain separate the random oracle into two random oracles (rather than k): one for the first round, and one for the other $k - 1$ rounds. However, the case of one random oracle per IP round is easier to analyze, and also leads to minor improvements in concrete security parameters (e.g., the zero-knowledge error that we establish in Theorem 16.2.1 is slightly smaller than one can obtain for the case of two random oracles).

Theorem 15.2.1

Let IP be a public-coin IP for a relation \mathcal{R} with state-restoration soundness error $\epsilon_{\text{IP}}^{\text{sr}}$. For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{NARG} := \mathbf{FS}_{\text{IP}}^{\star}[\text{IP}, \lambda, s]$ in Construction 15.1.1 is a non-interactive argument for \mathcal{R} with adaptive soundness error ϵ_{ARG} (see Definition 7.1.4) such that

$$\epsilon_{\text{ARG}}(\lambda, t, n) \leq \epsilon_{\text{IP}}^{\text{sr}}(s, t, n) + \frac{t^2}{2^\lambda}.$$

Proof. We reduce the soundness of Construction 15.1.1 to the soundness of Construction 14.1.1. We show that every malicious argument prover for Construction 15.1.1 can be translated to a malicious argument prover for Construction 14.1.1 that wins with almost the same probability.

Denote by $\mathcal{V}_{\text{fast}}$ the argument verifier in Construction 15.1.1 and by $\mathcal{V}_{\text{slow}}$ the argument verifier in Construction 14.1.1. Fix a query bound $t \in \mathbb{N}$, t -query argument prover $\tilde{\mathcal{P}}_{\text{fast}}$, and instance size bound $n \in \mathbb{N}$. We construct an argument prover $\tilde{\mathcal{P}}_{\text{slow}}$ that, using $\tilde{\mathcal{P}}_{\text{fast}}$ as a black box, convinces $\mathcal{V}_{\text{slow}}$ with almost the same probability as $\tilde{\mathcal{P}}_{\text{fast}}$ convinces $\mathcal{V}_{\text{fast}}$ (on the same instances). Specifically, see Construction 15.2.6 and Lemma 15.2.7, which directly implies the theorem.

This requires some effort. The construction of $\tilde{\mathcal{P}}_{\text{slow}}$ relies on a procedure F2SAlgo (“fast to slow”) that translates queries of $\tilde{\mathcal{P}}_{\text{fast}}$ to queries of $\tilde{\mathcal{P}}_{\text{slow}}$. In turn, F2SAlgo relies on a subroutine FS.Backtrack , which receives as input a round index, an IP verifier challenge ρ , and a trace tr , and outputs the unique hash chain in tr that ends in ρ (aborting if tr contains no such hash chain or contains multiple such hash chains). Below we define these procedures, and prove a generic claim about them. Then we return to $\tilde{\mathcal{P}}_{\text{slow}}$ and its analysis. \square

Definition 15.2.2. The procedure FS.Backtrack is defined as follows.

$\text{FS.Backtrack}(i, \rho, \text{tr})$:

1. Find query-answer pairs $(x_1, \rho_1), \dots, (x_i, \rho_i)$ in the trace tr such that:
 - (x_1, ρ_1) is a query-answer pair for the oracle f_1 and x_1 has the form $(\mathbf{x}, \alpha_1, \tau_1)$;
 - for every $j \in \{2, \dots, i\}$, (x_j, ρ_j) is a query-answer pair for the oracle f_j and x_j has the form $(\rho_{j-1}, \alpha_j, \tau_j)$;
 - $\rho_i = \rho$.
2. Abort if no such list exists or if at least two such lists exist (of any lengths).
3. Output $(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i))$.

If the query-answer trace tr is sorted lexicographically by answer (and then by query), then FS.Backtrack involves i lookups each taking time $O(r_{\max} \cdot \log t)$, for a total running time of $O(i \cdot r_{\max} \cdot \log t)$. If tr is not sorted then FS.Backtrack first sorts tr in time $O(r_{\max} \cdot t \cdot \log t)$ and then performs the i lookups, for a total running time of $O(r_{\max} \cdot (t + i) \cdot \log t) = O(r_{\max} \cdot (t + k) \cdot \log t)$.

Construction 15.2.3. The procedure F2SAlgo receives query access to random oracles $f = (f_i)_{i \in [k]}$ and black-box access to an algorithm A , an input a , and works as follows.

$\text{F2SAlgo}^f(A)(a)$:

1. For every $i \in [k]$, lazily sample an oracle $\dot{g}_i \leftarrow \mathcal{U}(r_i)$ (to answer malformed queries).

2. Set tr_{fast} to be an empty query-answer trace.
3. Simulate $A(a)$ while answering a query x to the i -th oracle f_i :
 - a) If $i = 1$ and x has the form $(\mathbf{x}, \alpha_1, \tau_1)$:
 - i. answer with $y := f_1(x)$;
 - ii. add (x, y) to tr_{fast} as a query to oracle f_1 .
 - b) If $i > 1$ and x has the form $(\rho_{i-1}, \alpha_i, \tau_i)$:
 - i. run $(\mathbf{x}, (\alpha_1, \dots, \alpha_{i-1}), (\tau_1, \dots, \tau_{i-1})) := \text{FS.Backtrack}(i-1, \rho_{i-1}, \text{tr}_{\text{fast}})$;
 - ii. if FS.Backtrack aborted then answer with $\dot{g}_i(x)$;
 - iii. otherwise answer with $y := f_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i))$, and add (x, y) to tr_{fast} as a query to oracle f_i .
 - c) Else answer with $\dot{g}_i(x)$.
4. Once A halts with outputs b , output b .

We use $b \leftarrow \text{F2SAlgo}^f(A)$ to denote the internal variable tr_{fast} at the end of the execution of A ; note that tr_{fast} is not the query-answer trace of A to f .

If A makes at most t queries to $f = (f_i)_{i \in [k]}$ then $\text{F2SAlgo}(A)$ makes at most t queries to $f = (f_i)_{i \in [k]}$. Moreover, the running time of $\text{F2SAlgo}(A)$ equals the running time of A plus $O(r_{\max} \cdot k \cdot t \cdot \log t)$. Indeed, F2SAlgo runs A and answers each of its t queries following a certain procedure. The time to answer a query is dominated by the time to run FS.Backtrack , which is at most $O(k \cdot r_{\max} \cdot \log t)$ provided that tr_{fast} is sorted; this amounts to $t \cdot O(k \cdot r_{\max} \cdot \log t)$ total work across all t queries. Separately, ensuring that tr_{fast} is always sorted takes time $O(r_{\max} \cdot t \cdot \log t)$ across all t queries.

Construction 15.2.4. The function F2STrace receives as input a query-answer trace tr_{fast} of length t and outputs a query-answer trace tr_{slow} , and works as follows.

$\text{F2STrace}(\text{tr}_{\text{fast}})$:

1. Initialize an empty query-answer trace tr_{slow} .
2. For $j = 1, \dots, t$:
 - a) Let (x, y) be the j -th query-answer pair in tr_{fast} , and let it be to the i -th oracle.
 - b) If $i = 1$ then add (x, y) to tr_{slow} as a query-answer pair for f_1 .
 - c) If $i \in \{2, \dots, k\}$ and x has the form $(\rho_{i-1}, \alpha_i, \tau_i)$ then:
 - i. run $(\mathbf{x}, (\alpha_1, \dots, \alpha_{i-1}), (\tau_1, \dots, \tau_{i-1})) := \text{FS.Backtrack}(i-1, \rho_{i-1}, \text{tr}_{\text{fast}})$;
 - ii. if FS.Backtrack did not abort then add $((\mathbf{x}, (\alpha_1, \dots, \alpha_{i-1}), (\tau_1, \dots, \tau_{i-1})), y)$ to tr_{slow} as a query-answer pair for f_i .
3. Output tr_{slow} .

The function F2STrace can be computed in time $O(r_{\max} \cdot k \cdot t \cdot \log t)$ by sorting tr_{fast} (by answer and then by query) and then, inside the main loop, searching from each previous element in the sequence one by one.

We prove a generic claim about F2SAlgo (Construction 15.2.3) and F2STrace (Construction 15.2.4). We define two events based on a query-answer trace, and then the union of the two events:

1. $E_{\text{col}}(\text{tr}_{\text{fast}})$ is the event that tr_{fast} contains a collision on the same oracle (i.e., there exists $i \in [k]$ such that tr_{fast} contains two distinct queries to the oracle f_i with the same answer);
2. $E_{\text{inv}}(\text{tr}_{\text{fast}})$ is the event that there exist $j_1, j_2 \in [t]$ with $j_1 < j_2$ and $i \in \{2, \dots, k\}$ such that the j_1 -th query in tr_{fast} is to oracle f_i and has the form $(\rho_{i-1}, \alpha_i, \tau_i)$ and the j_2 -th answer in tr_{fast} is to the oracle f_{i-1} and has the answer ρ_{i-1} .

$$3. \quad E(\text{tr}_{\text{fast}}) := E_{\text{col}}(\text{tr}_{\text{fast}}) \vee E_{\text{inv}}(\text{tr}_{\text{fast}}).$$

Claim 15.2.5. *For every algorithm A the following two distributions are identical:*

$$\begin{aligned} & \left\{ \begin{array}{l} (\text{tr}_{\text{slow}}, y) \\ \text{conditioned on} \\ \overline{E(\text{tr})} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ y \xleftarrow{\text{tr}} A^f \\ \text{tr}_{\text{slow}} \leftarrow \text{F2STrace}(\text{tr}) \end{array} \right\} \\ & \equiv \left\{ \begin{array}{l} (\text{tr}, y) \\ \text{conditioned on} \\ \overline{E(\text{tr}_{\text{fast}})} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ y \xleftarrow[\text{tr}_{\text{fast}}]{} \text{F2SAlgo}^f(A) \end{array} \right\}. \end{aligned}$$

Above, tr_{fast} in the right-side experiment is the query-answer trace internally maintained by F2SAlgo, and tr is the query-answer trace to f . Moreover, $\Pr[E(\text{tr})] = \Pr[E(\text{tr}_{\text{fast}})]$.

Proof. We show that the distribution of query answers given to A in both experiments are identical. Then we discuss the distributions of $(\text{tr}_{\text{slow}}, y)$ and (tr, y) .

We separately consider the case of duplicate queries and new queries. Below, for every $j \in [t]$, we let tr_j be the prefix of tr_{fast} containing the first j query-answer pairs.

Case 1: duplicate queries In the left-side experiment, it is clear that duplicate queries to f are answered consistently. Next we argue that, in the right-side experiment, answers given by F2SAlgo^f for duplicate queries are consistent. This is evident for all queries except for queries to f_i of the form $(\rho_{i-1}, \alpha_i, \tau_i)$ for some $i \in \{2, \dots, k\}$. These queries are answered via the output of FS.Backtrack (if no abort) or via the lazily sampled oracles $(g_i)_{i \in [k]}$ (if there is an abort). Duplicate queries are handled in F2SAlgo^f by multiple executions of FS.Backtrack with the same index and input ρ but *different input traces*, which in principle might result in different answers. However, we prove that the relevant executions of FS.Backtrack all have the same output (provided $E_{\text{col}}(\text{tr}_{\text{fast}}) \wedge E_{\text{inv}}(\text{tr}_{\text{fast}})$ holds), using the fact that the traces for later executions are extensions of traces for earlier executions.

Assume that $E_{\text{col}}(\text{tr}_{\text{fast}}) \wedge E_{\text{inv}}(\text{tr}_{\text{fast}})$ holds. Let $j_1, j_2 \in [t]$ with $j_1 < j_2$. Let $(\rho_{i-1}, \alpha_i, \tau_i)$ be the j_1 -th query in tr_{fast} (assume it is a query to f_i). We prove that

$$\text{FS.Backtrack}(i-1, \rho_{i-1}, \text{tr}_{j_1}) = \text{FS.Backtrack}(i-1, \rho_{i-1}, \text{tr}_{j_2}),$$

where above we assume that if FS.Backtrack aborts then it outputs a special abort symbol.

Recall that FS.Backtrack searches for valid sequences $(x_1, \rho_1), \dots, (x_{i-1}, \rho_{i-1})$ of query-answer pairs in the given trace. It aborts if no valid sequence is found, or if multiple valid sequences are found. We distinguish between different cases.

- $\text{FS.Backtrack}(i-1, \rho_{i-1}, \text{tr}_{j_1})$ aborts and $\text{FS.Backtrack}(i-1, \rho_{i-1}, \text{tr}_{j_2})$ aborts. In this case, both executions output the same special abort symbol and hence have the same output.
- $\text{FS.Backtrack}(i-1, \rho_{i-1}, \text{tr}_{j_1})$ aborts but $\text{FS.Backtrack}(i-1, \rho_{i-1}, \text{tr}_{j_2})$ does not abort. This means that tr_{j_2} contains a valid sequence but tr_{j_1} does not contain one. However, since the query $(\rho_{i-1}, \alpha_i, \tau_i)$ was performed in tr_{j_1} this contradicts $\overline{E_{\text{inv}}(\text{tr}_{\text{fast}})}$.
- $\text{FS.Backtrack}(i-1, \rho_{i-1}, \text{tr}_{j_1})$ does not abort and $\text{FS.Backtrack}(i-1, \rho_{i-1}, \text{tr}_{j_2})$ aborts. This means that tr_{j_1} contains a valid sequence, and thus also tr_{j_2} . If $\text{FS.Backtrack}(i-1, \rho_{i-1}, \text{tr}_{j_2})$ aborts then it must be because more than one valid sequence exists, which contradicts $\overline{E_{\text{col}}(\text{tr}_{\text{fast}})}$.

- $\text{FS.Backtrack}(i - 1, \rho_{i-1}, \text{tr}_{j_1})$ does not abort and $\text{FS.Backtrack}(i - 1, \rho_{i-1}, \text{tr}_{j_2})$ does not abort. This means that tr_{j_1} contains a single valid sequence and tr_{j_2} contains a single valid sequence. These sequences must be the same, as otherwise tr_{j_2} would contain two sequences (as tr_{j_1} is a prefix of tr_{j_2}), contradicting $\overline{E_{\text{col}}(\text{tr}_{\text{fast}})}$ (two valid sequences would cause a collision with the same oracle). Thus, the output of FS.Backtrack is the same in both cases.

Case 2: new queries It is clear that on the left side each new query gets a random response conditioned on the events. We move to the right side, and argue that answers given by F2SAlgo^f to new query x (a query that was not previously issued) are uniformly random, conditioned on the event. Suppose that x is the j -th query. There are several cases.

- The query is to f_1 but does not have the form $(\mathbf{x}, \alpha_1, \tau_1)$. In this case F2SAlgo^f answers with $\dot{g}_1(x)$. Since x is a new query, \dot{g}_1 has not been queried at x and thus the answer $\dot{g}_1(x)$ is uniformly random.
- The query is to f_1 and has the form $(\mathbf{x}, \alpha_1, \tau_1)$. In this case F2SAlgo^f answers with $f_1(x)$. Since x is a new query, f_1 has not been queried at x and thus the answer $f_1(x)$ is uniformly random.
- The query is to f_i (for $1 < i \leq k$) but does not have the form $(\rho_{i-1}, \alpha_i, \tau_i)$. In this case F2SAlgo^f answers with $\dot{g}_i(x)$. Since x is a new query, \dot{g}_i has not been queried at x and thus the answer $\dot{g}_i(x)$ is uniformly random.
- The query is to f_i (for $1 < i \leq k$), has the form $(\rho_{i-1}, \alpha_i, \tau_i)$, but FS.Backtrack aborts. In this case F2SAlgo^f answers with $\dot{g}_i(x)$. Since x is a new query, \dot{g}_i has not been queried at x and thus the answer $\dot{g}_i(x)$ is uniformly random.
- The query is to f_i (for $1 < i \leq k$), has the form $(\rho_{i-1}, \alpha_i, \tau_i)$, and FS.Backtrack does not abort. Denote the output of FS.Backtrack as $(\mathbf{x}, (\alpha_1, \dots, \alpha_{i-1}), (\tau_1, \dots, \tau_{i-1}))$. In this case F2SAlgo^f answers with $f_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i))$. Suppose, towards a contradiction, that $(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i))$ was queried before. Hence there exists a previous query at time $j' < j$ of the form $(\rho'_{i-1}, \alpha_i, \tau_i)$ such that

$$\text{FS.Backtrack}(i - 1, \rho'_{i-1}, \text{tr}_{j'}) = \text{FS.Backtrack}(i - 1, \rho_{i-1}, \text{tr}_j).$$

This implies that $\rho'_i = \rho_i$ and thus it is a duplicate query. Since we assumed that x is a new query, we deduce that the answer $f_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i))$ is uniformly random.

In both experiments the queries of A are answered with the same distribution, so A 's output y has the same distribution in both experiments. Moreover, in the left-side experiment tr_{slow} is the output of F2STrace applied to A 's query-answer trace, while in the right-side experiment tr is the query-answer trace of $\text{F2SAlgo}(A)$; both F2STrace and F2SAlgo apply the same logic to A 's queries, and therefore produce the same trace (since A 's queries are answered with the same distribution). Overall, the distribution of $(\text{F2STrace}(\text{tr}), y)$ is identical to that of (tr, y) which concludes the proof.

Moreover, the events $E(\text{tr})$ and $E(\text{tr}_{\text{fast}})$ are the same event applied to the two traces that we have shown have the same distribution. We conclude that $\Pr[E(\text{tr})] = \Pr[E(\text{tr}_{\text{fast}})]$. \square

Construction 15.2.6. The argument prover $\tilde{\mathcal{P}}_{\text{slow}}$ receives query access to the random oracles $f = (f_i)_{i \in [k]}$ and black-box access to the argument prover $\tilde{\mathcal{P}}_{\text{fast}}$, and is defined as

$$\tilde{\mathcal{P}}_{\text{slow}}^f(\tilde{\mathcal{P}}_{\text{fast}}) := \text{F2SAlgo}^f(\tilde{\mathcal{P}}_{\text{fast}}).$$

The running time of $\tilde{\mathcal{P}}_{\text{slow}}$ is $\tau_{\tilde{\mathcal{P}}_{\text{fast}}} + O(r_{\max} \cdot k \cdot t \cdot \log t)$.

Lemma 15.2.7. *The following two distributions are $\frac{t^2}{2^\lambda}$ -close in statistical distance:*

$$\left\{ (b, \mathbf{x}, \pi, \text{tr}_{\text{slow}}) \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr}_{\text{slow}}]{\text{tr}} \tilde{\mathcal{P}}_{\text{fast}}^f \\ b \xleftarrow[\text{tr}_{\text{slow}}]{\text{tr}_v} \mathcal{V}_{\text{fast}}^f(\mathbf{x}, \pi) \\ \text{tr}_{\text{slow}} \leftarrow \text{F2STrace}(\text{tr} \parallel \text{tr}_v) \end{array} \right\} \text{ and } \left\{ (b, \mathbf{x}, \pi, \text{tr} \parallel \text{tr}_v) \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr} \parallel \text{tr}_v]{\text{tr}} \tilde{\mathcal{P}}_{\text{slow}}^f \\ b \xleftarrow[\text{tr} \parallel \text{tr}_v]{\text{tr}_v} \mathcal{V}_{\text{slow}}^f(\mathbf{x}, \pi) \end{array} \right\}.$$

Proof. First, we upper bound the probability of the event $E(\text{tr}_{\text{fast}}) = E_{\text{col}}(\text{tr}_{\text{fast}}) \vee E_{\text{inv}}(\text{tr}_{\text{fast}})$.

Claim 15.2.8. *The following bound holds*

$$\Pr \left[E(\text{tr}_{\text{fast}} \parallel \text{tr}_{\text{fast},v}) \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr}_{\text{fast}}]{\text{tr}} \text{F2SAlgo}^f(\tilde{\mathcal{P}}_{\text{fast}}) \\ b \xleftarrow[\text{tr}_{\text{fast},v}]{\text{tr}_v} \text{F2SAlgo}^f(\mathcal{V}_{\text{fast}}(\mathbf{x}, \pi)) \end{array} \right] \leq \frac{t^2}{2^\lambda}.$$

Proof. Let I_i be the indices of queries to oracle f_i in tr_{fast} , and let $t_i := |I_i|$; also let $t_0 := 0$. Note that $\sum_{i \in [k]} t_i \leq t + k$, because each simulated query of $\tilde{\mathcal{P}}_{\text{fast}}$ (which makes at most t queries) adds at most one query to tr_{fast} , and each simulated query of $\mathcal{V}_{\text{fast}}$ (which makes k queries) adds at most one query to tr_{fast} . Also, $t_i \leq t + 1$ for every $i \in [k]$ because $\mathcal{V}_{\text{fast}}$ makes one query to each oracle.

For every $j \in [t+k]$, let tr_j the prefix of tr_{fast} of length j . We upper bound the probability that the j -th query causes the first collision or inversion: we upper bound the probability of the event $E(\text{tr}_j)$ conditioned on $\overline{E(\text{tr}_{j-1})}$. Let A_j be the algorithm that performs the first $j-1$ queries of A and then outputs the j -th query x_j . By Claim 15.2.5 applied to A_j , the answer to x_j from F2SAlgo^f is uniformly random. This lets us derive the following two bounds.

- Let E_{col}^j be the event that the j -th query causes a collision. Suppose that $E(\text{tr}_{j-1})$ does not hold, and suppose that the j -th query is to oracle f_i . Fix any previous query to oracle f_i ; the probability that the j -th query has the same answer is at most $\frac{1}{2^{t_i}} \leq \frac{1}{2^\lambda}$. Letting $t_{i,j}$ be the number of queries to f_i before the j -th query, we deduce that $\Pr \left[E_{\text{col}}^j \mid \overline{E(\text{tr}_{j-1})} \right] \leq \frac{t_{i,j}}{2^\lambda}$.

Let $i(j) \in [k]$ be the index such that the j -th query is to oracle $f_{i(j)}$. Then

$$\sum_{j \in [t+k]} \Pr \left[E_{\text{col}}^j \mid \overline{E(\text{tr}_{j-1})} \right] \leq \sum_{j \in [t+k]} \frac{t_{i(j),j}}{2^\lambda} = \sum_{i \in [k]} \sum_{\ell=1}^{t_i} \frac{\ell}{2^\lambda} \leq \sum_{i \in [k]} \frac{1}{2} \cdot \frac{t_i^2}{2^\lambda}.$$

- Let E_{inv}^j be the event that the j -th query causes an inversion, i.e., there exists $j' \in \{1, \dots, j-1\}$ such that: (i) the j' -th query is to f_i and has the form $(\rho_{i-1}, \alpha_i, \tau_i)$; and (ii) the j -th query, denoted by x_j , is to oracle f_{i-1} , does not equal any prior query, and has the answer $f_{i-1}(x_j) = \rho_{i-1}$. Since the answer to x_j is random (as $E(\text{tr}_{j-1})$ does not hold), the probability that x_j causes an inversion is $\Pr \left[f_{i-1}(x_j) = \rho_{i-1} \mid \overline{E(\text{tr}_{j-1})} \right] = \frac{1}{2^{t_{i-1}}} \leq \frac{1}{2^\lambda}$.

This lets us derive the following bound:

$$\sum_{j \in [t+k]} \Pr \left[E_{\text{inv}}^j \mid \overline{E(\text{tr}_{j-1})} \right] \leq \sum_{i=2}^k \sum_{j \in I_{i-1}, j' \in I_i} \Pr \left[f_{i-1}(x_j) = \rho_{i-1} \mid \overline{E(\text{tr}_{j-1})} \right]$$

$$\begin{aligned} &\leq \sum_{i=2}^k \sum_{j \in I_{i-1}, j' \in I_i} \frac{1}{2^\lambda} \\ &= \sum_{i=2}^k \frac{t_i \cdot t_{i+1}}{2^\lambda}. \end{aligned}$$

Together, we conclude the following upper bound for the probability of $E(\text{tr}_{\text{fast}})$:

$$\begin{aligned} \Pr[E(\text{tr}_{\text{fast}})] &= \Pr[E_{\text{col}}(\text{tr}_{\text{fast}}) \vee E_{\text{inv}}(\text{tr}_{\text{fast}})] \\ &\leq \sum_{j \in [t+k]} \Pr \left[E_{\text{col}}^j(\text{tr}_{\text{fast}}) \vee E_{\text{inv}}^j(\text{tr}_{\text{fast}}) \mid \overline{E(\text{tr}_{j-1})} \right] \\ &\leq \sum_{j \in [t+k]} \Pr \left[E_{\text{col}}^j(\text{tr}_{\text{fast}}) \mid \overline{E(\text{tr}_{j-1})} \right] + \Pr \left[E_{\text{inv}}^j(\text{tr}_{\text{fast}}) \mid \overline{E(\text{tr}_{j-1})} \right] \\ &\leq \frac{1}{2^\lambda} \cdot \left(\sum_{i \in [k]} \frac{t_i^2}{2} + \sum_{i=2}^k t_{i-1} \cdot t_i \right) \\ &\leq \frac{1}{2^\lambda} \cdot \left(\frac{(t+1)^2}{2} + \frac{(t+1)^2}{4} \right) \\ &\leq \frac{t^2}{2^\lambda}. \end{aligned}$$

In the second to last inequality above, we use the upper bound $\sum_{i \in [k]} \frac{t_i^2}{2} \leq \frac{(t+1)^2}{2}$ since the maximum is attained when $t_i = t+1$ for some $i \in [k]$, and we use the upper bound $\sum_{i \in [k]} t_{i-1} \cdot t_i \leq \frac{(t+1)^2}{4}$ implied by Lemma 1.4.4 (by setting $\delta_i := \frac{t_i}{t+1}$). The last inequality holds for every $t \geq 7$.

□

Next, we relate the execution of $\mathcal{V}_{\text{slow}}^f(\mathbf{x}, \pi) = 1$ to the execution of $\text{F2SAlgo}^f(\mathcal{V}_{\text{fast}})(\mathbf{x}, \pi) = 1$.

Claim 15.2.9. *The following two distributions are equivalent:*

$$\begin{aligned} &\left\{ \begin{array}{l} (b, \mathbf{x}, \pi, \text{tr} \parallel \text{tr}_v) \\ \text{conditioned on} \\ \overline{E(\text{tr}_{\text{fast}} \parallel \text{tr}_{\text{fast},v})} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr}_{\text{fast}}]{\text{tr}} \tilde{\mathcal{P}}_{\text{slow}}^f \\ b \xleftarrow[\text{tr}_{\text{fast},v}]{\text{tr}_v} \text{F2SAlgo}^f(\mathcal{V}_{\text{fast}})(\mathbf{x}, \pi) \end{array} \right\} \\ &\equiv \left\{ \begin{array}{l} (b, \mathbf{x}, \pi, \text{tr} \parallel \text{tr}_v) \\ \text{conditioned on} \\ \overline{E(\text{tr}_{\text{fast}} \parallel \text{tr}_{\text{fast},v})} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr}_{\text{fast}}]{\text{tr}} \tilde{\mathcal{P}}_{\text{slow}}^f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}_{\text{slow}}^f(\mathbf{x}, \pi) \\ b' \xleftarrow[\text{tr}_{\text{fast},v}]{\text{tr}'_v} \text{F2SAlgo}^f(\mathcal{V}_{\text{fast}})(\mathbf{x}, \pi) \end{array} \right\}. \end{aligned}$$

Proof. The analysis below relies only on the fact that the event $E_{\text{col}}(\text{tr}_{\text{fast}})$ does not hold; the claim statement includes the condition that the event $E_{\text{inv}}(\text{tr}_{\text{fast}})$ does not hold only to simplify how the claim is invoked later on. Let (\mathbf{x}, π) be the output of $\tilde{\mathcal{P}}_{\text{slow}}$, where $\pi = ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]})$.

On the right side, $\mathcal{V}_{\text{slow}}^f(\mathbf{x}, \pi) = 1$ if and only if $\mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 1$ where $\forall i \in [k], \rho_i = f_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i))$.

On the left side, $\text{F2SAlgo}^f(\mathcal{V}_{\text{fast}}(\mathbf{x}, \pi)) = 1$ if and only if $\mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}) = 1$ where ρ'_i is derived by the F2SAlgo^f algorithm. We show that $\rho'_i = \rho_i$ for every $i \in [k]$.

The case where $i = 1$ holds straightforwardly because

$$\rho'_1 = f_1(\mathbf{x}, \alpha_1, \tau_1) = \rho_1.$$

We continue to the case where $i = 2$. The randomness ρ'_2 is derived as follows. The algorithm F2SAlgo_2^f , on input $(\rho_1, \alpha_2, \tau_2)$, runs $\text{FS.Backtrack}(1, \rho_1, \text{tr})$ and must obtain the output $(\mathbf{x}, \alpha_1, \tau_1)$; indeed, a different output would yield a collision in tr_{fast} (contradicting $E_{\text{col}}(\text{tr}_{\text{fast}})$). Hence, we get that $f_2(\mathbf{x}, (\alpha_1, \alpha_2), (\tau_1, \tau_2)) = \rho_2$.

This continues in a similar manner up to k , which we show by induction on i . We already discussed the base case of $i = 2$ (and also $i = 1$). For every $i \in \{3, \dots, k\}$, ρ'_i is derived as follows. The algorithm F2SAlgo^f , on input $(\rho_{i-1}, \alpha_i, \tau_i)$, runs $\text{FS.Backtrack}(i-1, \rho_{i-1}, \text{tr}_{\text{fast}})$. Note that $\text{FS.Backtrack}(i-1, \rho_{i-1}, \text{tr}_{\text{fast}})$ does not abort because: (i) the trace tr_{fast} contains all queries of $\text{F2SAlgo}^f(\mathcal{V}_{\text{fast}}(\mathbf{x}, \pi))$, so FS.Backtrack has at least one chain to backtrack to; and (ii) no other chain can exist since that would imply a collision (contradicting $E_{\text{col}}(\text{tr}_{\text{fast}})$). By the induction hypothesis, $\text{FS.Backtrack}(i-2, \rho_{i-2}, \text{tr}_{\text{fast}})$ gives the output $(\mathbf{x}, (\alpha_1, \dots, \alpha_{i-2}), (\tau_1, \dots, \tau_{i-2}))$. This means that the output of $\text{FS.Backtrack}(i-1, \rho_{i-1}, \text{tr}_{\text{fast}})$ must be $(\mathbf{x}, (\alpha_1, \dots, \alpha_{i-1}), (\tau_1, \dots, \tau_{i-1}))$, as a different output would yield a collision in tr_{fast} . Hence, we get that $f_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i)) = \rho_i$, which lets us conclude that

$$\rho'_i = f_i(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\tau_1, \dots, \tau_i)) = \rho_i.$$

□

The claims we have proven allow us to upper bound the statistical distance as follows. We begin by showing that the following two distributions are equivalent:

$$\left\{ \begin{array}{l} (b, \mathbf{x}, \pi, \text{tr}_{\text{slow}}) \\ \text{conditioned on} \\ \frac{E(\text{tr} \parallel \text{tr}_{\nu})}{E(\text{tr} \parallel \text{tr}_{\nu})} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}_{\text{fast}}^f \\ b \xleftarrow{\text{tr}_{\nu}} \mathcal{V}_{\text{fast}}^f(\mathbf{x}, \pi) \\ \text{tr}_{\text{slow}} \leftarrow \text{F2STrace}(\text{tr} \parallel \text{tr}_{\nu}) \end{array} \right\}$$

and

$$\left\{ \begin{array}{l} (b, \mathbf{x}, \pi, \text{tr} \parallel \text{tr}_{\nu}) \\ \text{conditioned on} \\ \frac{E(\text{tr}_{\text{fast}} \parallel \text{tr}_{\text{fast}, \nu})}{E(\text{tr}_{\text{fast}} \parallel \text{tr}_{\text{fast}, \nu})} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr}_{\text{fast}}]{\text{tr}} \tilde{\mathcal{P}}_{\text{slow}}^f \\ b \xleftarrow{\text{tr}_{\nu}} \mathcal{V}_{\text{slow}}^f(\mathbf{x}, \pi) \\ b' \xleftarrow[\text{tr}_{\text{fast}, \nu}]{\text{tr}_{\nu}} \text{F2SAlgo}^f(\mathcal{V}_{\text{fast}})(\mathbf{x}, \pi) \end{array} \right\}.$$

Note that $\overline{E(\text{tr} \parallel \text{tr}_{\nu})}$ implies that $\overline{E(\text{tr})}$ and $\overline{E(\text{tr}_{\nu})}$. Hence, by applying Claim 15.2.5 for $A := \tilde{\mathcal{P}}_{\text{fast}}$ the left-side distribution is equivalent to

$$\left\{ \begin{array}{l} (b, \mathbf{x}, \pi, \text{tr} \parallel \text{tr}_{\text{slow}, \nu}) \\ \text{conditioned on} \\ \frac{E(\text{tr}_{\text{fast}} \parallel \text{tr}_{\nu})}{E(\text{tr}_{\text{fast}} \parallel \text{tr}_{\nu})} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr}_{\text{fast}}]{\text{tr}} \text{F2SAlgo}^f(\tilde{\mathcal{P}}_{\text{fast}}) \\ b \xleftarrow{\text{tr}_{\nu}} \mathcal{V}_{\text{fast}}^f(\mathbf{x}, \pi) \\ \text{tr}_{\text{slow}, \nu} \leftarrow \text{F2STrace}(\text{tr}_{\nu}) \end{array} \right\};$$

moreover, the claim tells us that $\Pr[E(\text{tr} \parallel \text{tr}_\nu)] = \Pr[E(\text{tr}_{\text{fast}} \parallel \text{tr}_\nu)]$. Then, by applying Claim 15.2.5 for $A := \mathcal{V}_{\text{fast}}$ the last distribution is equivalent to:

$$\left\{ \begin{array}{l} (b, \mathbf{x}, \pi, \text{tr} \parallel \text{tr}_\nu) \\ \text{conditioned on} \\ \overline{E(\text{tr}_{\text{fast}} \parallel \text{tr}_{\text{fast},\nu})} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr}_{\text{fast}}]{\text{tr}} \text{F2SAlgo}^f(\tilde{\mathcal{P}}_{\text{fast}}) \\ b \xleftarrow[\text{tr}_{\text{fast},\nu}]{\text{tr}_\nu} \text{F2SAlgo}^f(\mathcal{V}_{\text{fast}})(\mathbf{x}, \pi) \end{array} \right\};$$

moreover, the claim tells us that $\Pr[E(\text{tr}_{\text{fast}} \parallel \text{tr}_\nu)] = \Pr[E(\text{tr}_{\text{fast}} \parallel \text{tr}_{\text{fast},\nu})] = \Pr[E(\text{tr} \parallel \text{tr}_\nu)]$. By the definition of $\tilde{\mathcal{P}}_{\text{slow}}$ (see Construction 15.2.6), the last distribution is equivalent to:

$$\left\{ \begin{array}{l} (b, \mathbf{x}, \pi, \text{tr} \parallel \text{tr}_\nu) \\ \text{conditioned on} \\ \overline{E(\text{tr}_{\text{fast}} \parallel \text{tr}_{\text{fast},\nu})} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr}_{\text{fast}}]{\text{tr}} \tilde{\mathcal{P}}_{\text{slow}}^f \\ b \xleftarrow[\text{tr}_{\text{fast},\nu}]{\text{tr}_\nu} \text{F2SAlgo}^f(\mathcal{V}_{\text{fast}})(\mathbf{x}, \pi) \end{array} \right\}.$$

Finally, by Claim 15.2.9 the last distribution is equivalent to:

$$\left\{ \begin{array}{l} (b, \mathbf{x}, \pi, \text{tr} \parallel \text{tr}_\nu) \\ \text{conditioned on} \\ \overline{E(\text{tr}_{\text{fast}} \parallel \text{tr}_{\text{fast},\nu})} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr}_{\text{fast}}]{\text{tr}} \tilde{\mathcal{P}}_{\text{slow}}^f \\ b \xleftarrow[\text{tr}_{\text{fast},\nu}]{\text{tr}_\nu} \mathcal{V}_{\text{slow}}^f(\mathbf{x}, \pi) \\ b' \xleftarrow[\text{tr}_{\text{fast},\nu}]{\text{tr}'_\nu} \text{F2SAlgo}^f(\mathcal{V}_{\text{fast}})(\mathbf{x}, \pi) \end{array} \right\}.$$

By Claim 15.2.8, we get the bound $\Pr[E(\text{tr} \parallel \text{tr}_\nu)] \leq \frac{t^2}{2^\lambda}$. Moreover, we have argued above that $\Pr[E(\text{tr} \parallel \text{tr}_\nu)] = \Pr[E(\text{tr}_{\text{fast}} \parallel \text{tr}_{\text{fast},\nu})]$. Thus we can apply Claim 1.2.10, and obtain that:

$$\begin{aligned} & \Delta \left(\left\{ (b, \mathbf{x}, \pi, \text{tr}_{\text{slow}}) \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr}_{\text{slow}}]{\text{tr}} \tilde{\mathcal{P}}_{\text{fast}}^f \\ b \xleftarrow[\text{tr}_{\text{slow}}]{\text{tr}} \mathcal{V}_{\text{fast}}^f(\mathbf{x}, \pi) \\ \text{tr}_{\text{slow}} \leftarrow \text{F2STrace}(\text{tr} \parallel \text{tr}_\nu) \end{array} \right\}, \left\{ (b, \mathbf{x}, \pi, \text{tr} \parallel \text{tr}_\nu) \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr}_{\text{slow}}]{\text{tr}} \tilde{\mathcal{P}}_{\text{slow}}^f \\ b \xleftarrow[\text{tr}_{\text{slow}}]{\text{tr}} \mathcal{V}_{\text{slow}}^f(\mathbf{x}, \pi) \end{array} \right\} \right) \\ & \leq \Delta \left(\left\{ (b, \mathbf{x}, \pi, \text{tr}_{\text{slow}}) \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr}_{\text{slow}}]{\text{tr}} \tilde{\mathcal{P}}_{\text{fast}}^f \\ b \xleftarrow[\text{tr}_{\text{slow}}]{\text{tr}} \mathcal{V}_{\text{fast}}^f(\mathbf{x}, \pi) \\ \text{tr}_{\text{slow}} \leftarrow \text{F2STrace}(\text{tr} \parallel \text{tr}_\nu) \end{array} \right\}, \left\{ (b, \mathbf{x}, \pi, \text{tr} \parallel \text{tr}_\nu) \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr}_{\text{fast}}]{\text{tr}} \tilde{\mathcal{P}}_{\text{slow}}^f \\ b \xleftarrow[\text{tr}_{\text{fast}}]{\text{tr}} \mathcal{V}_{\text{slow}}^f(\mathbf{x}, \pi) \end{array} \right\} \right) \\ & \quad + \Pr[E(\text{tr} \parallel \text{tr}_\nu)] \\ & \leq 0 + \frac{t^2}{2^\lambda} \leq \frac{t^2}{2^\lambda}. \end{aligned}$$

□

Remark 15.2.10 (on adaptive security). In Construction 15.1.1 only the first query to the random oracle (by the argument prover and by the argument verifier) includes the instance

\mathbf{x} . This differs from Construction 14.1.1 and the generic transformation to achieve adaptive security in Section 6.2. Indeed, in Construction 15.1.1, it suffices for the hash chain to start with the instance \mathbf{x} to achieve adaptive soundness; moreover, adaptive soundness in Theorem 15.2.1 is “for free” because the underlying security reduction does not incur a multiplicative factor of t in the adaptive soundness error compared to the non-adaptive soundness error (the generic case in Section 6.2 has this factor).

16 Additional security definitions

We prove that Construction 15.1.1 satisfies additional security definitions.

16.1 Knowledge soundness

In Construction 15.1.1, if the IP satisfies (state-restoration) knowledge soundness then the resulting non-interactive argument satisfies adaptive knowledge soundness.

Theorem 16.1.1

Let IP be a public-coin IP for a relation \mathcal{R} with rewinding state-restoration knowledge soundness error $\kappa_{\text{IP}}^{\text{sr}}$ with extraction time et_{IP} (see Definition 13.2.5), round complexity k , and maximum randomness complexity $r_{\max} := \max_{i \in [k]} r_i$. For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{NARG} := \text{FS}_{\text{IP}}^{\star}[\text{IP}, \lambda, s]$ in Construction 15.1.1 is a non-interactive argument for \mathcal{R} with rewinding knowledge soundness error κ_{ARG} with extraction time et_{ARG} (see Definition 7.1.7) such that

- $\kappa_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}}(\lambda, n)) \leq \kappa_{\text{IP}}^{\text{sr}}(s, t, n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n)) + \frac{t^2}{2^\lambda}$, and
- $\text{et}_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}}(\lambda, n), \tau_{\tilde{\mathcal{P}}}(\lambda, n)) \leq \text{et}_{\text{IP}}^{\text{sr}}(s, t, n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n), \tau'_{\tilde{\mathcal{P}}}(\lambda, n)) + O(r_{\max} \cdot k \cdot t \cdot \log t)$.

Above, $\delta'_{\tilde{\mathcal{P}}}(\lambda, n) := \delta_{\tilde{\mathcal{P}}}(\lambda, n) + \frac{t^2}{2^\lambda}$ and $\tau'_{\tilde{\mathcal{P}}}(\lambda, n) := \tau_{\tilde{\mathcal{P}}}(\lambda, n) + O(r_{\max} \cdot k \cdot t \cdot \log t)$.

Moreover, if the IP state-restoration extractor is straightline (see Definition 13.2.3) then the NARG extractor is also straightline (see Definition 7.1.5). In this case:

- the (straightline) knowledge soundness error is $\kappa_{\text{ARG}}(\lambda, t, n) \leq \kappa_{\text{IP}}^{\text{sr}}(s, t, n) + \frac{t^2}{2^\lambda}$; and
- the (straightline) extraction time is $\text{et}_{\text{ARG}}(\lambda, t, n) \leq \text{et}_{\text{IP}}^{\text{sr}}(s, t, n) + O(r_{\max} \cdot k \cdot t \cdot \log t)$.

Construction 16.1.2. Let $\mathcal{E}_{\text{slow}}$ be the knowledge extractor for $\mathcal{V}_{\text{slow}}$ in Construction 14.4.2 (constructed and analyzed in Section 14.4). The extractor $\mathcal{E}_{\text{fast}}$ for $\mathcal{V}_{\text{fast}}$ receives as input an instance \mathbf{x} , argument string π , query-answer trace tr of the argument prover, query-answer trace tr_v of the argument verifier, and (black-box access to) the IP prover $\tilde{\mathcal{P}}_{\text{fast}}$, and works as follows.

$\mathcal{E}_{\text{fast}}(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}}_{\text{fast}})$:

1. Compute the query-answer trace $\text{tr}_{\text{slow}} := \text{F2STrace}(\text{tr} \parallel \text{tr}_v)$.
2. Set $\text{tr}_{\text{slow},v}$ to be the suffix of tr_{slow} or length k .
3. Let $\tilde{\mathcal{P}}_{\text{slow}} := \tilde{\mathcal{P}}_{\text{slow}}(\tilde{\mathcal{P}}_{\text{fast}})$ be the argument prover in Construction 15.2.6.
4. Output $\mathbf{w} := \mathcal{E}_{\text{slow}}(\mathbf{x}, \pi, \text{tr}_{\text{slow}}, \text{tr}_{\text{slow},v}, \tilde{\mathcal{P}}_{\text{slow}})$.

Proof. Fix a t -query malicious argument prover $\tilde{\mathcal{P}}_{\text{fast}}$. We analyze the error probability of the extractor $\mathcal{E}_{\text{fast}}$. We can write:

$$\begin{aligned}
 & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xrightarrow{\text{tr}} \tilde{\mathcal{P}}_{\text{fast}}^f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}_{\text{fast}}^f(\mathbf{x}, \pi) \\ \mathbf{w} \leftarrow \mathcal{E}_{\text{fast}}(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}}_{\text{fast}}) \end{array} \right] \\
 &= \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xrightarrow{\text{tr}} \tilde{\mathcal{P}}_{\text{fast}}^f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}_{\text{fast}}^f(\mathbf{x}, \pi) \\ \text{tr}_{\text{slow}} := \text{F2STrace}(\text{tr} \| \text{tr}_v) \\ \text{tr}_{\text{slow}, v} \text{ is the suffix of } \text{tr}_{\text{slow}} \\ \mathbf{w} \leftarrow \mathcal{E}_{\text{slow}}(\mathbf{x}, \pi, \text{tr}_{\text{slow}}, \text{tr}_{\text{slow}, v}, \tilde{\mathcal{P}}_{\text{slow}}) \end{array} \right] \quad (\text{by definition of } \mathcal{E}_{\text{fast}}) \\
 &\leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xrightarrow{\text{tr}} \tilde{\mathcal{P}}_{\text{slow}}^f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}_{\text{slow}}^f(\mathbf{x}, \pi) \\ \mathbf{w} \leftarrow \mathcal{E}_{\text{slow}}(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}}_{\text{slow}}) \end{array} \right] + \frac{t^2}{2^\lambda} \quad (\text{by Lemma 15.2.7}) \\
 &\leq \kappa_{\text{slow}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}_{\text{slow}}}) \quad (\text{by the knowledge soundness error for the } t\text{-query prover } \tilde{\mathcal{P}}_{\text{slow}}^f) \\
 &\leq \kappa_{\text{slow}} \left(\lambda, t, n, \delta_{\tilde{\mathcal{P}}_{\text{fast}}} + \frac{t^2}{2^\lambda} \right) \quad (\text{by Lemma 15.2.7, } \delta_{\tilde{\mathcal{P}}_{\text{slow}}} \leq \delta_{\tilde{\mathcal{P}}_{\text{fast}}} + \frac{t^2}{2^\lambda}) \\
 &\leq \kappa_{\text{IP}}^{\text{sr}} \left(s, t, n, \delta_{\tilde{\mathcal{P}}_{\text{fast}}} + \frac{t^2}{2^\lambda} \right) \quad (\text{by Theorem 14.4.1}) .
 \end{aligned}$$

Finally, by combining the above equations we get

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xrightarrow{\text{tr}} \tilde{\mathcal{P}}_{\text{fast}}^f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}_{\text{fast}}^f(\mathbf{x}, \pi) \\ \mathbf{w} \leftarrow \mathcal{E}_{\text{fast}}(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}}_{\text{fast}}) \end{array} \right] \leq \kappa_{\text{IP}}^{\text{sr}} \left(s, t, n, \delta_{\tilde{\mathcal{P}}_{\text{fast}}} + \frac{t^2}{2^\lambda} \right) + \frac{t^2}{2^\lambda} .$$

We discuss the running time of $\mathcal{E}_{\text{fast}}$ on $\tilde{\mathcal{P}}_{\text{fast}}$. The extractor $\mathcal{E}_{\text{fast}}$ runs F2STrace on both traces in time $O(r_{\max} \cdot k \cdot t \cdot \log t)$, and then emulates the execution of $\mathcal{E}_{\text{slow}}$ on $\tilde{\mathcal{P}}_{\text{slow}} := \tilde{\mathcal{P}}_{\text{slow}}(\tilde{\mathcal{P}}_{\text{fast}})$. We deduce that

$$\mathbf{et}_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}_{\text{fast}}}, \tau_{\tilde{\mathcal{P}}_{\text{fast}}}) \leq \mathbf{et}_{\text{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathcal{P}}_{\text{slow}}}, \tau_{\tilde{\mathcal{P}}_{\text{slow}}}) + O(r_{\max} \cdot k \cdot t \cdot \log t) .$$

Since $\delta_{\tilde{\mathcal{P}}_{\text{slow}}} \leq \delta_{\tilde{\mathcal{P}}_{\text{fast}}} + \frac{t^2}{2^\lambda}$, $\tau_{\tilde{\mathcal{P}}_{\text{slow}}} \leq \tau_{\tilde{\mathcal{P}}_{\text{fast}}} + O(r_{\max} \cdot k \cdot t \cdot \log t)$, and using Theorem 14.4.1 we get that

$$\begin{aligned}
 & \mathbf{et}_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}_{\text{fast}}}, \tau_{\tilde{\mathcal{P}}_{\text{fast}}}) \\
 &\leq \mathbf{et}_{\text{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathcal{P}}_{\text{slow}}}, \tau_{\tilde{\mathcal{P}}_{\text{slow}}}) + O(r_{\max} \cdot k \cdot t \cdot \log t) \\
 &\leq \mathbf{et}_{\text{IP}}^{\text{sr}} \left(s, t, n, \delta_{\tilde{\mathcal{P}}_{\text{fast}}} + \frac{t^2}{2^\lambda}, \tau_{\tilde{\mathcal{P}}_{\text{fast}}} + O(r_{\max} \cdot k \cdot t \cdot \log t) \right) + O(r_{\max} \cdot k \cdot t \cdot \log t) .
 \end{aligned}$$

The straightline case The above analysis directly specializes to the straightline case. Suppose that $\mathbf{E}_{\text{IP}}^{\text{sr}}$ is a straightline extractor (Definition 13.2.3): it is a deterministic algorithm that does not need access to the IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$. Then, by Theorem 14.4.1, $\mathcal{E}_{\text{slow}}$ is a straightline extractor (Definition 7.1.5): it is a deterministic algorithm that does not need access to the argument prover $\tilde{\mathcal{P}}_{\text{slow}}$. In turn, $\mathcal{E}_{\text{fast}}$ in Construction 16.1.2 is a straightline extractor (Definition 7.1.5): it is a deterministic algorithm that does not need access to the argument prover $\tilde{\mathcal{P}}_{\text{fast}}$. In this case, the knowledge soundness error bound does not depend on the failure probability of the prover, and simplifies to $\kappa_{\text{ARG}}(\lambda, t, n) \leq \kappa_{\text{IP}}^{\text{sr}}(s, t, n) + \frac{t^2}{2^\lambda}$. Similarly, the extraction time bound does not depend on the failure probability or running time of the prover, and simplifies to $\mathbf{et}_{\text{ARG}}(\lambda, t, n) \leq \mathbf{et}_{\text{IP}}^{\text{sr}}(s, t, n) + O(r_{\max} \cdot k \cdot t \cdot \log t)$. \square

16.2 Zero knowledge

In Construction 15.1.1, if the public-coin IP satisfies honest-verifier zero knowledge (and the privacy parameter s of the construction is large enough) then the resulting non-interactive argument satisfies adaptive zero knowledge.

Theorem 16.2.1

Let IP be a public-coin IP for a relation \mathcal{R} with round complexity k and honest-verifier zero-knowledge error z_{IP} (see Definition 13.1.7). For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{NARG} := \mathbf{FS}_{\text{IP}}^{\star}[\text{IP}, \lambda, s]$ in Construction 15.1.1 is a non-interactive argument for \mathcal{R} with adaptive zero-knowledge error z_{ARG} (see Definition 7.1.8) such that

$$z_{\text{ARG}}(\lambda, t, n) \leq z_{\text{IP}}(n) + \frac{t}{2^s}.$$

Construction 16.2.2. The simulator is an algorithm $\mathcal{S}^f(\mathbf{x})$ that works as follows. Below we denote by \mathbf{S}_{IP} the honest-verifier zero-knowledge simulator of the IP (see Definition 13.1.7).

- $\mathcal{S}^f(\mathbf{x})$:

1. Sample a simulated view of the IP verifier: $(\mathbf{x}, (\rho_i)_{i \in [k]}, (\alpha_i)_{i \in [k]}) \leftarrow \mathbf{S}_{\text{IP}}(\mathbf{x})$.
2. For $i = 1, \dots, k$:
 - a) Sample a random salt $\tau_i \in \{0, 1\}^s$.
 - b) Set $x_i := \begin{cases} (\mathbf{x}, \alpha_1, \tau_1) & \text{if } i = 1 \\ (\rho_{i-1}, \alpha_i, \tau_i) & \text{if } i > 1 \end{cases}$.
 - c) Set the query-answer list $\mu_i := \{(x_i, \rho_i)\}$, to be used to program oracle f_i .
3. Set the argument string $\pi := ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]})$.
4. Set $\mu := (\mu_i)_{i \in [k]}$.
5. Output (π, μ) .

Note that \mathcal{S} programs the oracles $f = (f_i)_{i \in [k]}$ at one point each via the output μ (and does not query f).

Proof. Fix a query bound $t \in \mathbb{N}$, t -query admissible adversary \mathcal{A} , and instance bound $n \in \mathbb{N}$. We wish to upper bound the statistical distance between the output of \mathcal{A} in the real world

and in the simulated world. For that, we introduce a hybrid simulator (that takes the witness as input and programs the oracle): $\mathcal{S}_H^f(\mathbf{x}, \mathbf{w})$ acts as $\mathcal{S}^f(\mathbf{x})$ except that the view in Item 1 is sampled as a real view $(\mathbf{x}, (\rho_i)_{i \in [k]}, (\alpha_i)_{i \in [k]}) \leftarrow \text{View}_{\text{IP}}(\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}}, \mathbf{x}, \mathbf{w})$. We list the real-world, hybrid-world, and simulated-world distributions, making explicit the operations underlying the relevant algorithms.

1. The real-world distribution:

$$\mathcal{D}_{\text{real}} := \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ \pi \leftarrow \mathcal{P}^f(\mathbf{x}, \mathbf{w}) \\ \text{out} \leftarrow \mathcal{A}^f(\mathbf{aux}, \pi) \end{array} \right. \right\} \equiv \left\{ \text{out} \left| \begin{array}{l} f = (f_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ \text{For } i = 1, \dots, k: \\ \quad \bullet (\alpha_i, \mathbf{aux}_i) := \begin{cases} \mathbf{P}_{\text{IP}}(\mathbf{x}, \mathbf{w}) & \text{if } i = 1 \\ \mathbf{P}_{\text{IP}}(\mathbf{aux}_{i-1}, \rho_{i-1}) & \text{if } i > 1 \end{cases} \\ \quad \bullet \tau_i \leftarrow \{0, 1\}^s \\ \quad \bullet x_i := \begin{cases} (\mathbf{x}, \alpha_1, \tau_1) & \text{if } i = 1 \\ (\rho_{i-1}, \alpha_i, \tau_i) & \text{if } i > 1 \end{cases} \\ \quad \bullet \rho_i := f_i(x_i) \\ \pi := ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]}) \\ \text{out} \leftarrow \mathcal{A}^f(\mathbf{aux}, \pi) \end{array} \right. \right\} .$$

2. The hybrid-world distribution:

$$\mathcal{D}_H := \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ (\pi, \mu) \leftarrow \mathcal{S}_H^f(\mathbf{x}, \mathbf{w}) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right. \right\} \equiv \left\{ \text{out} \left| \begin{array}{l} f = (f_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ \text{For } i = 1, \dots, k: \\ \quad \bullet (\alpha_i, \mathbf{aux}_i) := \begin{cases} \mathbf{P}_{\text{IP}}(\mathbf{x}, \mathbf{w}) & \text{if } i = 1 \\ \mathbf{P}_{\text{IP}}(\mathbf{aux}_{i-1}, \rho_{i-1}) & \text{if } i > 1 \end{cases} \\ \quad \bullet \tau_i \leftarrow \{0, 1\}^s \\ \quad \bullet x_i := \begin{cases} (\mathbf{x}, \alpha_1, \tau_1) & \text{if } i = 1 \\ (\rho_{i-1}, \alpha_i, \tau_i) & \text{if } i > 1 \end{cases} \\ \quad \bullet \rho_i \leftarrow \{0, 1\}^{r_i} \\ \quad \bullet \mu_i := \{(x_i, \rho_i)\} \\ \mu := (\mu_i)_{i \in [k]} \\ \pi := ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]}) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right. \right\} .$$

3. The simulated-world distribution:

$$\mathcal{D}_{\text{sim}} := \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ (\pi, \mu) \leftarrow \mathcal{S}^f(\mathbf{x}) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right. \right\} \equiv \left\{ \text{out} \left| \begin{array}{l} f = (f_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ (\mathbf{x}, (\rho_i)_{i \in [k]}, (\alpha_i)_{i \in [k]}) \leftarrow \mathbf{S}_{\text{IP}}(\mathbf{x}) \\ \text{For } i = 1, \dots, k: \\ \quad \bullet \tau_i \leftarrow \{0, 1\}^s \\ \quad \bullet x_i := \begin{cases} (\mathbf{x}, \alpha_1, \tau_1) & \text{if } i = 1 \\ (\rho_{i-1}, \alpha_i, \tau_i) & \text{if } i > 1 \end{cases} \\ \quad \bullet \mu_i := \{(x_i, \rho_i)\} \\ \mu := (\mu_i)_{i \in [k]} \\ \pi := ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]}) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right. \right\} .$$

Next we analyze the statistical difference between these distributions.

- *Real versus hybrid.* We analyze the statistical distance between $\mathcal{D}_{\text{real}}$ and \mathcal{D}_{H} .

The two distributions only differ in that the hybrid simulator \mathcal{S}_{H} programs the oracles with freshly sampled IP challenges, whereas the argument prover \mathcal{P} does not program the oracles (it sets the IP challenges to be the answers of the random oracles to certain queries).

For every $i \in [k]$, the distribution of the query-answer pair (x_i, ρ_i) for the oracle f_i is the same in both cases. However, \mathcal{A} has access to the oracles $f = (f_i)_{i \in [k]}$ before they are programmed. Thus, \mathcal{A} can distinguish between the two distributions *only* if \mathcal{A} queries an oracle f_i at x_i before f_i is programmed there (and also after f_i is programmed).

Suppose that \mathcal{A} makes t_i queries to oracle f_i ; note that $\sum_{i \in [k]} t_i \leq t$. For every $i \in [k]$, the query x_i includes a salt $\tau_i \in \{0, 1\}^s$ that is sampled independently and uniformly at random. Hence, for every $i \in [k]$, the probability that \mathcal{A} queries f_i at x_i before τ_i is sampled is at most $\frac{t_i}{2^s}$. We conclude that the probability that there exists $i \in [k]$ for which this happens is at most

$$\sum_{i \in [k]} \frac{t_i}{2^s} \leq \frac{t}{2^s}.$$

In particular, this is an upper bound on the statistical distance between $\mathcal{D}_{\text{real}}$ and \mathcal{D}_{H} .

- *Hybrid versus simulated.* We analyze the statistical distance between \mathcal{D}_{H} and \mathcal{D}_{sim} .

The two distributions only differ in that \mathcal{S}_{H} samples a real IP view and \mathcal{S} samples a simulated IP view. By the zero-knowledge property of the IP, the statistical distance between the IP views for $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ is at most $z_{\text{IP}}(\mathbf{x})$. Since \mathcal{A} outputs $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ with $|\mathbf{x}| \leq n$ (as \mathcal{A} is admissible), the statistical distance between \mathcal{D}_{H} and \mathcal{D}_{sim} is upper bounded by $z_{\text{IP}}(n)$.

Adding these error terms yields the statistical difference claimed in the lemma. □

16.3 A security reduction from state-restoration

We have established the soundness and knowledge soundness of Construction 15.1.1 via a reduction to the soundness and knowledge soundness of Construction 14.1.1. Given a malicious argument prover $\tilde{\mathcal{P}}_{\text{fast}}$ against $\mathcal{V}_{\text{fast}}$, we construct a corresponding malicious argument prover $\tilde{\mathcal{P}}_{\text{slow}} := \tilde{\mathcal{P}}_{\text{slow}}(\tilde{\mathcal{P}}_{\text{fast}})$ against $\mathcal{V}_{\text{slow}}$. In turn, we have established the soundness and knowledge soundness of Construction 14.1.1 via a reduction to the state-restoration soundness and knowledge soundness of the underlying IP. Given a malicious argument prover $\tilde{\mathcal{P}}_{\text{slow}}$ against $\mathcal{V}_{\text{slow}}$, we construct a corresponding IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}} := \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}}_{\text{slow}})$. Each of these steps is accompanied by a “trace translation” function: F2STrue for the first step and FStoSR for the second step.

Below we state a lemma that captures the concatenation of these two steps, yielding a direct reduction from the security of Construction 15.1.1 to the security of the underlying IP. We rely on the lemma in Part VI, when constructing non-interactive arguments from IOPs. The statement of the lemma relies on the concatenation of the two malicious provers (given in Construction 16.3.1) and the concatenation of the two trace translation functions (given in Construction 16.3.2).

Construction 16.3.1. The IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ receives black-box access to an argument prover $\tilde{\mathcal{P}}_{\text{fast}}$, and is defined as follows:

$$\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}}_{\text{fast}}) := \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}}_{\text{slow}}(\tilde{\mathcal{P}}_{\text{fast}})).$$

Above, $\tilde{\mathcal{P}}_{\text{slow}}$ is the argument prover in Construction 15.2.6, and $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ is the IP state-restoration prover in Construction 14.3.2. If $\tilde{\mathcal{P}}_{\text{fast}}$ makes at most t queries to $f = (f_i)_{i \in [k]}$ then $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ makes at most t moves; moreover, the running time of $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ equals the running time of $\tilde{\mathcal{P}}_{\text{fast}}$ plus $O(r_{\max} \cdot k \cdot t \cdot \log t)$.

Construction 16.3.2. The function FastToSRTTrace receives as input a query-answer trace tr and outputs a move-response trace tr^{sr} computed as follows.

$\text{FastToSRTTrace}(\text{tr})$:

1. Compute $\text{tr}_{\text{slow}} := \text{F2STrace}(\text{tr})$ (see Construction 15.2.4).
2. Compute $\text{tr}^{\text{sr}} := \text{FStoSR}(\text{tr}_{\text{slow}})$ (see Construction 14.3.3).
3. Output tr^{sr} .

Note that if the query-answer trace of $\tilde{\mathcal{P}}_{\text{fast}}$ is tr then the move-response trace of $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}}_{\text{fast}})$ is tr^{sr} . The running time of FastToSRTTrace is $O(r_{\max} \cdot k \cdot t \cdot \log t)$.

Lemma 16.3.3. *The following two distributions are $\frac{t^2}{2^\lambda}$ -close in statistical distance:*

$$\left\{ (\text{tr}^{\text{sr}}, (\rho_i)_{i \in [k]}, \mathbf{x}, \pi, b) \middle| \begin{array}{l} f = (f_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}_{\text{fast}}^f \\ b \xleftarrow{\text{try}} \mathcal{V}_{\text{fast}}^f(\mathbf{x}, \pi) \\ \text{tr}^{\text{sr}} := \text{FastToSRTTrace}(\text{tr}) \\ \text{parse } \text{tr}_v \text{ as } (x_i, \rho_i)_{i \in [k]} \end{array} \right\}$$

and

$$\left\{ (\text{tr}^{\text{sr}}, (\rho_i)_{i \in [k]}, \mathbf{x}, \pi, b) \middle| \begin{array}{l} \text{rnd} = (\text{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}}_{\text{fast}})) \\ (\tau_i)_{i \in [k]} := (\sigma_i)_{i \in [k]} \\ \pi := ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]}) \\ b \leftarrow \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \end{array} \right\}.$$

Proof. By definition of FastToSRTTrace (Construction 16.3.2), the upper distribution in the lemma statement is equivalent to the following distribution:

$$\left\{ (\text{tr}^{\text{sr}}, (\rho_i)_{i \in [k]}, \mathbf{x}, \pi, b) \middle| \begin{array}{l} f = (f_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}_{\text{fast}}^f \\ b \xleftarrow{\text{try}} \mathcal{V}_{\text{fast}}^f(\mathbf{x}, \pi) \\ \text{tr}_{\text{slow}} := \text{F2STrace}(\text{tr}) \\ \text{tr}^{\text{sr}} := \text{FStoSR}(\text{tr}_{\text{slow}}) \\ \text{parse } \text{tr}_v \text{ as } (x_i, \rho_i)_{i \in [k]} \end{array} \right\}.$$

By Lemma 15.2.7 the above distribution is $\frac{t^2}{2^\lambda}$ -close in statistical distance to the following distribution:

$$\left\{ (\text{tr}^{\text{sr}}, (\rho_i)_{i \in [k]}, \mathbf{x}, \pi, b) \middle| \begin{array}{l} f = (f_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr}_{\text{fast}}]{} \tilde{\mathcal{P}}_{\text{slow}}^f \\ b \leftarrow \mathcal{V}_{\text{slow}}^f(\mathbf{x}, \pi) \\ \text{tr}_{\text{slow}} := \text{F2STrace}(\text{tr}) \\ \text{tr}^{\text{sr}} := \text{FStoSR}(\text{tr}_{\text{slow}}) \\ \text{tr}_v \leftarrow \mathcal{V}_{\text{slow}} \text{ queries of } \text{tr}_{\text{fast}} \\ \text{parse } \text{tr}_v \text{ as } (x_i, \rho_i)_{i \in [k]} \end{array} \right\}.$$

By definition of $\tilde{\mathcal{P}}_{\text{slow}}$ (Construction 15.2.6) and of F2STrace (Construction 15.2.4), the above distribution is equivalent to the following distribution:

$$\left\{ (\text{tr}^{\text{sr}}, (\rho_i)_{i \in [k]}, \mathbf{x}, \pi, b) \middle| \begin{array}{l} f = (f_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xleftarrow[\text{tr}_{\text{slow}}]{} \tilde{\mathcal{P}}_{\text{slow}}^f \\ b \leftarrow \mathcal{V}_{\text{slow}}^f(\mathbf{x}, \pi) \\ \text{tr}^{\text{sr}} := \text{FStoSR}(\text{tr}_{\text{slow}}) \\ \text{tr}_v \leftarrow \mathcal{V}_{\text{slow}} \text{ queries of } \text{tr}_{\text{fast}} \\ \text{parse } \text{tr}_v \text{ as } (x_i, \rho_i)_{i \in [k]} \end{array} \right\}.$$

By Equation 14.1, the above distribution is equivalent to the following distribution (which is the lower distribution in the lemma statement):

$$\left\{ (\text{tr}^{\text{sr}}, (\rho_i)_{i \in [k]}, \mathbf{x}, \pi, b) \middle| \begin{array}{l} \text{rnd} = (\text{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow[\text{tr}^{\text{sr}}]{} \text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}}_{\text{slow}})) \\ (\tau_i)_{i \in [k]} := (\sigma_i)_{i \in [k]} \\ \pi := ((\alpha_i)_{i \in [k]}, (\tau_i)_{i \in [k]}) \\ b \leftarrow \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \end{array} \right\}.$$

□

Part IV

Commitment Schemes

Overview

Succinct arguments in the random oracle model (ROM) rely on suitable commitment schemes in the ROM, which we study in detail in this part. First we study the security properties of a basic commitment scheme in the ROM: the commitment is the output of the random oracle when given as input the message and a random salt. This commitment scheme is only a warmup because it lacks a key property used for succinct arguments: the ability to cheaply open a subset of entries of the message (as opposed to only being able to open the entire message). Then we study Merkle commitment schemes in the ROM, which have this “succinct opening” capability.

17 Basic commitment scheme	153
17.1 Binding	154
17.2 Extractability	155
17.3 Hiding	161
18 Merkle commitment scheme	166
18.1 Definition	166
18.2 Completeness	171
18.3 Collision lemma	172
18.4 Binding	176
18.5 Extractability	178
18.6 Hiding	189

17 Basic commitment scheme

A (non-interactive) commitment scheme enables a party to commit to a message m to obtain a commitment cm in such a way that: (a) the committing party can subsequently “open” the commitment cm to the message m ; (b) it is intractable to open cm to a different message m' ; and (c) the commitment cm reveals no information about the message m . Commitment schemes are a fundamental cryptographic primitive, and in particular they play a key role in the construction of succinct arguments in the random oracle model.

Here we describe a basic commitment scheme obtained from the random oracle, which we denote by CM . This serves two purposes: (1) analyzing the security properties of CM is a useful warm up to more sophisticated constructions in the random oracle model; and (2) CM is an ingredient to the construction of a Merkle commitment scheme MT (studied in Chapter 18), which is used to construct succinct arguments.

The basic commitment scheme is a tuple of algorithms

$$CM = (CM.\text{Commit}, CM.\text{Check})$$

and has several parameters: an output size $\sigma \in \mathbb{N}$ of the random oracle, a message length $\ell \in \mathbb{N}$, and a salt size $s \in \mathbb{N}$. We use the notation $CM[\sigma, \ell, s]$ when we wish to make these parameters explicit. The algorithms receive query access to a random oracle $f \in \mathcal{U}(\sigma)$ and work as follows.

- *Commit.* The algorithm $CM.\text{Commit}$ receives as input a message $m \in \{0, 1\}^\ell$, samples a random salt $\tau \in \{0, 1\}^s$, queries the random oracle f to compute the commitment $cm := f(m, \tau) \in \{0, 1\}^\sigma$, and outputs the pair (cm, τ) .
- *Check.* The algorithm $CM.\text{Check}$ receives as input a commitment cm , message $m \in \{0, 1\}^\ell$, and salt $\tau \in \{0, 1\}^s$, and queries the random oracle to check that $cm = f(m, \tau)$.

See Figure 17.1 for a diagram.

Organization We prove several properties about the basic commitment scheme CM .

- In Section 17.1 we prove that CM is *binding*.
- In Section 17.2 we prove that CM is *extractable*.
- In Section 17.3 we prove that CM is *hiding*.

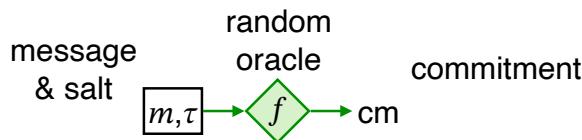


Figure 17.1: Diagram of the basic commitment scheme CM .

17.1 Binding

We prove that CM is *binding*: an adversary cannot find two distinct messages that open the same commitment. In more detail, we consider an experiment where the adversary, given access to a random oracle f , is challenged to output a commitment cm and two message-salt pairs (\mathbf{m}_0, τ_0) and (\mathbf{m}_1, τ_1) such that: (i) the messages \mathbf{m}_0 and \mathbf{m}_1 are distinct (and the salts τ_0 and τ_1 may or may not be distinct); and (ii) $\text{CM.Check}^f(\text{cm}, \mathbf{m}_0, \tau_0) = 1$ and $\text{CM.Check}^f(\text{cm}, \mathbf{m}_1, \tau_1) = 1$. The maximum winning probability of an adversary in this experiment is the *binding error* of CM.

Breaking the binding property of CM is related to finding collisions in the random oracle, because the aforementioned condition of opening the same commitment via two different messages is equivalent to $f(\mathbf{m}_0, \tau_0) = f(\mathbf{m}_1, \tau_1)$ for $\mathbf{m}_0 \neq \mathbf{m}_1$.

By the discussion on finding collisions in Section 3.3, the binding error is $\Omega(\frac{t^2}{2^\sigma})$: the collision-finding strategy in the proof of Claim 3.3.2 can be modified in a straightforward way to find, with probability $\Omega(\frac{t^2}{2^\sigma})$, (\mathbf{m}_0, τ_0) and (\mathbf{m}_1, τ_1) such that $\mathbf{m}_0 \neq \mathbf{m}_1$ and $f(\mathbf{m}_0, \tau_0) = f(\mathbf{m}_1, \tau_1)$. On the other hand, the lemma below states that any t -query algorithm can break the binding property with probability at most $O(\frac{t^2}{2^\sigma})$, where the probability is taken over the choice of random oracle. The proof relies on collision resistance of the random oracle, as well as on unpredictability of the random oracle for an edge case in the analysis. Note that the salt size s of CM does *not* play a role in this upper bound, and in particular it can be zero.

Lemma 17.1.1 (CM is binding). *Let $\text{CM} := \text{CM}[\sigma, \ell, s]$. For every query bound $t \in \mathbb{N}$ and t -query algorithm A (that outputs a commitment cm , messages $\mathbf{m}_0, \mathbf{m}_1$, and salts τ_0, τ_1), if $t \geq 2$ then*

$$\Pr \left[\begin{array}{l} \mathbf{m}_0 \neq \mathbf{m}_1 \\ \wedge \text{CM.Check}^f(\text{cm}, \mathbf{m}_0, \tau_0) = 1 \\ \wedge \text{CM.Check}^f(\text{cm}, \mathbf{m}_1, \tau_1) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\text{cm}, \mathbf{m}_0, \tau_0, \mathbf{m}_1, \tau_1) \leftarrow A^f \end{array} \right] \leq \frac{1}{2} \cdot \frac{t^2}{2^\sigma}.$$

Proof. The condition in the probability statement is, by definition of CM.Check, equivalent to

$$\mathbf{m}_0 \neq \mathbf{m}_1 \text{ and } f(\mathbf{m}_0, \tau_0) = f(\mathbf{m}_1, \tau_1).$$

In particular, (\mathbf{m}_0, τ_0) and (\mathbf{m}_1, τ_1) form a collision for f .

Let E be the event that A makes the queries (\mathbf{m}_0, τ_0) and (\mathbf{m}_1, τ_1) to f (i.e., the query-answer trace of A contains (\mathbf{m}_0, τ_0) and (\mathbf{m}_1, τ_1) as queries). We distinguish between two cases.

- *Case 1: A wins the binding game and E does not hold.* In this case A outputs (\mathbf{m}_0, τ_0) and (\mathbf{m}_1, τ_1) that are a collision and either: (i) neither pair was queried, or (ii) exactly one pair was queried. By the unpredictability property of a random oracle (Lemma 3.1.1), the probability that either happens is $\frac{1}{2^\sigma}$ because, in either case, A outputs a string that is not part of the query-answer trace tr and the random oracle f maps to a certain output.
- *Case 2: A wins the binding game and E holds.* In this case the query-answer trace tr of A contains a collision, because (\mathbf{m}_0, τ_0) and (\mathbf{m}_1, τ_1) are contained in tr and satisfy $f(\mathbf{m}_0, \tau_0) = f(\mathbf{m}_1, \tau_1)$. By Lemma 3.3.1 the probability that tr contains a collision is at most $\frac{1}{2} \cdot \frac{(t-1) \cdot t}{2^\sigma}$.

We conclude that

$$\Pr \left[\begin{array}{l} A \text{ wins the} \\ \text{binding game} \end{array} \right]$$

$$\begin{aligned}
&= \Pr \left[\begin{array}{l} A \text{ wins the} \\ \text{binding game} \end{array} \wedge E \right] + \Pr \left[\begin{array}{l} A \text{ wins the} \\ \text{binding game} \end{array} \wedge \bar{E} \right] \\
&\leq \frac{1}{2} \cdot \frac{(t-1) \cdot t}{2^\sigma} + \frac{1}{2^\sigma} \\
&= \frac{1}{2} \cdot \frac{t^2 - t + 2}{2^\sigma}.
\end{aligned}$$

Finally, if $t \geq 2$ then the last expression above is at most $\frac{1}{2} \cdot \frac{t^2}{2^\sigma}$. \square

Remark 17.1.2 (statistical binding). The binding property of CM in Lemma 17.1.1 is *computational* because the binding error $\frac{1}{2} \cdot \frac{t^2}{2^\sigma}$ is small only if the adversary makes a bounded number of queries t to the random oracle. However, if the output size σ of the random oracle is large enough, then CM achieves *statistical* binding, namely, a binding property that holds even against adversaries that make an unbounded number of queries to the random oracle (think of $t = \infty$). The statistical security comes from the simple fact that, if σ is large enough, the commitment scheme CM is *injective* with high probability over the choice of random oracle, in which case each commitment has a single preimage (and thus “collisions” do not exist).

To see this, we upper bound the probability that the commitment scheme CM is not injective:

$$\begin{aligned}
&\Pr \left[\begin{array}{l} \exists m_0, m_1 \in \{0, 1\}^\ell \text{ with } m_0 \neq m_1 \\ \exists \tau_0, \tau_1 \in \{0, 1\}^s \end{array} \text{ s.t. } \text{CM.Commit}(m_0, \tau_0) = \text{CM.Commit}(m_1, \tau_1) \right] \\
&\leq 2^{2 \cdot (\ell+s)} \cdot \Pr[\text{CM.Commit}(0^\ell, 0^s) = \text{CM.Commit}(1^\ell, 0^s)] \\
&= 2^{2 \cdot (\ell+s)} \cdot \frac{1}{2^\sigma} \\
&= \frac{1}{2^{\sigma - 2 \cdot (\ell+s)}}.
\end{aligned}$$

We conclude that the probability that CM is injective is at least $1 - \frac{1}{2^{\sigma - 2 \cdot (\ell+s)}}$, which is close to 1 if $\sigma \gg 2 \cdot (\ell + s)$. This regime of parameters is, however, not relevant for the setting of succinct arguments (where we need the commitment size σ to be much smaller than the message size ℓ), and so we limit the discussion of statistical binding to this remark.

Note that, conversely, whenever the commitment scheme CM is not injective (with respect to messages), an adversary that makes sufficiently many queries to the random oracle can indeed find distinct messages m_0 and m_1 such that $\text{CM.Commit}(m_0, \tau_0) = \text{CM.Commit}(m_1, \tau_1)$ for some salts τ_0 and τ_1 , and thereby break the binding property.

17.2 Extractability

We prove that CM is *extractable*, which informally means that if an adversary outputs a commitment that it subsequently opens then the adversary “knew” the opening at commitment time. In Section 17.2.1 we study this property for one commitment and in Section 17.2.2 we study it for multiple commitments; then in Section 17.2.3 we explain how extractability implies binding.

17.2.1 Extraction for one commitment

The extractability property is formalized via an efficient algorithm CM.Extract called an *extractor*. We consider an experiment of the following form.

- In a commitment phase, the adversary, given oracle access to the random oracle, performs a computation and outputs a commitment cm and a private state aux . Let tr be the query-answer trace of the adversary in this phase.
- Then, in an opening phase, the adversary, given oracle access to the random oracle and as input the private state aux , continues its computation and outputs a message m and salt τ .

Extractability states that if the adversary's output message m and salt τ are a valid opening of the previously-output commitment cm then the extractor CM.Extract , given the commitment cm and the query-answer trace tr , also outputs the same message m and salt τ , up to a small error. In other words, the adversary "knew" the opening (cm, τ) in the commitment phase because CM.Extract finds (cm, τ) by examining only the query-answer trace of the adversary relevant for producing the commitment (without seeing the subsequent query-answer trace in the opening phase).

The lemma below formally states the property and bounds the extraction error, which happens to be the same as the binding error in Lemma 17.1.1.

Lemma 17.2.1 (CM is extractable). *Let $\text{CM} := \text{CM}[\sigma, \ell, s]$. There exists a polynomial-time deterministic algorithm CM.Extract such that for every query bound $t \in \mathbb{N}$ and t -query algorithm A , if $t \geq 3$ then*

$$\Pr \left[\begin{array}{l} \text{CM.Check}^f(\text{cm}, \text{m}, \tau) = 1 \\ \wedge (\text{m}', \tau') \neq (\text{m}, \tau) \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\text{cm}, \text{aux}) \xleftarrow{\text{tr}} A^f \\ (\text{m}', \tau') \leftarrow \text{CM.Extract}(\text{cm}, \text{tr}) \\ (\text{m}, \tau) \leftarrow A^f(\text{aux}) \end{array} \right] \leq \frac{1}{2} \cdot \frac{t^2}{2^\sigma}.$$

Moreover, CM.Extract runs in time $O(t \cdot (\ell + s + \sigma))$.

Proof. We define the extractor CM.Extract as follows.

$\text{CM.Extract}(\text{cm}, \text{tr})$: Search the trace tr for a query-answer pair of the form $((\text{m}', \tau'), \text{cm})$, where $\text{m}' \in \{0, 1\}^\ell$ is a message and $\tau' \in \{0, 1\}^s$ is a salt. If no such pair exists in tr then abort. If such a pair is found then output (m', τ') .

The extractor has the stated time complexity because it scans a query-answer trace consisting of at most t entries, in order to search for a $(\ell + s + \sigma)$ -bit entry where the answer equals cm .

We argue that the extractor satisfies the statement.

Fix any $t_1, t_2 \in \mathbb{N}$ such that $t_1 + t_2 \leq t$. Let tr' be the query-answer trace of the computation $(\text{m}, \tau) \leftarrow A^f(\text{aux})$. We prove the upper bound of $\frac{1}{2} \cdot \frac{t^2}{2^\sigma}$ conditioned on the event that $t_1 = |\text{tr}|$ and $t_2 = |\text{tr}'|$. Then the lemma follows because this upper bound holds for every $t_1, t_2 \in \mathbb{N}$ such that $t_1 + t_2 \leq t$, and every computation of the t -query adversary A implies a setting of t_1, t_2 .

If $\text{CM.Check}^f(\text{cm}, \text{m}, \tau) = 1$ and $\text{CM.Extract}(\text{cm}, \text{tr})$ outputs (m', τ') such that $(\text{m}', \tau') \neq (\text{m}, \tau)$ then one of the following cases must hold.

- There are two or more queries in the query-answer trace tr whose answer is cm . By Lemma 3.3.1 the probability that this happens is at most $\frac{1}{2} \cdot \frac{(t_1-1) \cdot t_1}{2^\sigma}$.
- There exists a query (m_j, τ_j) in tr' whose answer is cm that is not a query in tr . For every query (m_j, τ_j) in tr' , the probability that $f(\text{m}_j, \tau_j) = \text{cm}$ is $\frac{1}{2^\sigma}$. Hence by a union bound the probability that there exists such a query is at most $t_2 \cdot \frac{1}{2^\sigma}$.

- The adversary A outputs (\mathbf{m}, τ) such that $f(\mathbf{m}, \tau) = \mathbf{cm}$ but $((\mathbf{m}, \tau), \mathbf{cm})$ is not a query-answer pair in \mathbf{tr} or \mathbf{tr}' . By Lemma 3.1.1, the probability that this happens is at most $\frac{1}{2^\sigma}$.

We conclude that the extraction error is at most

$$\begin{aligned}
& \frac{1}{2} \cdot \frac{(t_1 - 1) \cdot t_1}{2^\sigma} + t_2 \cdot \frac{1}{2^\sigma} + \frac{1}{2^\sigma} \\
&= \frac{1}{2} \cdot \frac{t_1^2 - t_1 + 2t_2 + 2}{2^\sigma} \\
&\leq \frac{1}{2} \cdot \frac{t_1^2 - t_1 + 2(t - t_1) + 2}{2^\sigma} \quad (\text{since } t_1 + t_2 \leq t) \\
&= \frac{1}{2} \cdot \frac{t_1^2 - 3t_1 + 2t + 2}{2^\sigma} \\
&\leq \frac{1}{2} \cdot \frac{\max\{2t + 2, t^2 - t + 2\}}{2^\sigma}.
\end{aligned}$$

The last inequality is due to the fact that the expression is a convex function in t_1 so its maximum on the interval $t_1 \in \{0, 1, \dots, t\}$ is achieved at either $t_1 = 0$ or $t_1 = t$.

Finally, if $t \geq 3$ then the last expression above is at most $\frac{1}{2} \cdot \frac{t^2 - t + 2}{2^\sigma}$ (in the maximum the first term is at most the second term), which in turn is at most $\frac{1}{2} \cdot \frac{t^2}{2^\sigma}$. \square

17.2.2 Extraction for multiple commitments

We discuss how extractability extends to adversaries that output *multiple* commitments. This serves as a warm up to similar properties that we study for Merkle commitments in Chapter 18, which we use to analyze constructions of succinct arguments that involve multiple commitments.

We wish to prove that for any commitment that the adversary subsequently opens successfully, the extractor is able to find a corresponding opening while only given the query-answer trace of the adversary up to the point of producing that commitment. (We cannot expect to say anything about commitments that the adversary outputs but does not open successfully.)

We consider an experiment of the following form.

- In a commitment phase, the adversary, given oracle access to the random oracle, performs a stateful computation during which it outputs n commitments $\mathbf{cm}_1, \dots, \mathbf{cm}_n$. Let \mathbf{aux}_i be the private state of the adversary when it outputs \mathbf{cm}_i , and let \mathbf{tr}_i be the query-answer trace of the adversary between outputting the $(i-1)$ -th commitment and the i -th commitment.
- Then, in an opening phase, the adversary, given oracle access to the random oracle and as input the private state \mathbf{aux}_n , concludes its computation and outputs n message-salt pairs $((\mathbf{m}_i, \tau_i))_{i \in [n]}$.

We wish to prove that, for every $i \in [n]$, if the message \mathbf{m}_i and salt τ_i are a valid opening of the previously-output commitment \mathbf{cm}_i then an extractor $\mathbf{CM}.\mathbf{MultiExtract}$, given the commitment \mathbf{cm}_i and the query-answer trace $\mathbf{tr}_1 \| \dots \| \mathbf{tr}_i$ (all the query-answer pairs leading to outputting \mathbf{cm}_i), *also* outputs the same message \mathbf{m}_i and salt τ_i , up to a small error.

Intuitively, such a statement should follow directly from Lemma 17.2.1 via a union bound: if the adversary outputs n commitments and subsequently n openings, then the probability

that there exists one of the openings that is valid but the extractor did not succeed for the corresponding commitment is at most n times the error in Lemma 17.2.1.

Perhaps surprisingly, the multiplicative loss of n can be avoided. The dominant error incurred in extraction is linked to a global event rather than an event specific to one or another commitment. Indeed, from the proof of Lemma 17.2.1, extraction for a particular commitment fails if the query-answer trace either contains multiple queries whose answer is the commitment (the adversary found a collision) or does not contain a query whose answer is the commitment (the adversary guessed the answer). The former event is global and leads to the dominant error, while the latter event is specific to a single commitment and hence is incurred once per commitment.

This leads to a lemma with the error bound below. We let $\text{CM}.\text{MultiExtract}$ be a *stateful* algorithm; hence $\text{CM}.\text{MultiExtract}$ may store information from prior invocations and so it suffices, in its i -th invocation, to receive only the commitment and query-answer trace corresponding to the i -th output of the adversary. Maintaining state across invocations also enables efficiency improvements.

Lemma 17.2.2 (CM is multi-extractable). *Let $\text{CM} := \text{CM}[\sigma, \ell, s]$. There exists a polynomial-time deterministic **stateful** algorithm $\text{CM}.\text{MultiExtract}$ such that for every query bound $t \in \mathbb{N}$, t -query algorithm A , and number of commitments $n \in \mathbb{N}$, if $t \geq 2n$ then*

$$\Pr \left[\begin{array}{l} \exists i \in [n] \text{ such that:} \\ \text{CM.Check}^f(\text{cm}_i, \text{m}_i, \tau_i) = 1 \\ \wedge (\text{m}'_i, \tau'_i) \neq (\text{m}_i, \tau_i) \end{array} \right] \leq \frac{1}{2} \cdot \frac{t^2}{2^\sigma} + \frac{n \cdot t}{2^\sigma} \quad \left| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \text{For } i = 1, \dots, n: \\ \quad \bullet (\text{cm}_i, \text{aux}_i) \xleftarrow{\text{tr}_i} A^f(\text{aux}_{i-1}) \\ \quad \bullet (\text{m}'_i, \tau'_i) \leftarrow \text{CM}.\text{MultiExtract}(\text{cm}_i, \text{tr}_i) \\ ((\text{m}_i, \tau_i))_{i \in [n]} \leftarrow A^f(\text{aux}_n) \end{array} \right]$$

Above we denote by aux_0 the empty string for notational simplicity.

Moreover, $\text{CM}.\text{MultiExtract}$ runs in time $O(n \cdot t \cdot (\ell + s + \sigma))$ (across the n invocations).

We provide an intuitive justification for the error expression above, for simplicity for the case with no salts ($s = 0$). Consider the following adversary. For the first $n - 1$ commitments, the adversary performs no queries and simply outputs random commitment strings $(\text{cm}_i)_{i \in [n-1]}$. Then the adversary performs t distinct queries, using all of its query budget in the last trace tr_n . If there exists two distinct queries x_1 and x_2 such that $f(x_1) = f(x_2)$, then the adversary outputs $\text{cm}_n := f(x_1)$ and wins (the adversary can open cm_n in two ways). This explains the first term of the error bound. Otherwise, if there exists a query x with a response $y \in \{\text{cm}_i\}_{i \in [n-1]}$, then the adversary also wins (if $y = \text{cm}_i$ then the extractor fails on cm_i because it runs given an empty trace while the adversary can open cm_i). This explains the second term in the error bound, as each query has a probability of $\frac{n-1}{2^\sigma}$ to hit a previous commitment.

Proof. The stateful extractor $\text{CM}.\text{MultiExtract}$ is similar to the (stateless) extractor $\text{CM}.\text{Extract}$ in the proof of Lemma 17.2.1. The extractor $\text{CM}.\text{MultiExtract}$ internally maintains the concatenation of all query-answer traces received so far (after the i -th invocation it stores the trace $\text{tr} := \text{tr}_1 \parallel \dots \parallel \text{tr}_i$), and uses this trace to search for the message and salt corresponding to the given commitment.

$\text{CM}.\text{MultiExtract}(\text{cm}_i, \text{tr}_i)$:

1. Append tr_i to the query-answer trace tr stored in the internal state.
2. Run CM.Extract on input (cm_i, tr) . (Search the trace tr for a query-answer pair of the form $((\text{m}'_i, \tau'_i), \text{cm}_i)$, where $\text{m}'_i \in \{0, 1\}^\ell$ is a message and $\tau'_i \in \{0, 1\}^s$ is a salt. If no such pair exists in tr then abort. If such a pair is found then output (m'_i, τ'_i) .)

By Lemma 17.2.1 each invocation of CM.Extract runs in time at most $O(t \cdot (\ell + s + \sigma))$, because the stored query-answer trace contains at most t entries. The n invocations yield the claimed total running time. The time complexity can be improved via a suitable use of dynamic data structures in the internal state (and foregoing CM.Extract as a subroutine); see Remark 17.2.3.

We argue that the extractor satisfies the statement.

Fix any $t_1, t_2 \in \mathbb{N}$ such that $t_1 + t_2 \leq t$. Let tr' be the query-answer trace of the computation $((\text{m}_i, \tau_i))_{i \in [n]} \leftarrow A^f(\text{aux}_n)$. We prove the upper bound of $\frac{1}{2} \cdot \frac{t^2}{2^\sigma} + \frac{n \cdot t}{2^\sigma}$ conditioned on the event that $t_1 = |\text{tr}_1| \dots |\text{tr}_n|$ and $t_2 = |\text{tr}'|$. Then the lemma follows because this upper bound holds for every $t_1, t_2 \in \mathbb{N}$ such that $t_1 + t_2 \leq t$, and every computation of the t -query adversary A implies a setting of t_1, t_2 .

If there exists $i \in [n]$ such that $\text{CM.Check}^f(\text{cm}_i, \text{m}_i, \tau_i) = 1$ and $\text{CM.MultiExtract}(\text{cm}_i, \text{tr}_i)$ outputs (m'_i, τ'_i) such that $(\text{m}'_i, \tau'_i) \neq (\text{m}_i, \tau_i)$ then one of the following cases must hold.

- *Collision.* The adversary finds a collision in the trace. That is, there are two or more queries in the query-answer trace $\text{tr}_1 \parallel \dots \parallel \text{tr}_n$ with the same answer (in particular, whose answer is one of $\{\text{cm}_i\}_{i \in [n]}$). By Lemma 3.3.1 the probability that this happens is at most $\frac{1}{2} \cdot \frac{(t_1 - 1) \cdot t_1}{2^\sigma}$.
- *Inversion.* The adversary inverts one of the previously given commitments. That is, there exists $i \in [n]$ and a query (m, τ) in $\text{tr}_{i+1} \parallel \dots \parallel \text{tr}_n \parallel \text{tr}'$ whose answer is cm_i that is not a query in $\text{tr}_1 \parallel \dots \parallel \text{tr}_i$. For every query (m, τ) in $\text{tr}_{i+1} \parallel \dots \parallel \text{tr}_n \parallel \text{tr}'$, the probability that $f(\text{m}, \tau) = \text{cm}_i$ is at most $\frac{1}{2^\sigma}$. Hence by a union bound over $i \in [n]$ and all queries, the probability that there exists such an $i \in [n]$ and a query is at most $(t_1 + t_2) \cdot \frac{n}{2^\sigma}$.
- *Pure luck.* The adversary is lucky, and CM.Check accepts even though no collisions or inversions were found. That is, there exists $i \in [n]$ such that the adversary A outputs (m, τ) with $f(\text{m}, \tau) = \text{cm}_i$ but $((\text{m}, \tau), \text{cm}_i)$ is not a query-answer pair in $\text{tr}_1 \parallel \dots \parallel \text{tr}_n \parallel \text{tr}'$. By Lemma 3.1.1, the probability that this happens is at most $\frac{1}{2^\sigma}$. Taking a union bound over $i \in [n]$, the probability there exists such an $i \in [n]$ is at most $n \cdot \frac{1}{2^\sigma}$.

We conclude that the extraction error is at most

$$\begin{aligned} & \frac{1}{2} \cdot \frac{(t_1 - 1) \cdot t_1}{2^\sigma} + (t_1 + t_2) \cdot \frac{n}{2^\sigma} + \frac{n}{2^\sigma} \\ &= \frac{1}{2} \cdot \frac{t_1^2 - t_1 + 2n}{2^\sigma} + (t_1 + t_2) \cdot \frac{n}{2^\sigma} \\ &\leq \frac{1}{2} \cdot \frac{t^2 - t + 2n}{2^\sigma} + t \cdot \frac{n}{2^\sigma}. \end{aligned}$$

Finally, if $t \geq 2n$ then the last expression above is at most $\frac{1}{2} \cdot \frac{t^2}{2^\sigma} + \frac{n \cdot t}{2^\sigma}$ (itself at most $\frac{t^2}{2^\sigma}$). \square

Remark 17.2.3 (time complexity of CM.MultiExtract). The running time of CM.MultiExtract stated in Lemma 17.2.2 can be improved, by maintaining a *sorted* trace in order to support fast searching. At the start, CM.MultiExtract initializes an empty search tree that supports inserts

and searches in logarithmic time complexity (e.g., an AVL tree). At the i -th invocation of $\text{CM}.\text{MultiExtract}$, given input $(\text{cm}_i, \text{tr}_i)$, do the following: insert into the search tree the query-answer pairs of tr_i one by one; then search the search tree for a query-answer pair of the form $((\mathbf{m}'_i, \tau'_i), \text{cm}_i)$ (and abort if no such pair is found). The running time of this implementation is $O((t + n) \cdot \log t \cdot (\ell + s + \sigma))$ (the multiplicative term $(t + n) \cdot \log t$ replaces $n \cdot t$), as we now explain. Each element in the trace consists of $O(\ell + s + \sigma)$ bits, and there are at most t elements. Hence, each insert or search operation takes time $O((\ell + s + \sigma) \cdot \log t)$. The total number of insert and search operations is $t + n$ (t inserts and n searches), yielding the claimed time.

17.2.3 Extractability implies binding

The extractability property in Lemma 17.2.1 is a strong security guarantee: any bounded-query adversary that opens a commitment must have “known” the opening at commitment time. In fact, *extractability is stronger than binding*: we show that if CM satisfies extractability with error κ_{CM} then CM satisfies binding with error $\epsilon_{\text{CM}} \leq 2 \cdot \kappa_{\text{CM}}$ (a closely related error).¹

Let A be a t -query algorithm that opens a commitment in two ways with probability ϵ_{CM} :

$$\epsilon_{\text{CM}} := \Pr \left[\begin{array}{l} \mathbf{m}_0 \neq \mathbf{m}_1 \\ \wedge \text{CM}.\text{Check}^f(\text{cm}, \mathbf{m}_0, \tau_0) = 1 \\ \wedge \text{CM}.\text{Check}^f(\text{cm}, \mathbf{m}_1, \tau_1) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\text{cm}, \mathbf{m}_0, \tau_0, \mathbf{m}_1, \tau_1) \leftarrow A^f \end{array} \right].$$

Let A_e be the t -query algorithm for the extraction game that is defined as follows.

- A_e^f : Run A^f to get $(\text{cm}, \mathbf{m}_0, \tau_0, \mathbf{m}_1, \tau_1)$, set $\text{aux} := (\mathbf{m}_0, \tau_0, \mathbf{m}_1, \tau_1)$, and output (cm, aux) .
- $A_e^f(\text{aux})$: Sample a random bit b (by using private randomness or via a fresh additional query to the random oracle) and output the message and salt (\mathbf{m}_b, τ_b) stored in aux .

Fix any candidate extractor $\text{CM}.\text{Extract}$. Consider an output $(\text{cm}, \mathbf{m}_0, \tau_0, \mathbf{m}_1, \tau_1)$ of A that breaks the binding property (i.e., satisfies the conditions in the probability statement above). If we run $\text{CM}.\text{Extract}$ on cm (the commitment output by A_e in the first step) and tr (the query-answer trace by A_e in the first step) then we obtain an output (\mathbf{m}, τ) that, with probability at least $1/2$, satisfies $(\mathbf{m}, \tau) \neq (\mathbf{m}_b, \tau_b)$. This is because the inputs to the extractor $\text{CM}.\text{Extract}$ are independent of the bit b (which is sampled by A_e after the inputs to $\text{CM}.\text{Extract}$ are determined). Moreover, by the definition of A_e , we know that $\text{CM}.\text{Check}^f(\text{cm}, \mathbf{m}_b, \tau_b) = 1$ regardless of the choice of bit b . We deduce that

$$\frac{1}{2} \cdot \epsilon_{\text{CM}} \leq \Pr \left[\begin{array}{l} \text{CM}.\text{Check}^f(\text{cm}, \mathbf{m}_b, \tau_b) = 1 \\ \wedge (\mathbf{m}, \tau) \neq (\mathbf{m}_b, \tau_b) \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\text{cm}, \text{aux}) \xleftarrow{\text{tr}} A_e^f \\ (\mathbf{m}, \tau) \leftarrow \text{CM}.\text{Extract}(\text{cm}, \text{tr}) \\ (\mathbf{m}_b, \tau_b) \leftarrow A_e^f(\text{aux}) \end{array} \right].$$

We conclude that any extraction error κ_{CM} that can be proved must satisfy the relation $\frac{1}{2} \cdot \epsilon_{\text{CM}} \leq \kappa_{\text{CM}}$ which gives us the desired bound on the binding error ϵ_{CM} .

¹This implication loses a factor of 2, and we present it merely for didactic purposes. Indeed, for CM we prove almost the same errors in Lemma 17.1.1 (binding error) and Lemma 17.2.1 (extraction error).

17.3 Hiding

We prove that CM is *hiding*, which informally means that a commitment cm computed as $f(\mathbf{m}, \tau)$ for a random $\tau \in \{0, 1\}^s$ reveals no information about \mathbf{m} , up to a small statistical “leakage” error.

In more detail, we compare the output of an adversary $A^f(\text{cm})$ in two cases:

- cm is computed as $f(\mathbf{m}, \tau)$;
- cm is output by a probabilistic algorithm $\text{CM}.\text{Simulate}$ that is only given oracle access to f (and in particular does not know the message \mathbf{m}).

The hiding property requires that the outputs of the adversary in these two cases are statistically close. Moreover, the adversary may choose the message \mathbf{m} by querying the random oracle f .

The statistical distance depends, in general, on several parameters: (a) the output size $\sigma \in \mathbb{N}$ of the random oracle; (b) the message size $\ell \in \mathbb{N}$; (c) the salt size $s \in \mathbb{N}$; and (d) the number of queries t to the random oracle made by the algorithm trying to distinguish between the two distributions. In the lemma below we also consider the case where $t = \infty$, which corresponds to adversaries that can make an unbounded number of queries to the random oracle (in order to choose the message and also in order to distinguish between the two distributions).

Lemma 17.3.1 (CM is hiding). *Let $\text{CM} := \text{CM}[\sigma, \ell, s]$. There exists a polynomial-time probabilistic algorithm $\text{CM}.\text{Simulate}$ such that for every query bound $t \in \mathbb{N}$ and t -query algorithm A , the following two distributions are $z_{\text{CM}}(\sigma, \ell, s, t)$ -close in statistical distance*

$$\left\{ A^f(\text{aux}, \text{cm}) \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{m}, \text{aux}) \leftarrow A^f \\ (\text{cm}, \tau) \leftarrow \text{CM}.\text{Commit}^f(\mathbf{m}) \end{array} \right\} \text{ and } \left\{ A^f(\text{aux}, \text{cm}) \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{m}, \text{aux}) \leftarrow A^f \\ \text{cm} \leftarrow \text{CM}.\text{Simulate}^f \end{array} \right\}.$$

Above,

$$z_{\text{CM}}(\sigma, \ell, s, t) \leq \begin{cases} \frac{t}{2^s} & \text{if } t < \infty \\ 3\sqrt{\sigma\ell s} \cdot 2^{-\frac{s-\sigma}{2}} & \text{if } t = \infty \end{cases}.$$

Moreover, $\text{CM}.\text{Simulate}^f$ outputs a random string in $\{0, 1\}^\sigma$.

Of course, the bound for $t = \infty$ is also a bound for $t < \infty$. This is useful when t is large compared to s , in which case the expression $t \cdot 2^{-s}$ is not small anymore.

Below we prove the upper bounds for the two cases, providing intuition before each analysis.

The case of $t < \infty$ There is a straightforward t -query distinguishing strategy:

- In the first phase, A^f outputs $\mathbf{m} := 0^\ell$ (some fixed arbitrary message) and $\text{aux} := \perp$.
- In the second phase, $A^f(\text{aux}, \text{cm})$ repeatedly samples a random salt $\tau \in \{0, 1\}^s$ and checks if $\text{cm} = f(\mathbf{m}, \tau)$; if so then it outputs 1; else if all t attempts fail then it outputs a random bit b .

In one case, if $\text{cm} = f(\mathbf{m}, \tau)$ for a random salt τ then the probability that the above strategy guesses this salt is $\Omega(t \cdot 2^{-s})$. (One can obtain an explicit constant by applying the inclusion-exclusion principle.) In the other case, if cm is a random string in $\{0, 1\}^\sigma$ (independent of f) then, except with probability $2^{s-\sigma}$, no salt τ satisfies $\text{cm} = f(\mathbf{m}, \tau)$. Overall, the distinguishing strategy leads to a statistical distance between the two cases that is $\Omega(t \cdot 2^{-s} - 2^{s-\sigma})$.

Below we prove that this strategy is essentially optimal when σ is sufficiently larger than s (e.g., $\sigma \geq 2s$), in which case the lower bound becomes $\Omega(t \cdot 2^{-s})$. This is because *every* t -query algorithm A achieves a statistical distance that is at most $t \cdot 2^{-s}$.

Proof for $t < \infty$. The adversary A performs queries in two phases: the first phase leads to A choosing the message m ; and the second phase is after A receives the commitment cm . Let tr_1 be the query-answer trace of the first phase and tr_2 the query-answer trace of the second phase. Define $t_1 := |tr_1|$ and $t_2 := |tr_2|$. Note that $t_1 + t_2 \leq t$. The final output of A is a deterministic function of the query-answer pairs in the traces tr_1 and tr_2 and the commitment cm . In one experiment cm is computed as $f(m, \tau)$ and in the other experiment it is computed as $CM.\text{Simulate}^f$. By Claim 1.2.8, the statistical distance between the final output of A in the two experiments is at most the statistical distance between the following two random variables:

$$(tr_1, f(m, \tau), tr_2) \text{ and } (tr_1, CM.\text{Simulate}^f, tr_2).$$

The simulator $CM.\text{Simulate}$ outputs a random string in $\{0, 1\}^\sigma$.

Let E be the event that (m, τ) is one of the queries in tr_1 or tr_2 . Conditioned on \overline{E} , $f(m, \tau)$ is random $\{0, 1\}^\sigma$, in which case it is distributed as $CM.\text{Simulate}^f$. Therefore the statistical distance between the two random variables above is at most $\Pr[E]$, which we upper bound next.

Claim 17.3.2. *It holds that*

$$\Pr[E] = \Pr \left[A^f(\text{aux}, cm) \text{ queries } (m, \tau) \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (m, \text{aux}) \leftarrow A^f \\ \tau \leftarrow \{0, 1\}^s \\ cm \leftarrow f(cm, \tau) \end{array} \right] \leq \frac{t}{2^s}.$$

Proof. First, we consider the trace tr_1 . The probability that (m, τ) is a query in tr_1 is at most $\frac{|tr_1|}{2^s} \leq \frac{t_1}{2^s}$, because cm is the answer to the query (m, τ) where m is the message chosen by the adversary and τ is a random salt $\tau \in \{0, 1\}^s$ sampled independently of tr_1 . Next, we consider the queries in tr_2 , performed after the adversary has chosen the message m . Conditioned on the query (m, τ) not appearing in tr_1 , the probability that any of the queries in tr_2 is (m, τ) equals the same probability but when sampling τ *after* the adversary performs the queries in tr_2 . Formally, we apply Lemma 3.2.2 with respect to $f'(\tau) := f(m, \tau)$, and deduce that the probability that (m, τ) is a query in tr_2 is at most $\frac{|tr_2|}{2^s} \leq \frac{t_2}{2^s}$. Together, we conclude that

$$\Pr[E] \leq \frac{t_1}{2^s} + \frac{t_2}{2^s} = \frac{t}{2^s}.$$

□

□

The case of $t = \infty$ If the distinguisher may query the random oracle an unbounded number of times then it can use strategies that are quite different from the bounded case. In particular, the distinguisher may query the random oracle f at every input, and use the information to attempt to learn m . For example, for some f , it is the case that $\Pr_\tau [m = 1 \mid f(m\|\tau) = cm] \gg \Pr_\tau [m = 0 \mid f(m\|\tau) = cm]$ for $cm = f(m\|\tau)$, in which case A can determine m from cm with good accuracy. Nevertheless, we expect that most random oracles are “well-behaved”: for every

image $\text{cm} \in \{0, 1\}^\sigma$ and any two messages \mathbf{m}, \mathbf{m}' the number of pre-images of the form (\mathbf{m}, τ) is roughly the same as the number of pre-images of the form (\mathbf{m}', τ) . In such a case, it remains (statistically) hard to distinguish if the message is \mathbf{m} or \mathbf{m}' (or even cm is simply random).

Informally, the error is a sum of two terms: $2^{-2\sigma\ell s} + 2\sqrt{\sigma\ell s} \cdot 2^{-\frac{s-\sigma}{2}} = O(\sqrt{\sigma\ell s} \cdot 2^{-\frac{s-\sigma}{2}})$. The first term comes from the probability that a random oracle is not well-behaved. The second term (which is the dominant one) is the statistical distance of a random variable of the form $f(\mathbf{m} \parallel \tau)$ from a random string in $\{0, 1\}^\sigma$, for a fixed well-behaved f and over a random choice of τ . In Claim 17.3.3 further below we prove that the bound is essentially tight.

Proof for $t = \infty$. The adversary is unbounded, and so its final output is a deterministic function of the random oracle f (in its entirety) and the commitment cm . In one experiment cm is computed as $f(\mathbf{m}, \tau)$ and in the other experiment it is computed as CM.Simulate^f . By Claim 1.2.8, the statistical distance between the adversary's final output in the two experiments is at most the statistical distance between the following two random variables:

$$(f, f(\mathbf{m}, \tau)) \text{ and } (f, \text{CM.Simulate}^f).$$

The simulator CM.Simulate outputs a random string in $\{0, 1\}^\sigma$.

For every $\mathbf{m} \in \{0, 1\}^\ell$, $\text{cm} \in \{0, 1\}^\sigma$, and $\tau \in \{0, 1\}^s$, define $Z_{\mathbf{m}, \text{cm}, \tau}$ to be the indicator random variable (over the choice of f) that denotes the event that $f(\mathbf{m}, \tau) = \text{cm}$. Note that $(Z_{\mathbf{m}, \text{cm}, \tau})_{\tau \in \{0, 1\}^s}$ are independent random variables, and that $\Pr[Z_{\mathbf{m}, \text{cm}, \tau} = 1] = 2^{-\sigma}$ for every $\tau \in \{0, 1\}^s$.

For every $\mathbf{m} \in \{0, 1\}^\ell$ and $\text{cm} \in \{0, 1\}^\sigma$, define $Z_{\mathbf{m}, \text{cm}}$ to be the random variable (over the choice of f) that counts the number of salts that map the message \mathbf{m} to the commitment cm :

$$Z_{\mathbf{m}, \text{cm}} := \sum_{\tau \in \{0, 1\}^s} Z_{\mathbf{m}, \text{cm}, \tau}.$$

Note that $\mathbb{E}[Z_{\mathbf{m}, \text{cm}}] = \sum_{\tau \in \{0, 1\}^s} \mathbb{E}[Z_{\mathbf{m}, \text{cm}, \tau}] = 2^{s-\sigma}$. By a Chernoff bound (Lemma 1.2.4),

$$\Pr_f \left[|Z_{\mathbf{m}, \text{cm}} - \mathbb{E}[Z_{\mathbf{m}, \text{cm}}]| \geq \delta \cdot \mathbb{E}[Z_{\mathbf{m}, \text{cm}}] \right] \leq 2e^{-\frac{\mathbb{E}[Z_{\mathbf{m}, \text{cm}}]\delta^2}{3}}.$$

Setting $\delta := 4\sqrt{\frac{\sigma\ell s}{\mathbb{E}[Z_{\mathbf{m}, \text{cm}}]}} = 4\sqrt{\sigma\ell s} \cdot 2^{-\frac{s-\sigma}{2}}$, we get

$$\Pr_f \left[|Z_{\mathbf{m}, \text{cm}} - 2^{s-\sigma}| \geq 4\sqrt{\sigma\ell s} \cdot 2^{\frac{s-\sigma}{2}} \right] \leq 2e^{-\frac{16\sigma\ell s}{3}} \leq e^{-4\sigma\ell s}.$$

Taking a union bound over all $\text{cm} \in \{0, 1\}^\sigma$, and dividing by 2^s we deduce that

$$\Pr_f \left[\exists \text{cm} \in \{0, 1\}^\sigma : \left| \frac{Z_{\mathbf{m}, \text{cm}}}{2^s} - \frac{1}{2^\sigma} \right| \geq 4\sqrt{\sigma\ell s} \cdot 2^{-\frac{s+\sigma}{2}} \right] \leq 2^\sigma \cdot e^{-4\sigma\ell s}. \quad (17.1)$$

Let U_σ be the uniform distribution over $\{0, 1\}^\sigma$; also, for a given f , let $D_{\mathbf{m}}(f)$ be the distribution $f(\mathbf{m}, \tau)$ for a uniform random salt $\tau \in \{0, 1\}^s$. For every f such that the event in Equation 17.1 does not hold (namely, for every $\text{cm} \in \{0, 1\}^\sigma$ it holds that $\left| \frac{Z_{\mathbf{m}, \text{cm}}}{2^s} - \frac{1}{2^\sigma} \right| < 4\sqrt{\sigma\ell s} \cdot 2^{-\frac{s+\sigma}{2}}$), we have that

$$\Delta(D_{\mathbf{m}}(f), U_\sigma)$$

$$\begin{aligned}
&= \frac{1}{2} \cdot \sum_{\mathbf{cm} \in \{0,1\}^\sigma} \left| \frac{Z_{\mathbf{m}, \mathbf{cm}}(f)}{2^s} - \frac{1}{2^\sigma} \right| \\
&< \frac{1}{2} \cdot \sum_{\mathbf{cm} \in \{0,1\}^\sigma} 4\sqrt{\sigma \ell s} \cdot 2^{-\frac{s+\sigma}{2}} \\
&= 2 \cdot 2^\sigma \cdot \sqrt{\sigma \ell s} \cdot 2^{-\frac{s+\sigma}{2}} \\
&= 2\sqrt{\sigma \ell s} \cdot 2^{-\frac{s-\sigma}{2}}.
\end{aligned}$$

Taking a union bound over all $\mathbf{m} \in \{0,1\}^\ell$, we deduce that

$$\Pr_f \left[\exists \mathbf{m} \in \{0,1\}^\ell : \Delta(D_{\mathbf{m}}(f), U_\sigma) \geq 2\sqrt{\sigma \ell s} \cdot 2^{-\frac{s-\sigma}{2}} \right] \leq 2^\ell \cdot 2^\sigma \cdot e^{-4\sigma \ell s} \leq 2^{-2\sigma \ell s}.$$

Using Claim 1.2.9 we conclude that the statistical distance is

$$\Delta((f, D_{\mathbf{m}}(f)), (f, U_\sigma)) \leq 2^{-2\sigma \ell s} + 2\sqrt{\sigma \ell s} \cdot 2^{-\frac{s-\sigma}{2}} \leq 3\sqrt{\sigma \ell s} \cdot 2^{-\frac{s-\sigma}{2}}.$$

□

Claim 17.3.3. *Let $\text{CM} := \text{CM}[\sigma, \ell, s]$. For every message $\mathbf{m} \in \{0,1\}^\ell$ there exists an unbounded adversary A for which the following two distributions are $\Omega(2^{-\frac{s-\sigma}{2}})$ -far in statistical distance:*

$$\left\{ A^f(\mathbf{cm}) \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{cm}, \tau) \leftarrow \text{CM.Commit}^f(\mathbf{m}) \end{array} \right\} \text{ and } \left\{ A^f(\mathbf{cm}) \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \mathbf{cm} \leftarrow \text{CM.Simulate}^f \end{array} \right\}.$$

Proof. The adversary A performs the following 2^s -query attack: compute $\Pr_\tau[f(\mathbf{m}, \tau) = \mathbf{cm}]$ and output 1 if and only if this probability is at least $\alpha := 2^{-\sigma} + \frac{1}{4} \cdot 2^{-\frac{s-\sigma-1}{2}}$.

Fix a random oracle f . For every $y \in \{0,1\}^\sigma$, define $S_{f,y} := \{\tau \in \{0,1\}^s : f(\mathbf{m}, \tau) = y\}$. Define $X_{f,y}$ to be 1 if $\Pr_\tau[\tau \in S_{f,y}] \geq \alpha$, and 0 otherwise. Define $X_f := \sum_{y \in \{0,1\}^\sigma} X_{f,y}$.

For every $f \in \mathcal{U}(\sigma)$, the probability that A outputs 1 equals the probability that the random salt τ is such that $X_{f,f(\mathbf{m}, \tau)} = 1$. Hence

$$\begin{aligned}
&\Pr \left[A^f(\mathbf{cm}) = 1 \mid (\mathbf{cm}, \tau) \leftarrow \text{CM.Commit}^f(\mathbf{m}) \right] \\
&= \Pr \left[A^f(\mathbf{cm}) = 1 \mid \begin{array}{l} \tau \leftarrow \{0,1\}^s \\ \mathbf{cm} \leftarrow f(\mathbf{m}, \tau) \end{array} \right] \\
&= \sum_{y \in \{0,1\}^\sigma} \Pr_\tau[\tau \in S_{f,y}] \cdot X_{f,y} \\
&\geq \sum_{y \in \{0,1\}^\sigma} \alpha \cdot X_{f,y} = \alpha \cdot X_f.
\end{aligned}$$

Averaging over all f we get that

$$\begin{aligned}
&\Pr \left[A^f(\mathbf{cm}) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{cm}, \tau) \leftarrow \text{CM.Commit}^f(\mathbf{m}) \end{array} \right] \\
&= \Pr \left[A^f(\mathbf{cm}) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \tau \leftarrow \{0,1\}^s \\ \mathbf{cm} \leftarrow f(\mathbf{m}, \tau) \end{array} \right] \geq \alpha \cdot \mathbb{E}_f[X_f].
\end{aligned}$$

On the other hand,

$$\begin{aligned} & \Pr \left[A^f(\mathbf{cm}) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \mathbf{cm} \leftarrow \text{CM.Simulate}^f \end{array} \right] \\ &= \Pr \left[A^f(\mathbf{cm}) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ \mathbf{cm} \leftarrow \{0, 1\}^\sigma \end{array} \right] = 2^{-\sigma} \cdot \mathbb{E}_f[X_f]. \end{aligned}$$

Thus, the statistical distance δ is at least

$$\delta \geq \alpha \cdot \mathbb{E}_f[X_f] - 2^{-\sigma} \cdot \mathbb{E}_f[X_f] = \mathbb{E}_f[X_f] \cdot (\alpha - 2^{-\sigma}) = \mathbb{E}_f[X_f] \cdot \frac{1}{4} \cdot 2^{\frac{-s-\sigma-1}{2}}.$$

We lower bound $\mathbb{E}_f[X_f]$ in the following claim.

Claim 17.3.4. $\mathbb{E}_f[X_f] = \Omega(2^\sigma)$

Proof. We lower bound $\mathbb{E}_f[X_{f,y}]$ for every $y \in \{0, 1\}^\sigma$. The expectation $\mathbb{E}_f[X_{f,y}]$ equals the probability that a binomial distribution with 2^s experiments and success probability $2^{-\sigma}$ is at least $2^s \cdot \alpha$. Denoting the binomial probability with Bin , we have that $\mathbb{E}_f[X_{f,y}] = \Pr[\text{Bin}(2^s, 2^{-\sigma}) \geq 2^s \cdot \alpha]$. The binomial distribution can be approximated (up to constant factor) by a normal distribution (denoted by \mathcal{N}) as follows: for any x , $\Pr[\text{Bin}(n, p) \geq x] = \Omega(\Pr[\mathcal{N}(np, np(1-p)) \geq x])$. Thus, there is a constant $c > 0$ such that

$$\begin{aligned} \mathbb{E}_f[X_{f,y}] &= \Pr[\text{Bin}(2^s, 2^{-\sigma}) \geq 2^s \cdot \alpha] \\ &\geq c \cdot \Pr[\mathcal{N}(2^{s-\sigma}, 2^{s-\sigma}(1-2^{-\sigma})) \geq 2^s \cdot \alpha] \\ &\geq c \cdot \Pr[\mathcal{N}(2^{s-\sigma}, 2^{s-\sigma-1}) \geq 2^s \cdot \alpha] \\ &= c \cdot \Pr[\mathcal{N}(0, 2^{s-\sigma-1}) \geq 2^s \cdot \alpha - 2^{s-\sigma}] \\ &= c \cdot \Pr \left[\mathcal{N}(0, 1) \geq \frac{\alpha - 2^{-\sigma}}{2^{\frac{-s-\sigma-1}{2}}} \right] \quad (\text{recall that } b \cdot \mathcal{N}(0, \sigma^2) = \mathcal{N}(0, b^2 \cdot \sigma^2).) \\ &= c \cdot \Pr \left[\mathcal{N}(0, 1) \geq \frac{1}{4} \right] = \Omega(1). \end{aligned}$$

Then, by linearity of expectation, we get that $\mathbb{E}_f[X_f] \geq \Omega(2^\sigma)$. □

Plugging this in the lower bound for δ we get that

$$\delta \geq \mathbb{E}_f[X_f] \cdot \frac{1}{4} \cdot 2^{\frac{-s-\sigma-1}{2}} = \Omega \left(2^\sigma \cdot \frac{1}{4} \cdot 2^{\frac{-s-\sigma-1}{2}} \right) = \Omega \left(2^{-\frac{s-\sigma}{2}} \right).$$

□

18 Merkle commitment scheme

We describe the *Merkle commitment scheme*, a commitment scheme in the ROM that produces succinct commitments to long lists of values and enables to cheaply open particular subsets of values in the list. This useful “succinct opening” property is used to construct succinct arguments in the ROM.¹ The commitment scheme is so called because it relies on Merkle trees [Mer89a], a powerful and versatile data structure based on hash functions. A Merkle commitment scheme uses the basic commitment scheme CM as a building block, so throughout this chapter we assume familiarity with the material in Chapter 17.

Organization In Section 18.1 we describe the construction of a Merkle commitment scheme, and discuss its efficiency properties. Then in Section 18.2 we discuss its *completeness property*, that is, how an honest use of the commitment scheme always leads to successful validity checks. After that we discuss security properties. In Section 18.3 we establish a technical property that we use in several analyses. Finally, we discuss the main security properties of a Merkle commitment scheme:

- in Section 18.4 we prove that MT is *binding*;
- in Section 18.5 we prove that MT is *extractable*;
- in Section 18.6 we prove that MT is *hiding*.

18.1 Definition

A *Merkle commitment scheme* is a tuple of three algorithms:

$$\text{MT} = (\text{MT.Commit}, \text{MT.Open}, \text{MT.Check}) .$$

The scheme has several parameters: an output size $\sigma \in \mathbb{N}$ of the random oracle, a message alphabet Σ , a message length ℓ , and a salt size $s \in \mathbb{N}$. We use the notation $\text{MT}[\sigma, \Sigma, \ell, s]$ when we wish to make these parameters explicit.

The algorithms receive query access to a random oracle $f \in \mathcal{U}(\sigma)$ and have the following syntax.

- *Commit*. The algorithm MT.Commit receives as input a message vector $\mathbf{m} \in \Sigma^\ell$, and computes a Merkle commitment $\text{rt} \in \{0, 1\}^\sigma$ and corresponding opening trapdoor $\text{td} \in \{0, 1\}^{\mathcal{O}((s+\sigma)\cdot\ell)}$. This algorithm is probabilistic if privacy is desired (otherwise it is deterministic).
- *Open*. The algorithm MT.Open receives as input opening trapdoor td and a subset $I \subseteq [\ell]$, and computes an opening proof pf that authenticates the values at the locations in I .

¹More generally, succinct commitments with succinct opening are known as *vector commitments* in the cryptography literature, and constructions are known from various cryptographic assumptions. They are used to construct succinct arguments in various settings also beyond the (pure) ROM.

- *Check.* The algorithm $\text{MT}.\text{Check}$ receives as input a Merkle commitment rt , subset $I \subseteq [\ell]$, claimed values $\mathbf{a} \in \Sigma^I$, and opening proof pf , and computes a bit indicating whether the opening proof pf authenticates \mathbf{a} as values for the locations in I with respect to rt .

Henceforth we assume that ℓ is a power of 2; see Remark 18.1.7 for a discussion of how to remove this assumption.

Binary tree We introduce notation for a (perfect) binary tree on ℓ leaves,² which then we use to describe the algorithms of the Merkle commitment scheme. Those algorithms can be viewed as labeling (or checking the labels of) vertices of this (perfect) binary tree.

Definition 18.1.1. For $\ell \in \mathbb{N}$, $T_\ell = (V_\ell, E_\ell)$ is the (perfect) **binary tree graph** of depth

$$d := \log_2 \ell \tag{18.1}$$

where the vertex set V_ℓ and edge set E_ℓ are defined as follows:

$$\begin{aligned} V_\ell &:= \{(j, i) : j \in \{0, 1, \dots, d\}, i \in [2^j]\}, \\ E_\ell &:= \{((j-1, i), (j, 2i-b)) : j \in \{1, \dots, d\}, i \in [2^{j-1}], b \in \{1, 0\}\}. \end{aligned}$$

We use the depth d throughout this chapter without explicitly re-defining it as in Equation 18.1.

Definition 18.1.2. We make the following notational definitions:

- The **root vertex** of T_ℓ is the vertex $(0, 1)$.
- The **leaf vertices** of T_ℓ are the vertices $\{(d, i)\}_{i \in [\ell]}$.
- A vertex (j, i) is **odd** if i is odd and it is **even** if i is even.
- The **sibling** of a non-root vertex (j, i) (i.e., with $j > 0$) is $(j, i+1)$ if (j, i) is odd and is $(j, i-1)$ if (j, i) is even.
- The **path** from the i -th leaf vertex (d, i) to the root vertex is denoted by $\text{path}(i)$; the vertex in $\text{path}(i)$ that is in layer j is denoted $\mathbf{p}(i, j) \in \{j\} \times [2^j]$.
- The **copath** from the i -th leaf vertex (d, i) to the root vertex is denoted by $\text{copath}(i)$, and is the list of siblings of each vertex in $\text{path}(i)$, except the root vertex (which has no siblings); the vertex in $\text{copath}(i)$ that is in layer j is denoted $\bar{\mathbf{p}}(i, j) \in \{j\} \times [2^j]$ (and is the sibling of $\mathbf{p}(i, j)$).
- For $I \subseteq [\ell]$, $\text{path}(I) := \cup_{i \in I} \text{path}(i)$ and $\text{copath}(I) := \cup_{i \in I} \text{copath}(i)$ (viewing lists as sets).

Using this notation we can write:

$$\text{path}(i) = \left(\mathbf{p}(i, j) \right)_{j \in \{0, 1, \dots, d\}} \text{ and } \text{copath}(i) = \left(\bar{\mathbf{p}}(i, j) \right)_{j \in \{1, \dots, d\}}.$$

For every $i \in [\ell]$, $\mathbf{p}(i, 0) = (0, 1)$ is the root vertex and $\mathbf{p}(i, d) = (d, i)$ is the i -th leaf.

Construction We describe each of the algorithms in the Merkle commitment scheme **MT**.

- $\text{MT}.\text{Commit}^f(\mathbf{m}) \rightarrow (\text{rt}, \text{td})$
- 1. For each message location $i = 1, \dots, \ell$:

²A binary tree is *perfect* if all internal vertices have two children and all leaf vertices belong to the same layer.

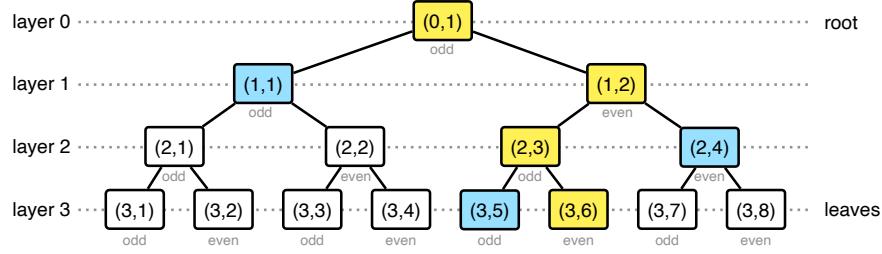


Figure 18.1: Diagram of the binary tree graph T_8 . The vertices highlighted in yellow are $\text{path}(6) = ((3,6), (2,3), (1,2), (0,1))$ and the vertices highlighted in blue are $\text{copath}(6) = ((3,5), (2,4), (1,1))$. For example, $p(6,1) = (1,2)$ and $\bar{p}(6,1) = (1,1)$.

- sample a salt $\tau_i \in \{0, 1\}^s$;
 - compute the (hiding) commitment $c_{(d,i)} := f(\mathbf{m}[i], \tau_i) \in \{0, 1\}^\sigma$.
2. For each layer $j = d - 1, \dots, 0$ and for each $i = 1, \dots, 2^j$:
 - compute the (non-hiding) commitment $c_{(j,i)} := f(c_{(j+1,2i-1)}, c_{(j+1,2i)}) \in \{0, 1\}^\sigma$.
 3. Set the Merkle commitment $\mathbf{rt} := c_{(0,1)}$.
 4. Set the salts $\boldsymbol{\tau} := (\tau_i)_{i \in [\ell]}$.
 5. Set the commitments $C := ((c_{(j,i)})_{i \in [2^j]})_{j=0}^d$, which we can visualize in the following way:

$$C = \begin{bmatrix} c_{(0,1)} \\ c_{(1,1)} & c_{(1,2)} \\ c_{(2,1)} & c_{(2,2)} & c_{(2,3)} & c_{(2,4)} \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ c_{(d,1)} & \cdots & \cdots & \cdots & \cdots & c_{(d,2^d)} \end{bmatrix}.$$

6. Set the opening trapdoor $\mathbf{td} := (\boldsymbol{\tau}, C)$.
7. Output $(\mathbf{rt}, \mathbf{td})$.

Informally, this algorithm first commits to each message entry via a basic commitment scheme (the one discussed in Chapter 17) and labels the leaves of the binary tree with these commitments, and then recursively labels every vertex in the tree with the hash of the labels of its two children.

- $\text{MT.Open}^f(\mathbf{td}, I) \rightarrow \mathbf{pf}$

1. For every $i \in I$, set the authentication path for location i :

$$\mathbf{auth}_i := (\tau_i, (c_{\bar{p}(i,j)})_{j \in \{1, \dots, d\}}). \quad (18.2)$$

2. Output the opening proof $\mathbf{pf} := (\mathbf{auth}_i)_{i \in I}$.

Informally, this algorithm includes an authentication path for each opened location. An authentication for a location includes the salt for that location, as well as the values assigned to siblings of vertices from that leaf location to the root.

- $\text{MT.Check}^f(\mathbf{rt}, I, \mathbf{a}, \mathbf{pf}) \rightarrow b$

1. Parse \mathbf{pf} as $(\mathbf{auth}_i)_{i \in I}$.

2. For every $i \in I$, check that auth_i authenticates the value $\mathbf{a}[i] \in \Sigma$ as the i -th opening relative to the Merkle commitment $\mathbf{rt} \in \{0, 1\}^\sigma$ by computing $(c_{p(i,j)})_{j \in \{0, 1, \dots, d\}}$:
 - a) compute the commitment $c_{(d,i)} := f(\mathbf{a}[i], \tau_i)$;
 - b) compute the commitments $(c_{p(i,j)})_{j \in \{0, 1, \dots, d-1\}}$ as follows:
for each layer $j = d-1, \dots, 1, 0$:
 - if $p(i, j+1)$ is an odd vertex then set $c_L := c_{p(i,j+1)}$ and $c_R := c_{\bar{p}(i,j+1)}$;
 - if $p(i, j+1)$ is an even vertex then set $c_L := c_{\bar{p}(i,j+1)}$ and $c_R := c_{p(i,j+1)}$;
 - compute the commitment $c_{p(i,j)} := f(c_L, c_R)$.
 - c) check that $\mathbf{rt} = c_{(0,1)}$.

Informally, this algorithm checks the authentication path from each location. Checking an authentication path involves re-computing the values assigned to vertices from the leaf vertex of the location to the root vertex, and checking that the resulting root value equals the given Merkle commitment. Note that

$$\text{MT.Check}^f(\mathbf{rt}, I, \mathbf{a}, \mathbf{pf}) = \wedge_{i \in I} \text{MT.Check}^f(\mathbf{rt}, i, \mathbf{a}[i], \text{auth}_i). \quad (18.3)$$

Above we slightly abuse notation in that we drop the set notation when MT.Check receives a single location and a single corresponding authentication path.

Efficiency We discuss the efficiency of each algorithm.

- The algorithm MT.Commit makes ℓ queries of size $\log |\Sigma| + s$, and $\sum_{j \in [d]} \ell / 2^{d-j-1} \leq \ell$ queries of size 2σ . The output Merkle commitment \mathbf{rt} has size σ and opening trapdoor \mathbf{td} has size $s\ell + 2\ell\sigma$.
- The algorithm MT.Open makes no oracle calls, and simply includes in the opening proof an authentication path for each index in the set I . An authentication path has size $s + d\sigma$, since it includes the salt and a hash per layer. Hence an opening proof consists of $|I| \cdot (s + d\sigma)$ bits. The path-pruning optimization discussed in Section 29.2 reduces the size of an opening proof.
- The algorithm MT.Check , for each location in the set I , makes one query of size $\log |\Sigma| + s$ and d queries of size 2σ . The total number of queries is $|\cup_{i \in I} \text{path}(i)|$ (there may be duplicates).

Remark 18.1.3 (no privacy). If no privacy is desired then the construction of MT simplifies as follows: (i) $s = 0$ (no salts are sampled); (ii) in MT.Commit , set $c_{(d,i)} := \mathbf{m}[i]$ for each $i \in [\ell]$ (instead of committing first by setting $c_{(d,i)} := f(\mathbf{m}[i], \tau_i)$) and then the opening trapdoor \mathbf{td} does not include any salts; (iii) in MT.Open , each authentication path auth_i does not include any salt; (iv) in MT.Check , similarly to MT.Commit , simply set $c_{(d,i)} := \mathbf{m}[i]$ for each $i \in I$ (in Item 2a). The efficiency measures are correspondingly reduced by setting $s = 0$.

Remark 18.1.4 (recomputing the tree). Some information from \mathbf{td} can be omitted (thereby reducing its size) at the expense of additional computation by MT.Open . Specifically, the commitments $((c_{(j,i)})_{i \in [2^j]})_{j=1}^{d-1}$ in the opening trapdoor \mathbf{td} can be derived via at most ℓ random oracle calls from the commitments $(c_{(d,i)})_{i \in [\ell]}$, and thus could be omitted from \mathbf{td} . In such a case, MT.Open would have to query the random oracle to (re)derive the commitments required to construct the relevant authentication paths. In fact, $(c_{(d,i)})_{i \in [\ell]}$ can themselves be derived

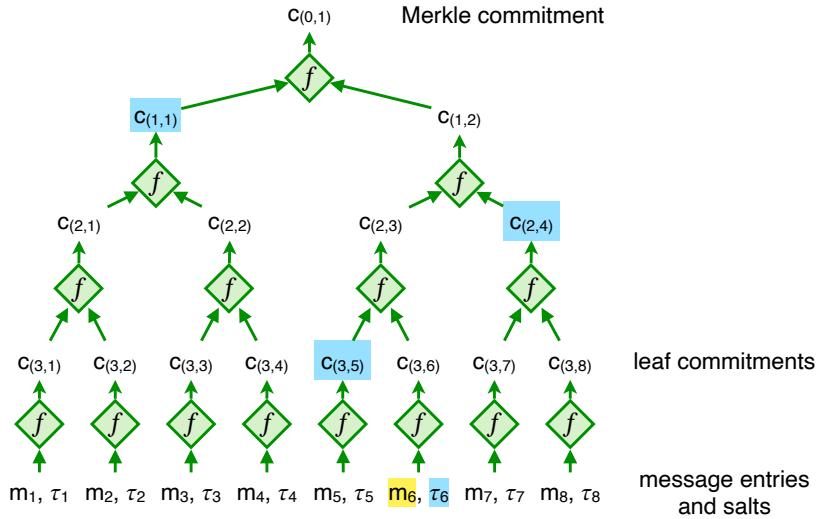


Figure 18.2: Diagram of the computation in $\text{MT}.\text{Commit}$ to produce a Merkle commitment for a message of length $\ell = 8$. Moreover, highlighted in blue is the information included in an opening proof pf for the 6-th entry of the message (which is highlighted in yellow): the salt τ_6 and the commitments $(c_{(3,5)}, c_{(2,4)}, c_{(1,1)})$ that correspond to the copath $\text{copath}(6) = ((3, 5), (2, 4), (1, 1))$.

from the message \mathbf{m} and salts $\boldsymbol{\tau}$ via ℓ random oracle calls; so one could even set $\text{td} := (\mathbf{m}, \boldsymbol{\tau})$, which may be advantageous if $\log |\Sigma|$ is smaller than σ . These can be useful time-memory tradeoffs in practice.

Remark 18.1.5 (local updates). Merkle commitments are *locally updatable*, which is a useful feature in many applications (though we do not explore it further within the context of succinct arguments). Suppose you are given a Merkle commitment rt of a message \mathbf{m} , and you want to update it to be a Merkle commitment rt' of a message \mathbf{m}' where \mathbf{m}' is equal to \mathbf{m} except for the i -th entry. There is no need to recompute the tree from scratch: one only needs to recompute the commitments for the vertices on the paths $\text{path}(i)$ and $\text{copath}(i)$, while the other commitments remain unchanged. The work needed for this update is proportional to the depth of the tree (which is logarithmic in the message length) rather than the message length.

Remark 18.1.6 ($\ell = 1$). The message length $\ell = 1$ is a power of 2, and in this case the binary tree T_ℓ contains a single vertex, the root vertex. In this degenerate case ($\ell = 1$) the Merkle commitment MT happens to equal the basic commitment CM .

Remark 18.1.7 (any message length). The description of the Merkle commitment scheme MT given above involves a binary tree over ℓ leaves, where ℓ is the number of entries in the message vector. Notationally it is far simpler to describe and analyze the case when ℓ is a power of 2 (which we assume throughout this chapter) because the internal vertices in the binary tree can be labeled via a simple naming scheme and, more generally, all paths from the root vertex to a leaf vertex “look the same”. But how do Merkle commitment schemes work when ℓ is not a power of 2?

One option is to pad the message with dummy values (an arbitrary symbol from the alphabet Σ) until the padded message length reaches a power of 2; this, at most, doubles the length

relative to the original (unpadded) message. All security analyses of this chapter hold for the padded message, and in particular for the original (unpadded) message.

While the padding is not expensive, there are cheaper solutions that involve binary trees with precisely ℓ leaves (without using any padding). This optimization is explained in Section 29.1.

Remark 18.1.8 (other arities). The description above involves a perfect *binary* tree. However the construction of a Merkle commitment scheme and the properties that we study (completeness, binding, extractability, hiding) directly extend to perfect trees of any arity. (In fact, see Section 29.1 for a *generalized* Merkle commitment scheme that is defined for *any tree*.) If the tree has arity $a \geq 2$ then each authentication path contains $a - 1$ vertices per layer rather than 1, and the number of layers in the tree is $\log_a \ell$ rather than $\log_2 \ell$. Larger arities are usually not advantageous for the size of opening proofs (the tree has fewer layers but there are more sibling vertices for an overall bigger opening proof size), though there may be other reasons to consider trees of higher arity (e.g., fewer calls to the random oracle).

18.2 Completeness

We prove that $\text{MT}.\text{Commit}$ has *completeness*, namely, if the algorithms of the Merkle commitment scheme are used as intended then validity checks always pass. In more detail, $\text{MT}.\text{Commit}$ can be used to commit to any message vector, and $\text{MT}.\text{Open}$ can be used to open any subset of locations of the message vector in such a way that the resulting opening proof is accepted by $\text{MT}.\text{Check}$.

Lemma 18.2.1 (MT is complete). *Let $\text{MT} := \text{MT}[\sigma, \Sigma, \ell, s]$. For every adversary A ,*

$$\Pr \left[\text{MT}.\text{Check}^f(\text{rt}, I, \mathbf{m}[I], \text{pf}) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{m}, I) \leftarrow A^f \\ (\text{rt}, \text{td}) \leftarrow \text{MT}.\text{Commit}^f(\mathbf{m}) \\ \text{pf} \leftarrow \text{MT}.\text{Open}^f(\text{td}, I) \end{array} \right] = 1.$$

A formal proof of this lemma would involve analyzing each step of each Merkle commitment algorithm in order to establish that all validity checks pass. This would not provide useful insights. Instead we sketch the proof at high level to provide the main intuition.

Fix an arbitrary choice of oracle $f \in \mathcal{U}(\sigma)$, message $\mathbf{m} \in \Sigma^\ell$, and subset $I \subseteq [\ell]$.

The algorithm $\text{MT}.\text{Commit}$ computes the Merkle commitment for \mathbf{m} by first committing to each entry of \mathbf{m} (using a random salt τ_i for each message entry $\mathbf{m}[i]$), then pairwise hashes the resulting list to halve its size, and so on, until it obtains a single output rt ; the trapdoor td is set to contain all the salts that were sampled and all the commitments $C := ((c_{(j,i)})_{i \in [2^j]})_{j=0}^d$ that were computed.

Subsequently, the algorithm $\text{MT}.\text{Open}$ includes in the opening proof pf an authentication path $\text{auth}_i = (\tau_i, (c_{\bar{p}(i,j)})_{j \in \{1, \dots, d\}})$ for each index i in the subset I .

The algorithm $\text{MT}.\text{Check}$, for each index i in the subset I , checks that the authentication path auth_i is valid relative to the Merkle commitment rt , index i , and value $\mathbf{a}[i]$. The check for each authentication path succeeds because $\text{MT}.\text{Open}$ constructed that path by including the correct salt and the correct commitments, which will lead $\text{MT}.\text{Check}$ to recompute the same value for the root vertex as rt .

18.3 Collision lemma

We formulate and prove a *collision lemma* for Merkle commitments, which is a basic security property that we use in later sections. Informally, the lemma states that authenticating different values for the same leaf in a Merkle commitment leads to a collision in the random oracle. Stating this property precisely, however, requires some care. The discussion below takes an incremental approach by considering simpler types of collision lemmas. First, we describe a collision lemma for the basic commitment scheme CM in Chapter 17. Second, we describe a special case of the collision lemma for Merkle commitment schemes. Third, and finally, we describe the (general) collision lemma for Merkle commitment schemes.

Starting simple: collisions in CM Consider the basic commitment scheme CM in Chapter 17, whose checking algorithm $\text{CM.Check}^f(\text{cm}, \mathbf{m}, \tau)$ simply consists of checking a query-answer pair of the random oracle: $\text{CM.Check}^f(\text{cm}, \mathbf{m}, \tau) = 1$ if and only if $\text{cm} = f(\mathbf{m}, \tau)$. Consider a commitment cm and two message-salt pairs, (\mathbf{m}_0, τ_0) and (\mathbf{m}_1, τ_1) . Suppose that both pairs are valid openings of the commitment, that is, $\text{CM.Check}^f(\text{cm}, \mathbf{m}_0, \tau_0) = 1$ and $\text{CM.Check}^f(\text{cm}, \mathbf{m}_1, \tau_1) = 1$. By the definition of CM.Check , these conditions are equivalent to $\text{cm} = f(\mathbf{m}_0, \tau_0)$ and $\text{cm} = f(\mathbf{m}_1, \tau_1)$, from which we deduce that $f(\mathbf{m}_0, \tau_0) = f(\mathbf{m}_1, \tau_1)$. This tells us that if either $\mathbf{m}_0 \neq \mathbf{m}_1$ (the messages are distinct) or $\tau_0 \neq \tau_1$ (the salts are distinct) then the two executions of CM.Check on the two inputs result in a collision for f . Thus, in CM, *opening the same commitment either via two different messages or via two different salts leads to a collision*. We can summarize this property as a “collision lemma for CM” via the following implication:

$$\left[\begin{array}{l} (\mathbf{m}_0 \neq \mathbf{m}_1 \vee \tau_0 \neq \tau_1) \\ \wedge \text{CM.Check}^f(\text{cm}, \mathbf{m}_0, \tau_0) = 1 \\ \wedge \text{CM.Check}^f(\text{cm}, \mathbf{m}_1, \tau_1) = 1 \end{array} \right] \Rightarrow \left[\begin{array}{l} (\mathbf{m}_0, \tau_0) \text{ and } (\mathbf{m}_1, \tau_1) \\ \text{are a collision for } f \end{array} \right].$$

What about MT? We wish to derive a similar lemma for the Merkle commitment scheme MT. The challenge is that messages in MT can be opened at subsets of locations (rather than only at all locations), which complicates the assumptions under which a collision can be proved to exist. Moreover, the algorithm MT.Check , which validates the corresponding opening proofs, involves a more complex computation. In particular, the computation involves multiple queries to the random oracle (rather than one query as in CM), and arguing that a collision exists in the relevant query-answer traces will involve a delicate case analysis. We tackle this additional complexity by first analyzing a special case and then analyzing the general case.

Special case: opening a single leaf As a warmup we consider the special case where we only consider opening proofs for a single leaf. In other words, we consider computations for $\text{MT.Check}^f(\text{rt}, I, \mathbf{a}, \text{pf})$ where the set $I = \{i\}$ is a singleton and so: (i) \mathbf{a} is a vector consisting of a single entry m in Σ ; and (ii) pf consists of a single authentication path auth .

Consider a Merkle commitment rt and position $i \in [\ell]$, and two openings: a claimed value $m_0 \in \Sigma$ and corresponding authentication path auth_0 , and another claimed value $m_1 \in \Sigma$ and corresponding authentication path auth_1 . Suppose that both openings are valid, that is, $\text{MT.Check}^f(\text{rt}, i, m_0, \text{auth}_0) = 1$ and $\text{MT.Check}^f(\text{rt}, i, m_1, \text{auth}_1) = 1$. Intuitively, since the two computations share the same Merkle commitment rt , if $m_0 \neq m_1$ then the two random oracle computations “collide” at some layer of the binary tree in order to eventually lead to the same Merkle commitment rt at the root of the tree. The collision is either at a leaf commitment,

or at an internal commitment; see Figure 18.3 for an example. In fact, we can prove that the same is true even if $m_0 = m_1$ and $\text{auth}_0 \neq \text{auth}_1$.

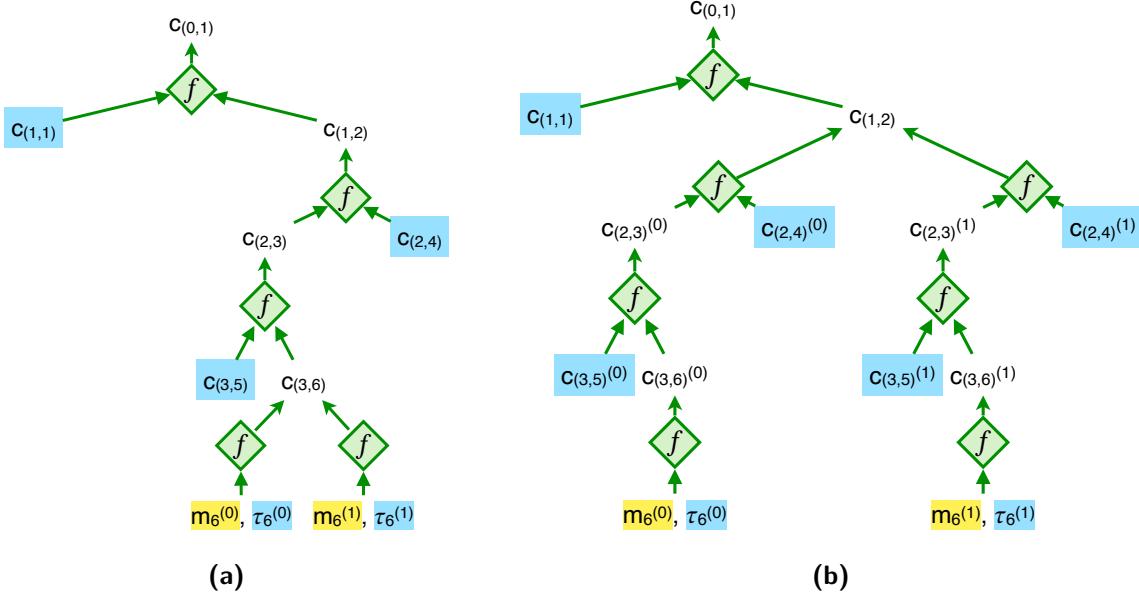


Figure 18.3: Example of a collision for two paths that authenticate two different values for the 6-th entry (out of 8 entries). There are two cases: in the left, a collision at a leaf commitment; and, on the right, a collision at an internal commitment.

More precisely, the collision involves a query from the query-answer trace S_0 of the computation $\text{MT}.\text{Check}^f(\text{rt}, i, m_0, \text{auth}_0)$ and a query from the query-answer trace S_1 of the computation $\text{MT}.\text{Check}^f(\text{rt}, i, m_1, \text{auth}_1)$.

Note that here it is necessary for the two openings to be about the same position i . This is because it is possible to open two different positions via two different values (or different authentication paths) without causing collisions. Indeed, this would happen for most choices of random oracle for a message with different values.

We formally state and prove this below.

Lemma 18.3.1. *Let $\text{MT} := \text{MT}[\sigma, \Sigma, \ell, s]$. Fix arbitrary choices of oracle $f \in \mathcal{U}(\sigma)$ and: a Merkle commitment $\text{rt} \in \{0, 1\}^\sigma$; an index $i \in [\ell]$; message entries $m_0, m_1 \in \Sigma$; and authentication paths $\text{auth}_0, \text{auth}_1$. For $b \in \{0, 1\}$, let S_b be the query-answer trace of $\text{MT}.\text{Check}^f(\text{rt}, i, m_b, \text{auth}_b)$. Then the following implication holds:*

$$\left[\begin{array}{l} (m_0 \neq m_1 \vee \text{auth}_0 \neq \text{auth}_1) \\ \wedge \text{MT}.\text{Check}^f(\text{rt}, i, m_0, \text{auth}_0) = 1 \\ \wedge \text{MT}.\text{Check}^f(\text{rt}, i, m_1, \text{auth}_1) = 1 \end{array} \right] \Rightarrow \left[\begin{array}{l} \exists (x_0, y_0) \in S_0, (x_1, y_1) \in S_1 \\ \text{with } x_0 \neq x_1 \text{ such that } y_0 = y_1 \end{array} \right]. \quad (18.4)$$

Proof. Fix $b \in \{0, 1\}$. The authentication path auth_b for index i has the following structure:

$$\text{auth}_b = \left(\tau_i^{(b)}, (c_{\text{p}(i,j)}^{(b)})_{j \in \{1, \dots, d\}} \right),$$

where $\tau_i^{(b)}$ is a Merkle salt and $(c_{\text{p}(i,j)}^{(b)})_{j \in \{1, \dots, d\}}$ are commitments labeling vertices in $\text{copath}(i)$.

Moreover, the computation $\text{MT.Check}^f(\text{rt}, i, m_b, \text{auth}_b)$ leads to a query-answer trace S_b consisting of $d + 1$ query-answer pairs. The answers in S_b are denoted

$$(c_{\mathbf{p}(i,j)}^{(b)})_{j \in \{0,1,\dots,d\}}.$$

Finally, $\text{MT.Check}^f(\text{rt}, i, m_b, \text{auth}_b) = 1$ implies that $\text{rt} = c_{(0,1)}^{(b)}$.

In particular, we deduce that $c_{(0,1)}^{(0)} = c_{(0,1)}^{(1)}$.

We consider two cases.

- Case 1: there exists $j \in [d]$ such that $c_{\mathbf{p}(i,j)}^{(0)} \neq c_{\mathbf{p}(i,j)}^{(1)}$ or $c_{\bar{\mathbf{p}}(i,j)}^{(0)} \neq c_{\bar{\mathbf{p}}(i,j)}^{(1)}$.

Let j be the minimal index satisfying this condition.

- If $\mathbf{p}(i,j)$ is an odd vertex then for $b \in \{0,1\}$ set $(c_L^{(b)}, c_R^{(b)}) := (c_{\mathbf{p}(i,j)}^{(b)}, c_{\bar{\mathbf{p}}(i,j)}^{(b)})$.
- If $\mathbf{p}(i,j)$ is an even vertex then for $b \in \{0,1\}$ set $(c_L^{(b)}, c_R^{(b)}) := (c_{\bar{\mathbf{p}}(i,j)}^{(b)}, c_{\mathbf{p}(i,j)}^{(b)})$.

For $b \in \{0,1\}$ the pair $(c_L^{(b)}, c_R^{(b)})$ is a query in S_b (see Item 2b in the description of MT.Check).

These two queries are a collision for f due to the following.

- $(c_L^{(0)}, c_R^{(0)}) \neq (c_L^{(1)}, c_R^{(1)})$.
This follows since $c_{\mathbf{p}(i,j)}^{(0)} \neq c_{\mathbf{p}(i,j)}^{(1)}$ or $c_{\bar{\mathbf{p}}(i,j)}^{(0)} \neq c_{\bar{\mathbf{p}}(i,j)}^{(1)}$.
- $f(c_L^{(0)}, c_R^{(0)}) = f(c_L^{(1)}, c_R^{(1)})$.
This follows since $c_{\mathbf{p}(i,j-1)}^{(0)} = f(c_L^{(0)}, c_R^{(0)})$, $c_{\mathbf{p}(i,j-1)}^{(1)} = f(c_L^{(1)}, c_R^{(1)})$, and $c_{\mathbf{p}(i,j-1)}^{(0)} = c_{\mathbf{p}(i,j-1)}^{(1)}$. The last equality holds because j is the minimal index satisfying the condition (and we know that $c_{(0,1)}^{(0)} = c_{(0,1)}^{(1)}$).

- Case 2: $c_{(d,i)}^{(0)} = c_{(d,i)}^{(1)}$, and $m_0 \neq m_1$ or $\tau_i^{(0)} \neq \tau_i^{(1)}$.

For $b \in \{0,1\}$ the pair $(m_b, \tau_i^{(b)})$ is a query in S_b (see Item 2a in the description of MT.Check).

These two queries are a collision for f due to the following.

- $(m_0, \tau_i^{(0)}) \neq (m_1, \tau_i^{(1)})$. This follows since $m_0 \neq m_1$ or $\tau_i^{(0)} \neq \tau_i^{(1)}$.
- $f(m_0, \tau_i^{(0)}) = f(m_1, \tau_i^{(1)})$. This follows since $c_{(d,i)}^{(0)} = f(m_0, \tau_i^{(0)})$, $c_{(d,i)}^{(1)} = f(m_1, \tau_i^{(1)})$, and $c_{(d,i)}^{(0)} = c_{(d,i)}^{(1)}$.

In either case we find a collision with one query in S_0 and the other query in S_1 .

We are left to argue that the antecedent in Equation 18.4 implies that Case 1 or Case 2 holds.

If Case 1 holds then we are done. So suppose that Case 1 does not hold, which means that $(c_{\mathbf{p}(i,j)}^{(0)})_{j \in [d]} = (c_{\mathbf{p}(i,j)}^{(1)})_{j \in [d]}$ and $(c_{\bar{\mathbf{p}}(i,j)}^{(0)})_{j \in [d]} = (c_{\bar{\mathbf{p}}(i,j)}^{(1)})_{j \in [d]}$ (and in particular that $c_{(d,i)}^{(0)} = c_{(d,i)}^{(1)}$). If $m_0 \neq m_1$ then Case 2 holds. Suppose instead that $\text{auth}_0 \neq \text{auth}_1$. The only way for the two authentication paths to differ in this case is for the Merkle salts to differ ($\tau_i^{(0)} \neq \tau_i^{(1)}$), in which case again Case 2 holds. \square

General case: opening multiple leaves We now consider the collision lemma for the general case, where opening proofs may authenticate multiple leaves. Consider a Merkle commitment rt , as well as: two index sets I_0, I_1 ; two vectors $\mathbf{a}_0, \mathbf{a}_1$; two opening proofs pf_0, pf_1 . Suppose that both openings are valid, that is, $\text{MT.Check}^f(\text{rt}, I_0, \mathbf{a}_0, \text{pf}_0) = 1$ and $\text{MT.Check}^f(\text{rt}, I_1, \mathbf{a}_1, \text{pf}_1) = 1$.

Intuitively, and similarly to the case of a single leaf, if there is a position $i \in I_0 \cap I_1$ such that $\mathbf{a}_0[i] \neq \mathbf{a}_1[i]$ then we can find a collision in (the union of the relevant) query-answer traces. Informally, the prior argument works applied to the authentication paths for the index i in the opening proofs pf_0 and pf_1 (and the subset of query-answer pairs used to validate these paths).

Less obvious is the fact that we cannot expect (in general) to find a collision if $\text{pf}_0 \neq \text{pf}_1$. This is because the two opening proofs are for potentially different index sets I_0 and I_1 , and so the opening proofs can be different. Nevertheless, we can prove that if the opening proofs are for the same index set ($I_0 = I_1$) and they are different ($\text{pf}_0 \neq \text{pf}_1$) then we can find a collision.

We formally state and prove this below.

Lemma 18.3.2. *Let $\text{MT} := \text{MT}[\sigma, \Sigma, \ell, s]$. Fix arbitrary choices of oracle $f \in \mathcal{U}(\sigma)$ and:*

- a Merkle commitment $\text{rt} \in \{0, 1\}^\sigma$;
- a set $I_0 \subseteq [\ell]$, vector $\mathbf{a}_0 \in \Sigma^{I_0}$, and opening proof pf_0 ;
- a set $I_1 \subseteq [\ell]$, vector $\mathbf{a}_1 \in \Sigma^{I_1}$, and opening proof pf_1 .

For $b \in \{0, 1\}$, let S_b be the query-answer trace of $\text{MT.Check}^f(\text{rt}, I_b, \mathbf{a}_b, \text{pf}_b)$. Then the following implication holds:

$$\left[\begin{array}{l} \left(\exists i \in I_0 \cap I_1 : \mathbf{a}_0[i] \neq \mathbf{a}_1[i] \right) \\ \text{or} \\ I_0 = I_1 \wedge \text{pf}_0 \neq \text{pf}_1 \\ \wedge \text{MT.Check}^f(\text{rt}, I_0, \mathbf{a}_0, \text{pf}_0) = 1 \\ \wedge \text{MT.Check}^f(\text{rt}, I_1, \mathbf{a}_1, \text{pf}_1) = 1 \end{array} \right] \Rightarrow \left[\begin{array}{l} \exists (x_0, y_0) \in S_0, (x_1, y_1) \in S_1 \\ \text{with } x_0 \neq x_1 \text{ such that } y_0 = y_1 \end{array} \right]. \quad (18.5)$$

Proof. Fix $b \in \{0, 1\}$. The opening proof pf_b for the set I_b has the following structure:

$$\text{pf}_b = \left(\tau_i^{(b)}, (c_{\bar{p}(i,j)}^{(b)})_{j \in \{1, \dots, d\}} \right)_{i \in I_b}.$$

Above, the value $\tau_i^{(b)}$ is the Merkle salt used to authenticate leaf i , and the values $(c_{\bar{p}(i,j)}^{(b)})_{j \in \{1, \dots, d\}}$ are the commitments labeling vertices in $\text{copath}(i)$ used to authenticate leaf i .

Moreover, the computation $\text{MT.Check}^f(\text{rt}, I_b, \mathbf{a}_b, \text{pf}_b)$ obtains as answers from f the commitments corresponding to vertices in $\text{path}(i)$. We denote these commitments as

$$(c_{p(i,j)}^{(b)})_{j \in \{0, 1, \dots, d\}}.$$

Finally, $\text{MT.Check}^f(\text{rt}, I_b, \mathbf{a}_b, \text{pf}_b) = 1$ implies that $\text{rt} = c_{(0,1)}^{(b)}$.

In particular, we deduce that $c_{(0,1)}^{(0)} = c_{(0,1)}^{(1)}$.

We consider two cases.

1. *Case 1: there exist $i \in I_0 \cap I_1$ and $j \in [d]$ such that $c_{p(i,j)}^{(0)} \neq c_{p(i,j)}^{(1)}$ or $c_{p(i,j)}^{(0)} \neq c_{\bar{p}(i,j)}^{(1)}$.*

Let j be the minimal index satisfying this condition (for an arbitrary choice of i).

- If $p(i, j)$ is an odd vertex then for $b \in \{0, 1\}$ set $(c_L^{(b)}, c_R^{(b)}) := (c_{p(i,j)}^{(b)}, c_{\bar{p}(i,j)}^{(b)})$.
- If $p(i, j)$ is an even vertex then for $b \in \{0, 1\}$ set $(c_L^{(b)}, c_R^{(b)}) := (c_{\bar{p}(i,j)}^{(b)}, c_{p(i,j)}^{(b)})$.

For $b \in \{0, 1\}$ the pair $(c_L^{(b)}, c_R^{(b)})$ is the query in S_b (see Item 2b in the description of MT.Check).

These two queries are a collision for f due to the following.

- $(c_L^{(0)}, c_R^{(0)}) \neq (c_L^{(1)}, c_R^{(1)})$.
This follows since $c_{p(i,j)}^{(0)} \neq c_{p(i,j)}^{(1)}$ or $c_{\bar{p}(i,j)}^{(0)} \neq c_{\bar{p}(i,j)}^{(1)}$.
- $f(c_L^{(0)}, c_R^{(0)}) = f(c_L^{(1)}, c_R^{(1)})$.
This follows since $c_{p(i,j-1)}^{(0)} = f(c_L^{(0)}, c_R^{(0)})$, $c_{p(i,j-1)}^{(1)} = f(c_L^{(1)}, c_R^{(1)})$, and $c_{p(i,j-1)}^{(0)} = c_{p(i,j-1)}^{(1)}$. The last equality holds because j is a minimal index satisfying the condition for this choice of i (and we know that $c_{(0,1)}^{(0)} = c_{(0,1)}^{(1)}$).

2. *Case 2: there exists $i \in I_0 \cap I_1$ such that $c_{(d,i)}^{(0)} = c_{(d,i)}^{(1)}$ and such that $\mathbf{a}_0[i] \neq \mathbf{a}_1[i]$ or $\tau_i^{(0)} \neq \tau_i^{(1)}$.*

For $b \in \{0, 1\}$ the pair $(\mathbf{a}_b[i], \tau_i^{(b)})$ is a query in S_b (see Item 2a in the description of MT.Check).

These two queries are a collision for f due to the following.

- $(\mathbf{a}_0[i], \tau_i^{(0)}) \neq (\mathbf{a}_1[i], \tau_i^{(1)})$.
This follows since $\mathbf{a}_0[i] \neq \mathbf{a}_1[i]$ or $\tau_i^{(0)} \neq \tau_i^{(1)}$.
- $f(\mathbf{a}_0[i], \tau_i^{(0)}) = f(\mathbf{a}_1[i], \tau_i^{(1)})$.
This follows since $c_{(d,i)}^{(0)} = f(\mathbf{a}_0[i], \tau_i^{(0)})$, $c_{(d,i)}^{(1)} = f(\mathbf{a}_1[i], \tau_i^{(1)})$, and $c_{(d,i)}^{(0)} = c_{(d,i)}^{(1)}$.

In either case we find a collision with one query in S_0 and the other query in S_1 .

We are left to argue that the antecedent in Equation 18.5 implies that Case 1 or Case 2 holds.

Suppose that there exists $i \in I_0 \cap I_1$ such that $\mathbf{a}_0[i] \neq \mathbf{a}_1[i]$. One of the following must hold.

- If there exists $j \in [d]$ such that $c_{p(i,j)}^{(0)} \neq c_{p(i,j)}^{(1)}$, then Case 1 holds.
- If instead $(c_{p(i,j)}^{(0)})_{j \in [d]} = (c_{p(i,j)}^{(1)})_{j \in [d]}$, then in particular $c_{(d,i)}^{(0)} = c_{(d,i)}^{(1)}$ and so Case 2 holds.

Alternatively, suppose that $I_0 = I_1$ and $\text{pf}_0 \neq \text{pf}_1$. If Case 1 holds then we are done. So suppose that Case 1 does not hold; in particular we know that $(c_{\bar{p}(i,j)}^{(0)})_{j \in [d]} = (c_{\bar{p}(i,j)}^{(1)})_{j \in [d]}$. If so, the only way for $\text{pf}_0 \neq \text{pf}_1$ to hold is that there exists $i \in I_0 \cap I_1$ such that $\tau_i^{(0)} \neq \tau_i^{(1)}$. Moreover for this i we know that $c_{(d,i)}^{(0)} = c_{(d,i)}^{(1)}$ because Case 1 does not hold. Hence Case 2 holds. \square

18.4 Binding

The basic security property of the Merkle commitment scheme is that it is *binding*: a Merkle commitment cannot be opened at the same locations to different values, regardless of the Merkle opening proofs. In more detail, the probability, over the choice of random oracle, that an adversary succeeds in this task is small relative to its query complexity. The salt size s does not play any role in this property, and in particular it could be zero. This property is analogous to the one that we studied for the basic commitment scheme CM in Chapter 17 (see Lemma 17.1.1 in Section 17.1).

Later sections actually rely on a stronger property known as extractability, which we discuss in Section 18.5. Nevertheless, it is useful to understand binding first in order to familiarize oneself with the basic structure of the Merkle commitment scheme.

Lemma 18.4.1 (MT is binding). *Let $\text{MT} := \text{MT}[\sigma, \Sigma, \ell, s]$. For every query bound $t \in \mathbb{N}$ and t -query algorithm A , if $t \geq 2(d+1)^2$ then*

$$\Pr \left[\begin{array}{l} \exists i \in I_0 \cap I_1 \text{ s.t. } \mathbf{a}_0[i] \neq \mathbf{a}_1[i] \\ \wedge \text{MT.Check}^f(\mathbf{rt}, I_0, \mathbf{a}_0, \mathbf{pf}_0) = 1 \\ \wedge \text{MT.Check}^f(\mathbf{rt}, I_1, \mathbf{a}_1, \mathbf{pf}_1) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{rt}, I_0, \mathbf{a}_0, \mathbf{pf}_0, I_1, \mathbf{a}_1, \mathbf{pf}_1) \leftarrow A^f \end{array} \right] \leq \frac{1}{2} \cdot \frac{t^2}{2^\sigma}.$$

Informally, by the collision lemma in Section 18.3, breaking the binding property implies a collision in the query-answer traces of the two executions of MT.Check . The proof below upper bounds the probability that the adversary produces outputs that results in such a collision.

Note that a straightforward analysis would consider the probability of a collision in the query-answer trace resulting from concatenating the adversary's trace and the query-answer traces of the two executions of MT.Check ; this would yield a bound of $\binom{t+2(d+1)}{2} \cdot \frac{1}{2^\sigma}$. The proof below improves on this upper bound via a more careful analysis of which collisions are relevant.

Proof. Let \mathbf{tr} be the query-answer trace of A^f . We define two events:

- E is the event that A 's output satisfies the conditions in the probability statement; and
- E_{col} is the event that \mathbf{tr} contains a collision.

Our goal is to upper bound the probability that E holds.

We break the analysis into two cases as follows:

$$\Pr[E] = \Pr[E \wedge E_{\text{col}}] + \Pr[E \wedge \overline{E_{\text{col}}}] .$$

We upper bound each term individually, yielding the claimed upper bound.

1. The probability of $E \wedge E_{\text{col}}$ is upper bounded by the probability of E_{col} , which by Lemma 3.3.1 is at most $\frac{1}{2} \cdot \frac{(t-1) \cdot t}{2^\sigma}$.
2. The event $E \wedge \overline{E_{\text{col}}}$ implies that
 - \mathbf{tr} contains no collisions,
 - $\exists i \in I_0 \cap I_1$ s.t. $\mathbf{a}_0[i] \neq \mathbf{a}_1[i]$,
 - $\text{MT.Check}^f(\mathbf{rt}, I_0, \mathbf{a}_0, \mathbf{pf}_0) = 1$, and
 - $\text{MT.Check}^f(\mathbf{rt}, I_1, \mathbf{a}_1, \mathbf{pf}_1) = 1$.

Let $i \in I_0 \cap I_1$ be the minimal index such that $\mathbf{a}_0[i] \neq \mathbf{a}_1[i]$. Let \mathbf{auth}_0 and \mathbf{auth}_1 be the authentication paths for location i in \mathbf{pf}_0 and \mathbf{pf}_1 respectively. Define the query-answer traces:

- S_0 is the query-answer trace of $\text{MT.Check}^f(\mathbf{rt}, i, \mathbf{a}_0[i], \mathbf{auth}_0)$; and
- S_1 is the query-answer trace of $\text{MT.Check}^f(\mathbf{rt}, i, \mathbf{a}_1[i], \mathbf{auth}_1)$.

Note that $\text{MT.Check}^f(\mathbf{rt}, I_0, \mathbf{a}_0, \mathbf{pf}_0) = 1$ implies that $\text{MT.Check}^f(\mathbf{rt}, i, \mathbf{a}_0[i], \mathbf{auth}_0) = 1$, and similarly $\text{MT.Check}^f(\mathbf{rt}, I_1, \mathbf{a}_1, \mathbf{pf}_1) = 1$ implies that $\text{MT.Check}^f(\mathbf{rt}, i, \mathbf{a}_1[i], \mathbf{auth}_1) = 1$.

By Lemma 18.3.2 we get a collision in these traces: there are distinct queries x_0, x_1 and an answer y such that $(x_0, y) \in S_0$ and $(x_1, y) \in S_1$. Thus, we show that

$$\Pr[E \wedge \overline{E_{\text{col}}}] \leq \Pr \left[\begin{array}{l} \text{tr contains no collisions and there exist} \\ (x_0, y) \in S_0 \text{ and } (x_1, y) \in S_1 \text{ with } x_0 \neq x_1 \end{array} \right] \leq \frac{(d+1)^2}{2^\sigma}.$$

We are left to argue the second inequality above. Since tr contains no collisions, it cannot be that $(x_0, y) \in \text{tr}$ and $(x_1, y) \in \text{tr}$ simultaneously. Fix any pair of distinct queries (x_0, x_1) such that x_0 appears as a query in S_0 and x_1 appears as a query in S_1 , but not both queries are in tr . Then, the probability that $f(x_0) = f(x_1)$ is $\frac{1}{2^\sigma}$. Taking a union bound over all such pairs (x_0, x_1) , the probability that there exist $(x_0, y) \in S_0$ and $(x_1, y) \in S_1$ with $x_0 \neq x_1$ is at most $\frac{|S_0| \cdot |S_1|}{2^\sigma}$. The claimed bound since $|S_0|, |S_1| \leq d+1$ (MT.Check makes one query to compute the commitment $c_{(d,i)}$ and then d queries to compute the commitments $(c_{p(i,j)})_{j \in \{0,1,\dots,d-1\}}$).

We conclude that

$$\begin{aligned} \Pr[E] &= \Pr[E \wedge E_{\text{col}}] + \Pr[E \wedge \overline{E_{\text{col}}}] \\ &\leq \frac{1}{2} \cdot \frac{(t-1) \cdot t}{2^\sigma} + \frac{(d+1)^2}{2^\sigma}. \end{aligned}$$

Finally, if $t \geq 2(d+1)^2$ then the above expression is at most $\frac{1}{2} \cdot \frac{t^2}{2^\sigma}$. \square

18.5 Extractability

We prove that the Merkle commitment scheme is *extractable*, a security property that informally states that if an adversary outputs a Merkle commitment and then subsequently opens the Merkle commitment at a subset of locations then the adversary “knew” the opening at commitment time. In Section 18.5.1 we study this property for one commitment and in Section 18.5.2 we study it for multiple commitments.

18.5.1 Extraction for one commitment

The extractability property is formalized via an efficient algorithm MT.Extract known as the *extractor*. This is analogous to the extractability property that we proved for the basic commitment scheme CM in Chapter 17 (see Section 17.2), but for a major difference. Namely, *we cannot expect to extract an entire message for a given Merkle commitment*.

The adversary may output a Merkle commitment obtained from a partial tree, and then subsequently open parts of this partial tree. Hence MT.Extract (an efficient algorithm) cannot output an entire message whose Merkle commitment is the one output by the adversary, because that would entail, e.g., inverting the random oracle.

In light of the above, the extractability property only requires MT.Extract to output a message that agrees with the adversary only at locations that the adversary knows how to open. In particular, if the adversary never provides an opening for certain locations then no requirements are imposed on those locations of the extracted message.

In more detail, we consider an experiment of the following form.

- First, in a commitment phase, the adversary, given oracle access to the random oracle, performs a computation and outputs a Merkle commitment rt and a private state aux . Let tr be the query-answer trace of the adversary in this phase.
- Then, in an opening phase, the adversary, given oracle access to the random oracle and as input the private state aux , continues its computation and outputs an index set I , opening values \mathbf{a} , and opening proof pf .

Extractability states that if the adversary's output satisfies $\text{MT}.\text{Check}^f(\text{rt}, I, \mathbf{a}, \text{pf}) = 1$ then the extractor $\text{MT}.\text{Extract}$, given the Merkle commitment rt and the query-answer trace tr , outputs a message \mathbf{m} and opening trapdoor td such that \mathbf{m} agrees with \mathbf{a} on I and td leads to the opening proof pf when specialized to I . (Up to a small error.) In other words, the adversary "knew" the local opening in the commitment phase because the extractor found it by examining only the query-answer trace of the adversary relevant for producing the commitment (without seeing the subsequent query-answer trace in the opening phase).

The lemma below formally states the property and provides the extraction error.

Lemma 18.5.1 (MT is extractable). *Let $\text{MT} := \text{MT}[\sigma, \Sigma, \ell, s]$. There exists a deterministic algorithm $\text{MT}.\text{Extract}$ such that for every query bound $t \in \mathbb{N}$ and t -query algorithm A*

$$\Pr \left[\begin{array}{l} \text{MT}.\text{Check}^f(\text{rt}, I, \mathbf{a}, \text{pf}) = 1 \\ \wedge (\mathbf{m}[I] \neq \mathbf{a} \vee \text{pf}' \neq \text{pf}) \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\text{rt}, \text{aux}) \xleftarrow{\text{tr}} A^f \\ (\mathbf{m}, \text{td}) \leftarrow \text{MT}.\text{Extract}(\text{rt}, \text{tr}) \\ (I, \mathbf{a}, \text{pf}) \leftarrow A^f(\text{aux}) \\ \text{pf}' \leftarrow \text{MT}.\text{Open}^f(\text{td}, I) \end{array} \right] \leq \kappa_{\text{MT}}(\sigma, t, \ell).$$

Above:

- $\kappa_{\text{MT}}(\sigma, t, \ell) \leq \frac{1}{2} \cdot \frac{(t-1) \cdot t}{2^\sigma} + \frac{2 \cdot (d+1) \cdot \ell}{2^\sigma}$ if $t \geq 4\ell+1$, which in turn is at most $\frac{1}{2} \cdot \frac{t^2}{2^\sigma}$ if $t \geq 4 \cdot (d+1) \cdot \ell$;
- $\text{MT}.\text{Extract}$ runs in time $\text{et}_{\text{MT}}(\sigma, \Sigma, \ell, s, t) = O(t \cdot \ell \cdot (\log |\Sigma| + s + \sigma))$.

We describe the intuition behind the extractor $\text{MT}.\text{Extract}$, and then provide a formal description of it. The extractor $\text{MT}.\text{Extract}$ is tasked with finding a message that "explains" any future opening to a given Merkle commitment rt , given the query-answer trace of the adversary that produced the Merkle commitment. Intuitively, all the information about possible openings of the adversary is available in the query-answer trace of the adversary's computation till it outputs the Merkle commitment rt . So the extractor merely needs to organize the query-answer trace into a partial tree, and read off the leaves of this tree, filling in with an arbitrary value any missing leaves. Because a random oracle is inversion resistant and collision resistant, no adversary is able to provide any other openings other than those contained in the leaves output this way, up to a small probability of error.

In order to define the extractor, it will be convenient to partition the queries into three types: leaf queries (for computing the leaves of the Merkle tree), inner vertex queries (for computing the internal vertices of the tree), and other (queries that do not match these previous categories).

Definition 18.5.2. *Let $\text{tr} = ((x_i, y_i))_{i \in [t]}$ be a query-answer trace for an oracle $f: \{0, 1\}^* \rightarrow \{0, 1\}^\sigma$. We partition the query-answer pairs in tr according to three **types of queries**.*

1. Leaf vertex queries. A set tr_{leaf} that contains queries-answer pairs (x, y) where $x = (m, \tau)$ for some message entry $m \in \Sigma$ and salt $\tau \in \{0, 1\}^s$.

2. Inner vertex queries. A set tr_{inner} that contains queries-answer pairs (x, y) where $x \in \{0, 1\}^{2\sigma}$.
3. Other queries. A set tr_{other} that contains all other query-answer pairs.

We describe the extractor MT.Extract .

Construction 18.5.3. Given as input a Merkle commitment $\text{rt} \in \{0, 1\}^\sigma$ and query-answer trace $\text{tr} = ((x_i, y_i))_{i \in [t]}$, MT.Extract works as follows.

1. If rt is not the answer to any query in tr , then output $(\mathbf{m}, \text{td}) := (\perp, \perp)$.
2. Let $\text{tr}_{\text{leaf}}, \text{tr}_{\text{inner}}, \text{tr}_{\text{other}}$ be the partitioning of tr according to Definition 18.5.2.
3. Label the binary tree T_ℓ (see Definition 18.1.1) as follows:
 - The root of T_ℓ is labeled with rt .
 - While there is a query-answer pair $(x, y) \in \text{tr}_{\text{inner}}$ where y is a label of a vertex in T_ℓ , set the label of the left child to be $[x]_L$ and the label of the right child to be $[x]_R$ (where $[x]_L$ and $[x]_R$ are the left half and right half of x respectively). Remove (x, y) from tr_{inner} .
 - While there is a query-answer pair $(x, y) \in \text{tr}_{\text{leaf}}$ where y is the label of the i -th leaf of T_ℓ for some $i \in [\ell]$, set $m_i := m$ and $\tau_i := \tau$, where $x = (m, \tau)$. Remove (x, y) from tr_{leaf} .
4. Set arbitrarily the values of any label in T_ℓ or pair (m_i, τ_i) not defined in the previous step.
5. Set the commitments $C := ((c_{(j,i)})_{i \in [2^j]})_{j=0}^d$, where $c_{(j,i)}$ is the label of the i -th vertex in layer j of the tree T_ℓ .
6. Set the message vector $\mathbf{m} := (m_i)_{i \in [\ell]}$.
7. Set the opening trapdoor $\text{td} := (\boldsymbol{\tau}, C)$, where $\boldsymbol{\tau} := (\tau_i)_{i \in [\ell]}$.
8. Output (\mathbf{m}, td) .

Proof of Lemma 18.5.1. We discuss the time complexity of MT.Extract in Remark 18.5.5. Below we analyze its success probability.

Consider the following query-answer traces:

- tr is the query-answer trace of the execution $(\text{rt}, \text{aux}) \xleftarrow{\text{tr}} A^f$;
- tr' is the query-answer trace of the execution $(I, \mathbf{a}, \text{pf}) \xleftarrow{\text{tr}'} A^f(\text{aux})$;
- S is the query-answer trace of the execution $\text{MT.Check}^f(\text{rt}, I, \mathbf{a}, \text{pf})$.

Fix any $t_1, t_2 \in \mathbb{N}$ with $t_1 + t_2 \leq t$. We prove the claimed upper bound conditioned on the event that $t_1 = |\text{tr}|$ and $t_2 = |\text{tr}'|$. The lemma follows from this because the upper bound holds for every $t_1, t_2 \in \mathbb{N}$ with $t_1 + t_2 \leq t$, and every computation of the t -query adversary A implies a setting of $t_1, t_2 \in \mathbb{N}$ with $t_1 + t_2 \leq t$.

We define several events:

- E is the event that the conditions in the probability statement hold;
- E_{col} is the event that tr contains a collision;
- E_{tree} is the event that $T_\ell \neq \hat{T}_\ell$ where
 - T_ℓ is the tree generated in the execution of $\text{MT.Extract}(\text{rt}, \text{tr})$ and
 - \hat{T}_ℓ is the tree generated in the execution of $\text{MT.Extract}(\text{rt}, \text{tr} \parallel \text{tr}')$.
- E_{sub} is the event that $S \not\subseteq \text{tr}$ and $\text{MT.Check}^f(\text{rt}, I, \mathbf{a}, \text{pf}) = 1$.

Our goal is to upper bound the probability that E holds. We break the analysis into four cases:

$$\Pr[E] \leq \Pr[E_{\text{col}}] + \Pr[E_{\text{tree}} \mid \overline{E_{\text{col}}}] + \Pr[E_{\text{sub}} \mid \overline{E_{\text{col}}} \wedge \overline{E_{\text{tree}}}] + \Pr[E \mid \overline{E_{\text{col}}} \wedge \overline{E_{\text{tree}}} \wedge \overline{E_{\text{sub}}}] .$$

We upper bound each term individually, yielding the claimed upper bound.

1. An upper bound on the probability of the event E_{col} follows directly from Lemma 3.3.1:

$$\Pr[E_{\text{col}}] \leq \frac{1}{2} \cdot \frac{(t_1 - 1) \cdot t_1}{2^\sigma} .$$

2. We upper bound the probability of the event $E_{\text{tree}} \mid \overline{E_{\text{col}}}$.

If $T_\ell \neq \hat{T}_\ell$ then there exists a query in tr' that is not in tr with an answer that equals a non-dummy label in T_ℓ . (A non-dummy label is one added in Item 3, rather than Item 4, of the extractor.) We bound the probability that such a query exists.

Since the event E_{col} does not hold (tr contains no collisions), each vertex in the tree T_ℓ is labeled at most once. There are at most $\min\{2t_1 + 1, 2\ell\}$ non-dummy labels in T_ℓ : the root vertex is labeled separately, and each query in tr results in at most 2 newly-labeled vertices; moreover, there are at most 2ℓ vertices in the tree. Hence the probability that any fixed query in tr' (that is not in tr) has an answer that equals one of the labels in T_ℓ is at most $\frac{\min\{2t_1 + 1, 2\ell\}}{2^\sigma}$.

Taking a union bound over all queries in tr' ,

$$\Pr[E_{\text{tree}} \mid \overline{E_{\text{col}}}] \leq |\text{tr}'| \cdot \frac{\min\{2t_1 + 1, 2\ell\}}{2^\sigma} = t_2 \cdot \frac{\min\{2t_1 + 1, 2\ell\}}{2^\sigma} .$$

3. We upper bound the probability of the event $E_{\text{sub}} \mid \overline{E_{\text{col}}} \wedge \overline{E_{\text{tree}}}$.

For every $i \in I$, let $S_i \subseteq S$ be the query-answer trace of $\text{MT.Check}^f(\text{rt}, i, \mathbf{a}[i], \text{auth}_i)$, where auth_i is the authentication path for location i in pf .

Suppose that E_{sub} holds. Then $S \not\subseteq \text{tr}$, so there exists $i \in I$ such that $S_i \not\subseteq \text{tr}$. Moreover, $\text{MT.Check}^f(\text{rt}, i, \mathbf{a}[i], \text{auth}_i) = 1$, so the query-answer trace S_i can be viewed as a path that “ends” in the Merkle commitment rt , and thus “connects” somewhere into T_ℓ (the tree generated in the execution of $\text{MT.Extract}(\text{rt}, \text{tr})$). In sum, if E_{sub} holds then there exists a query in S_i that is not in tr with an answer that equals a non-dummy label in T_ℓ . (A non-dummy label is one added in Item 3, rather than Item 4, of the extractor.)

Since the event E_{col} does not hold (tr contains no collisions), each vertex in the tree T_ℓ is labeled at most once. There are at most $\min\{2t_1 + 1, 2\ell\}$ non-dummy labels in T_ℓ (we argued this in the prior point). Moreover, $T_\ell = \hat{T}_\ell$ because E_{tree} does not hold. Hence there are no queries in $\text{tr}' \setminus \text{tr}$ that have an answer that equals a non-dummy label in T_ℓ (otherwise, there would be a leaf in \hat{T}_ℓ labeled differently from the corresponding leaf in T_ℓ). Hence the probability that any fixed query in S_i (that is not in $\text{tr} \cup \text{tr}'$) has an answer that equals a non-dummy label in T_ℓ is at most $\frac{\min\{2t_1 + 1, 2\ell\}}{2^\sigma}$.

Considering any choice of $I \subseteq [\ell]$ and $i \in I$ and taking a union bound over all queries in S_i ,

$$\Pr[E_{\text{sub}} \mid \overline{E_{\text{col}}} \wedge \overline{E_{\text{tree}}}] \leq \max_{I \subseteq [\ell]} \max_{i \in I} |S_i| \cdot \frac{\min\{2t_1 + 1, 2\ell\}}{2^\sigma} \leq (d+1) \cdot \frac{\min\{2t_1 + 1, 2\ell\}}{2^\sigma} .$$

4. We show that the event E cannot happen if E_{col} does not hold, E_{tree} does not hold, and E_{sub} does not hold (in fact, it suffices to assume $\overline{E_{\text{col}}} \wedge \overline{E_{\text{sub}}}$ but we include $\overline{E_{\text{tree}}}$ for symmetry). We provide this as a claim (as we reuse it in later sections).

Claim 18.5.4. *It holds that*

$$\Pr \left[\begin{array}{l} \text{MT.Check}^f(\text{rt}, I, \mathbf{a}, \text{pf}) = 1 \\ \wedge (\mathbf{m}[I] \neq \mathbf{a} \vee \text{pf}' \neq \text{pf}) \\ \text{conditioned on} \\ \overline{E_{\text{col}}} \wedge \overline{E_{\text{tree}}} \wedge \overline{E_{\text{sub}}} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\text{rt}, \text{aux}) \xleftarrow{\text{tr}} A^f \\ (\mathbf{m}, \text{td}) \leftarrow \text{MT.Extract}(\text{rt}, \text{tr}) \\ (I, \mathbf{a}, \text{pf}) \leftarrow A^f(\text{aux}) \\ \text{pf}' \leftarrow \text{MT.Open}^f(\text{td}, I) \end{array} \right] = 0.$$

Proof. The conditioning means that all of the events $E_{\text{col}}, E_{\text{tree}}, E_{\text{sub}}$ do not hold. Suppose that $\text{MT.Check}^f(\text{rt}, I, \mathbf{a}, \text{pf}) = 1$ (as otherwise the claim holds trivially). Then, $S \subseteq \text{tr}$ (since E_{sub} does not hold). Below we argue that (under the conditioning) this implies that $\mathbf{m}[I] = \mathbf{a}$ and $\text{pf}' = \text{pf}$. Note that the opening proofs pf and pf' consist of authentication paths $(\text{auth}_i)_{i \in I}$ and $(\text{auth}'_i)_{i \in I}$. Moreover, from Equation 18.3 we know that

$$\begin{aligned} \text{MT.Check}^f(\text{rt}, I, \mathbf{a}, \text{pf}) &= \wedge_{i \in I} \text{MT.Check}^f(\text{rt}, i, \mathbf{a}[i], \text{auth}_i), \text{ and} \\ \text{MT.Check}^f(\text{rt}, I, \mathbf{m}[I], \text{pf}') &= \wedge_{i \in I} \text{MT.Check}^f(\text{rt}, i, \mathbf{m}[i], \text{auth}'_i). \end{aligned}$$

- We show that $\mathbf{m}[I] = \mathbf{a}$.

Fix $i \in I$ and consider the following query-answer traces:

- S_i is the query-answer trace of the execution $\text{MT.Check}^f(\text{rt}, i, \mathbf{a}[i], \text{auth}_i)$;
- S'_i is the query-answer trace of the execution $\text{MT.Check}^f(\text{rt}, i, \mathbf{m}[i], \text{auth}'_i)$.

We know that $S_i \subseteq S$ (since MT.Check validates each authentication path in the opening proof), so $S_i \subseteq \text{tr}$. Moreover, we know that $\text{MT.Check}^f(\text{rt}, i, \mathbf{a}[i], \text{auth}_i) = 1$ (since $\text{MT.Check}^f(\text{rt}, I, \mathbf{a}, \text{pf}) = 1$).

Next, the extractor MT.Extract , given input (rt, tr) , uses the query-answer pairs in tr to assign values to the entries of \mathbf{m} that have a valid authentication paths in tr to the Merkle commitment rt (see Item 3 of Construction 18.5.3); the remaining entries of \mathbf{m} are assigned arbitrary values (see Item 4 of Construction 18.5.3). Since $S_i \subseteq \text{tr}$, we know that the i -th entry of \mathbf{m} is assigned a non-arbitrary value (as the trace tr contains a path from the i -th leaf to the root), and hence $S'_i \subseteq \text{tr}$. Moreover, the opening trapdoor td output by MT.Extract enables MT.Open to output valid authentication paths for entries that are assigned non-arbitrary values, including the authentication path auth'_i , so we also deduce that $\text{MT.Check}^f(\text{rt}, i, \mathbf{m}[i], \text{auth}'_i) = 1$.

We have established that:

- S_i is contained in tr , and $\text{MT.Check}^f(\text{rt}, i, \mathbf{a}[i], \text{auth}_i) = 1$; and
- S'_i is contained in tr , and $\text{MT.Check}^f(\text{rt}, i, \mathbf{m}[i], \text{auth}'_i) = 1$.

Now suppose by way of contradiction that $\mathbf{m}[I] \neq \mathbf{a}$. By Lemma 18.3.2, tr contains a collision, which contradicts the conditioning.

- We show that $\text{pf}' = \text{pf}$.

Suppose by way of contradiction that $\text{pf}' \neq \text{pf}$. This means that there exists an index $i \in I$ such that $\text{auth}'_i \neq \text{auth}_i$. Above we argued that $\mathbf{m}[I] = \mathbf{a}$, which means that

$\mathbf{m}[i] = \mathbf{a}[i]$. Above we also argued that $\text{MT.Check}^f(\text{rt}, i, \mathbf{m}[i], \text{auth}'_i) = 1$ and that $\text{MT.Check}^f(\text{rt}, i, \mathbf{a}[i], \text{auth}_i) = 1$, and that the corresponding traces S'_i and S_i are in tr . By Lemma 18.3.1, tr contains a collision, which contradicts the conditioning.

□

We conclude that

$$\begin{aligned} \Pr[E] &\leq \Pr[E_{\text{col}}] + \Pr[E_{\text{tree}} \mid \overline{E_{\text{col}}}] + \Pr[E_{\text{sub}} \mid \overline{E_{\text{col}}} \wedge \overline{E_{\text{tree}}}] + \Pr[E \mid \overline{E_{\text{col}}} \wedge \overline{E_{\text{tree}}} \wedge \overline{E_{\text{sub}}}] \\ &\leq \frac{1}{2} \cdot \frac{(t_1 - 1) \cdot t_1}{2^\sigma} + \frac{t_2 \cdot \min\{2t_1 + 1, 2\ell\}}{2^\sigma} + \frac{(d + 1) \cdot \min\{2t_1 + 1, 2\ell\}}{2^\sigma} + 0 \\ &\leq \frac{1}{2} \cdot \frac{(t_1 - 1) \cdot t_1}{2^\sigma} + \frac{(t - t_1 + d + 1) \cdot \min\{2t_1 + 1, 2\ell\}}{2^\sigma} \\ &\leq \frac{1}{2} \cdot \frac{(t_1 - 1) \cdot t_1}{2^\sigma} + \frac{(t - t_1 + d + 1) \cdot 2\ell}{2^\sigma}. \end{aligned}$$

The expression is a convex function in t_1 so its maximum on the interval $t_1 \in \{0, 1, \dots, t\}$ is achieved at either $t_1 = 0$ or $t_1 = t$. Hence the expression is upper bounded from the maximum of these two options:

$$\Pr[E] \leq \max \left\{ \frac{(t + d + 1) \cdot 2\ell}{2^\sigma}, \frac{1}{2} \cdot \frac{(t - 1) \cdot t}{2^\sigma} + \frac{(d + 1) \cdot 2\ell}{2^\sigma} \right\}.$$

We conclude that if $t \geq 4\ell + 1$ then the first term is no more than the second term, and so

$$\Pr[E] \leq \frac{1}{2} \cdot \frac{(t - 1) \cdot t}{2^\sigma} + \frac{2 \cdot (d + 1) \cdot \ell}{2^\sigma}.$$

□

Remark 18.5.5 (time complexity of MT.Extract). We discuss how a straightforward realization of MT.Extract leads to the time complexity claimed in Lemma 18.5.1, and how in common parameter regimes this time complexity can be improved.

The dominant contribution comes from Item 3, which we discuss below. Indeed, all other steps (partitioning queries, giving arbitrary values to unlabeled vertices in the tree, and assembling the extractor outputs) can be performed in time at most $O((t + \ell) \cdot (\log |\Sigma| + s + \sigma))$.

Item 3 requires structuring certain information from the query-trace into a partial binary tree. This involves at most $\min\{2\ell, t\} = O(\ell)$ iterations (each iteration corresponds to a vertex in the tree and an entry in the trace), proceeding layer by layer from the root towards the leaves. Each iteration involves searching the query-answer trace for an entry in which the answer equals the label of the vertex. Each of these $O(\ell)$ searches takes at most time $O(t \cdot (\log |\Sigma| + s + \sigma))$, which leads to the time complexity $O(t \cdot \ell \cdot (\log |\Sigma| + s + \sigma))$ claimed in Lemma 18.5.1.

We can do better by using suitable data structures. Realizing Item 3 essentially requires a *static dictionary* because we need to perform $\min\{2\ell, t\}$ lookups into a (static) list of at most t entries, with each entry consisting of at most $w := O(\log |\Sigma| + s + \sigma)$ bits. One way to achieve this is to sort the query-answer trace by answers, and then perform lookups via binary search. Sorting the trace takes time $O(t \cdot \log t \cdot w)$ and performing each of the $\min\{2\ell, t\}$ lookups takes time $O(\log t \cdot w)$; this leads to a total time complexity of $O((t + \min\{2\ell, t\}) \cdot \log t \cdot w) = O(t \cdot \log t \cdot w)$.

The above expression improves on the time in Lemma 18.5.1 provided that $\log t = O(\ell)$.

18.5.2 Extraction for multiple commitments

We discuss how extractability extends to adversaries that output multiple Merkle commitments. Intuitively, this should follow directly from Lemma 18.5.1 via a union bound: if the adversary outputs n commitments and subsequently n openings, then the probability that there exists one of the openings that is valid but the extractor did not succeed for the corresponding commitment is at most n times the error in Lemma 18.5.1. This error can, however, be improved. Each execution of the MT extractor is applied to a (possibly) different Merkle commitment, but with (possibly different prefixes of) the *same* query-answer trace tr . This enables a tighter analysis.

The lemma below extends Lemma 18.5.1 to the case where an adversary outputs multiple Merkle commitments and the extractor is tasked to extract for those commitments that the adversary chooses to open (successfully). This is useful when extracting multiple times for different Merkle commitments but with the same query-answer trace. Similarly to Lemma 17.2.2 for the basic commitment scheme CM, we let the extractor $\text{MT}.\text{MultiExtract}$ be a stateful algorithm.

Lemma 18.5.6 (MT is multi-extractable). *Let $\text{MT} := \text{MT}[\sigma, \Sigma, \ell, s]$. There exists a deterministic **stateful** algorithm $\text{MT}.\text{MultiExtract}$ such that for every query bound $t \in \mathbb{N}$, t -query algorithm A , number of commitments $n \in \mathbb{N}$, and number of openings $k \in [n]$, with $6 \cdot k \cdot (d + 1) \leq t$:*

$$\Pr \left[\begin{array}{l} \exists j \in [k] : \\ \quad \text{MT.Check}^f(\text{rt}_{i_j}, I_{i_j}, \mathbf{a}_{i_j}, \text{pf}_{i_j}) = 1 \\ \quad \wedge (\mathbf{m}_{i_j}[I_{i_j}] \neq \mathbf{a}_{i_j} \vee \text{pf}'_{i_j} \neq \text{pf}_{i_j}) \\ \quad \text{or} \\ \exists i, i' \in [n] : \\ \quad \text{rt}_i = \text{rt}_{i'} \wedge (\mathbf{m}_i, \text{td}_i) \neq (\mathbf{m}_{i'}, \text{td}_{i'}) \end{array} \right] \leq 2 \cdot \frac{t^2}{2^\sigma}.$$

$$\begin{aligned} & f \leftarrow \mathcal{U}(\sigma) \\ & \text{aux}_0 := \perp \\ & \text{For } i = 1, \dots, n: \\ & \quad \bullet (\text{rt}_i, \text{aux}_i) \xleftarrow{\text{tr}_i} A^f(\text{aux}_{i-1}) \\ & \quad \bullet (\mathbf{m}_i, \text{td}_i) \leftarrow \text{MT}.\text{MultiExtract}(\text{rt}_i, \text{tr}_i) \\ & \quad ((i_j, I_{i_j}, \mathbf{a}_{i_j}, \text{pf}_{i_j}))_{j \in [k]} \leftarrow A^f(\text{aux}_n) \\ & \text{For } j \in [k]: \text{pf}'_{i_j} \leftarrow \text{MT}.\text{Open}^f(\text{td}_{i_j}, I_{i_j}) \end{aligned}$$

Moreover, the total running time of $\text{MT}.\text{MultiExtract}$ is $\text{et}_{\text{MT}}(\sigma, \Sigma, \ell, s, t, n) = O(n \cdot t \cdot \ell \cdot (\log |\Sigma| + s + \sigma))$ (across the n invocations).

Proof. The stateful extractor $\text{MT}.\text{MultiExtract}$ is similar to the (stateless) extractor $\text{MT}.\text{Extract}$ in Construction 18.5.3. The extractor $\text{MT}.\text{MultiExtract}$ internally maintains the concatenation of all query-answer traces received so far (after the i -th invocation it stores the trace $\text{tr} := \text{tr}_1 \| \dots \| \text{tr}_i$), and uses this trace to construct the message and trapdoor corresponding to the given commitment.

$\text{MT}.\text{MultiExtract}(\text{rt}_i, \text{tr}_i)$:

1. Append tr_i to the query-answer trace tr stored in the internal state.
2. Compute and output $(\mathbf{m}_i, \text{td}_i) := \text{MT}.\text{Extract}(\text{rt}_i, \text{tr})$.

By Lemma 18.5.1 each invocation of $\text{MT}.\text{Extract}$ runs in time at most $O(t \cdot \ell \cdot (\log |\Sigma| + s + \sigma))$, because the stored query-answer trace contains at most t entries. The n invocations yield the claimed total running time. The time complexity can be improved via a suitable use of dynamic data structures in the internal state (and foregoing $\text{MT}.\text{Extract}$ as a subroutine); see Remark 18.5.7.

Now we analyze the extraction error. Consider the following query-answer traces:

- $\text{tr} := \text{tr}_1 \| \dots \| \text{tr}_n$;
- tr' is the query-answer trace of the execution $((i_j, I_{i_j}, \mathbf{a}_{i_j}, \mathbf{pf}_{i_j}))_{j \in [k]} \leftarrow A^f(\text{aux}_n)$;
- for every $j \in [k]$, S_j is the query-answer trace of the execution $\text{MT.Check}^f(\text{rt}_{i_j}, I_{i_j}, \mathbf{a}_{i_j}, \mathbf{pf}_{i_j})$.

Fix any $t_1, t_2 \in \mathbb{N}$ with $t_1 + t_2 \leq t$. We prove the claimed upper bound conditioned on the event that $t_1 = |\text{tr}|$ and $t_2 = |\text{tr}'|$. The lemma follows from this because the upper bound holds for every $t_1, t_2 \in \mathbb{N}$ with $t_1 + t_2 \leq t$, and every computation of the t -query adversary A implies a setting of $t_1, t_2 \in \mathbb{N}$ with $t_1 + t_2 \leq t$.

We define several events.

- E is the event that the conditions in the probability statement hold.
- E_{col} is the event that tr contains a collision.
- $E_{\text{tree},1}$ is the event that there exists $j \in [k]$ with $T_\ell^{i_j} \neq \hat{T}_\ell^{i_j}$ where
 - $T_\ell^{i_j}$ is the tree generated during the execution of $\text{MT.Extract}(\text{rt}_{i_j}, \text{tr}_1 \| \dots \| \text{tr}_{i_j})$, and
 - $\hat{T}_\ell^{i_j}$ is the tree generated during the execution of $\text{MT.Extract}(\text{rt}_{i_j}, \text{tr} \| \text{tr}')$.
- $E_{\text{tree},2}$ is the event that there exist $i, i' \in [n]$ with $\text{rt}_i = \text{rt}_{i'}$ and $T_\ell^i \neq T_\ell^{i'}$ where
 - T_ℓ^i is the tree generated during the execution of $\text{MT.Extract}(\text{rt}_i, \text{tr}_1 \| \dots \| \text{tr}_i)$, and
 - $T_\ell^{i'}$ is the tree generated during the execution of $\text{MT.Extract}(\text{rt}_{i'}, \text{tr}_1 \| \dots \| \text{tr}_{i'})$.
- E_{tree} is the event $E_{\text{tree},1} \vee E_{\text{tree},2}$.
- E_{sub} is the event that there exists $j \in [k]$ with $S_j \not\subseteq \text{tr}$ and $\text{MT.Check}^f(\text{rt}_{i_j}, I_{i_j}, \mathbf{a}_{i_j}, \mathbf{pf}_{i_j}) = 1$.

The trees $(T_\ell^i)_{i \in [n]}$ defined in E_{tree} are the trees generated during the execution of MT.MultiExtract : for every $i \in [n]$, $\text{MT.MultiExtract}(\text{rt}_i, \text{tr}_i)$ corresponds to $\text{MT.Extract}(\text{rt}_i, \text{tr}_1 \| \dots \| \text{tr}_i)$.

Our goal is to upper bound the probability that E holds. We break the analysis into five cases:

$$\begin{aligned} \Pr[E] &\leq \Pr[E_{\text{col}}] + \Pr[E_{\text{tree},1} \mid \overline{E_{\text{col}}}] + \Pr[E_{\text{tree},2} \mid \overline{E_{\text{col}}}] \\ &\quad + \Pr[E_{\text{sub}} \mid \overline{E_{\text{col}}} \wedge \overline{E_{\text{tree}}}] + \Pr[E \mid \overline{E_{\text{col}}} \wedge \overline{E_{\text{tree}}} \wedge \overline{E_{\text{sub}}}] . \end{aligned}$$

We upper bound each term individually, yielding the claimed upper bound.

1. An upper bound on the probability of the event E_{col} follows directly from Lemma 3.3.1:

$$\Pr[E_{\text{col}}] \leq \frac{1}{2} \cdot \frac{(t_1 - 1) \cdot t_1}{2^\sigma} .$$

2. We upper bound the probability of the event $E_{\text{tree},1} \mid \overline{E_{\text{col}}}$.

Since $E_{\text{tree},1}$ holds (there exists $j \in [k]$ such that $T_\ell^{i_j} \neq \hat{T}_\ell^{i_j}$), $\text{tr}_{i_j+1} \| \dots \| \text{tr}_n \| \text{tr}'$ contains a query that is not in $\text{tr}_1 \| \dots \| \text{tr}_{i_j}$ with an answer that equals a non-dummy label in $T_\ell^{i_j}$.

Since the event E_{col} does not hold (tr contains no collisions), the aforementioned query cannot be in $\text{tr}_{i_j+1} \| \dots \| \text{tr}_n$ because that would form a collision in tr . Hence it suffices to upper bound the probability that there exists a query in tr' that is not in $\text{tr}_1 \| \dots \| \text{tr}_{i_j}$ with an answer that equals a non-dummy label in $T_\ell^{i_j}$.

Since the event E_{col} does not hold (tr contains no collisions), each vertex in any of the trees $(T_\ell^i)_{i \in [n]}$ is labeled at most once. The number of non-dummy labels in all trees $(T_\ell^{i_j})_{j \in [k]}$ is upper bounded by $\min\{3t_1, k \cdot 2\ell\}$. The upper bound $3t_1$ is since the trees are constructed

from a trace of length at most t_1 and each query can add at most three labels in one tree (most queries add two labels but the first one in a tree adds three since the root is added as a label as well). The upper bound $k \cdot 2\ell$ is since each tree has at most 2ℓ labels. For every query in tr' , the probability that the query answer equals one of the these non-dummy labels is at most $\frac{\min\{3t_1, k \cdot 2\ell\}}{2^\sigma}$.

Taking a union bound over all queries in tr' ,

$$\Pr [E_{\text{tree},1} \mid \overline{E_{\text{col}}} \leq | \text{tr}' | \cdot \frac{\min\{3t_1, k \cdot 2\ell\}}{2^\sigma} = t_2 \cdot \frac{\min\{3t_1, k \cdot 2\ell\}}{2^\sigma}.$$

3. We upper bound the probability of the event $E_{\text{tree},2} \mid \overline{E_{\text{col}}}$.

Since the event E_{col} does not hold (tr contains no collisions), each vertex in any of the trees $(T_\ell^i)_{i \in [n]}$ is labeled at most once.

For every $i, i' \in [n]$ with $i < i'$ and $\text{rt}_i = \text{rt}_{i'}$, if $T_\ell^i \neq T_\ell^{i'}$ then there must be a query in $\text{tr}_1 \| \dots \| \text{tr}_{i'}$ that is not in $\text{tr}_1 \| \dots \| \text{tr}_i$ with an answer that equals a non-dummy label T_ℓ^i . Below we upper bound the probability that such a pair of indexes $i, i' \in [n]$ exist.

Suppose that, for $j \in [t_1]$, the adversary's j -th query happens in iteration i' of the loop (i.e., at the end of this iteration the adversary outputs $\text{rt}_{i'}$). The number of non-dummy labels in $T_\ell^1, \dots, T_\ell^{i'-1}$ is at most $2(j-1)$: each query-answer pair leads to at most two new labels in a single tree, and we exclude the roots since that would cause a collision inside tr . Hence the probability that the answer of the j -th query matches any of the non-dummy labels is at most $\frac{2(j-1)}{2^\sigma}$.

Summing over all t_1 queries in tr ,

$$\Pr [E_{\text{tree},2} \mid \overline{E_{\text{col}}} \leq \sum_{j=1}^{t_1} \frac{2(j-1)}{2^\sigma} = \frac{(t_1-1) \cdot t_1}{2^\sigma}.$$

4. We upper bound the probability of the event $E_{\text{sub}} \mid \overline{E_{\text{col}}} \wedge \overline{E_{\text{tree}}}$.

For every $j \in [k]$ and $\iota \in I_{i,j}$, let $S_{j,\iota} \subseteq S_j$ be the query-answer trace of the execution $\text{MT.Check}^f(\text{rt}_{i,j}, \iota, \mathbf{a}[\iota], \text{auth}_\iota)$, where auth_ι is the authentication path for location ι in $\text{pf}_{i,j}$.

Suppose that E_{sub} holds. Then there exists $j \in [k]$ such that $S_j \not\subseteq \text{tr}$. Let $S'_j := S_{j,\iota}$ where ι is any index such that $S_{j,\iota} \not\subseteq \text{tr}$. Moreover, $\text{MT.Check}^f(\text{rt}, i, \mathbf{a}[i], \text{auth}_i) = 1$, so the query-answer trace S'_j can be viewed as a path that "ends" in the Merkle commitment $\text{rt}_{i,j}$, and thus "connects" somewhere into $T_\ell^{i,j}$. In sum, if E_{sub} holds then there exists a query in S'_j that is not in tr with an answer that equals a non-dummy label in $T_\ell^{i,j}$.

Since the event E_{col} does not hold (tr contains no collisions), each vertex in any of the trees $(T_\ell^i)_{i \in [n]}$ is labeled at most once. The number of non-dummy labels in all trees $(T_\ell^{i,j})_{j \in [k]}$ is upper bounded by $\min\{3t_1, k \cdot 2\ell\}$. The upper bound $3t_1$ is since the trees are constructed from a trace of length at most t_1 and each query can add at most three labels in one tree (most queries add two labels but the first one in a tree adds three since the root is added as a label as well). The upper bound $k \cdot 2\ell$ is since each tree has at most 2ℓ labels. For every query in S'_j (that is not in tr), the probability that the query answer equals one of the these non-dummy labels is at most $\frac{\min\{3t_1, k \cdot 2\ell\}}{2^\sigma}$.

Since $\overline{E_{\text{tree}}}$ (and thus $\overline{E_{\text{tree},1}}$), there are no queries in tr' with an answer that equals a non-dummy label in $T_\ell^{i,j}$. Recall that S'_j has length $d+1$. Taking a union bound over all queries in S'_j , the probability that there exists a query in S'_j (that is not in $\text{tr} \cup \text{tr}'$) with an answer that equals a non-dummy label in $T_\ell^{i,j}$ is at most $(d+1) \cdot \frac{\min\{3t_1, k \cdot 2\ell\}}{2^\sigma}$.

Taking a union bound over all $j \in [k]$,

$$\Pr [E_{\text{sub}} \mid \overline{E_{\text{col}}} \wedge \overline{E_{\text{tree}}}] \leq k \cdot (d+1) \cdot \frac{\min\{3t_1, k \cdot 2\ell\}}{2^\sigma}.$$

5. We show that the event $E \mid \overline{E_{\text{col}}} \wedge \overline{E_{\text{tree}}} \wedge \overline{E_{\text{sub}}}$ cannot happen.

First we argue that for every $i, i' \in [n]$ with $\text{rt}_i = \text{rt}_{i'}$ it holds that $(\mathbf{m}_i, \text{td}_i) = (\mathbf{m}_{i'}, \text{td}_{i'})$. This follows directly from the fact that $\overline{E_{\text{tree},2}}$ implies that $T_\ell^i = T_\ell^{i'}$, and so the extracted message vectors and trapdoor information are the same, namely, $(\mathbf{m}_i, \text{td}_i) = (\mathbf{m}_{i'}, \text{td}_{i'})$.

Next we argue that for every $j \in [k]$ if $\text{MT.Check}^f(\text{rt}_{i_j}, I_{i_j}, \mathbf{a}_{i_j}, \text{pf}_{i_j}) = 1$ then $\mathbf{m}_{i_j}[I_{i_j}] = \mathbf{a}_{i_j}$ and $\text{pf}_{i_j} = \text{pf}_{i_j}$. This follows from applying Claim 18.5.4 for all commitments $\{\text{rt}_i\}_{i \in [n]}$, as we now explain. Fix any $i \in [n]$ and $j \in [k]$, and consider the trace up to the point where the adversary outputs rt_i (i.e., the trace $\text{tr}_1 \parallel \dots \parallel \text{tr}_i$), and the trace of all queries performed after (i.e., the trace $\text{tr}_{i+1} \parallel \dots \parallel \text{tr}_n \parallel \text{tr}'$). Since we assume $\overline{E_{\text{col}}} \wedge \overline{E_{\text{tree}}} \wedge \overline{E_{\text{sub}}}$, both conditions in the probability statement of Claim 18.5.4 hold as well. This implies that if $\text{MT.Check}^f(\text{rt}_{i_j}, I_{i_j}, \mathbf{a}_{i_j}, \text{pf}_{i_j}) = 1$ then $\mathbf{m}_{i_j}[I_{i_j}] = \mathbf{a}_{i_j}$ and $\text{pf}_{i_j} = \text{pf}_{i_j}$ as required.

We conclude that

$$\begin{aligned} & \Pr[E] \\ & \leq \Pr[E_{\text{col}}] + \Pr[E_{\text{tree},1} \mid \overline{E_{\text{col}}}] + \Pr[E_{\text{tree},2} \mid \overline{E_{\text{col}}}] + \Pr[E_{\text{sub}} \mid \overline{E_{\text{col}}} \wedge \overline{E_{\text{tree}}}] \\ & + \Pr[E \mid \overline{E_{\text{col}}} \wedge \overline{E_{\text{tree}}} \wedge \overline{E_{\text{sub}}}] \\ & \leq \frac{1}{2} \cdot \frac{(t_1 - 1) \cdot t_1}{2^\sigma} + \frac{t_2 \cdot \min\{3t_1, k \cdot 2\ell\}}{2^\sigma} + \frac{(t_1 - 1) \cdot t_1}{2^\sigma} + k \cdot (d+1) \cdot \frac{\min\{3t_1, k \cdot 2\ell\}}{2^\sigma} + 0 \\ & \leq \frac{3}{2} \cdot \frac{t_1^2}{2^\sigma} + \frac{(t - t_1 + k \cdot (d+1)) \cdot 3t_1}{2^\sigma}. \end{aligned}$$

The expression above is a concave function in t_1 , and is maximized for a value of $t_1 > t$. Since $t_1 \leq t$, the maximal value is attained when $t_1 = t$. We deduce that

$$\Pr[E] \leq \frac{3}{2} \cdot \frac{t^2}{2^\sigma} + \frac{k \cdot (d+1) \cdot 3t}{2^\sigma}.$$

We conclude that if $6 \cdot k \cdot (d+1) \leq t$ then we can derive the upper bound

$$\Pr[E] \leq \frac{3}{2} \cdot \frac{t^2}{2^\sigma} + \frac{t/6 \cdot 3t}{2^\sigma} = 2 \cdot \frac{t^2}{2^\sigma}.$$

□

Remark 18.5.7 (time complexity of $\text{MT}.\text{MultiExtract}$). A straightforward realization of procedure $\text{MT}.\text{MultiExtract}$ leads to the time complexity claimed in Lemma 18.5.6, because $\text{MT}.\text{Extract}$ is run at most n times.

If $\log t = O(n \cdot \ell)$ (a natural regime) then we can do better by using suitable data structures. The idea is similar to the improved realization of $\text{MT}.\text{Extract}$ described in Remark 18.5.5. The main difference is that $\text{MT}.\text{MultiExtract}$ receives, with each invocation, an additional query-answer trace. We use a dynamic dictionary (instead of a static dictionary), for which inserts and searches take time $\log t \cdot w$ (when there are t entries each consisting of w bits); these are the same costs as for a static dictionary. The total number of inserts into the dictionary is t , and the total number of searches is $O(n \cdot \ell)$. This leads to a time complexity that is $O((t + n \cdot \ell) \cdot \log t \cdot w)$ where $w := \log |\Sigma| + s + \sigma$. This expression improves on the one in Lemma 18.5.6 provided that $\log t = O(n \cdot \ell)$.

Remark 18.5.8 (simultaneous extraction of multiple Merkle commitments). We also consider adversaries that output multiple Merkle commitments in some invocations. In the experiment of Lemma 18.5.6, this corresponds to the case where an adversary makes no queries between some invocations (and so is covered by the lemma). We introduce notation for this special case.

We use the notation

$$((\mathbf{m}_j, \mathbf{td}_j))_{j \in [m]} \leftarrow \text{MT}.\text{MultiExtract}((\mathbf{rt}_j)_{j \in [m]}, \mathbf{tr}_i)$$

as syntactic sugar for these m invocations of $\text{MT}.\text{MultiExtract}$:

- $(\mathbf{m}_1, \mathbf{td}_1) \leftarrow \text{MT}.\text{MultiExtract}(\mathbf{rt}_1, \mathbf{tr})$;
- for $j = 2, \dots, m$, $(\mathbf{m}_j, \mathbf{td}_j) \leftarrow \text{MT}.\text{MultiExtract}(\mathbf{rt}_j, \perp)$.

By construction of $\text{MT}.\text{MultiExtract}$, this amounts to the following operations:

$\text{MT}.\text{MultiExtract}((\mathbf{rt}_j)_{j \in [m]}, \mathbf{tr}_i)$:

1. Append \mathbf{tr}_i to the query-answer trace \mathbf{tr} stored in the internal state.
2. For $j = 1, \dots, m$, compute $(\mathbf{m}_j, \mathbf{td}_j) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}_j, \mathbf{tr})$.
3. Output $((\mathbf{m}_j, \mathbf{td}_j))_{j \in [m]}$.

In particular, the order of extraction among the m Merkle commitments does not matter because each invocation of $\text{MT}.\text{Extract}$ is with respect to the same (internally stored) query-answer trace.

18.5.3 Extractability implies binding

The extractability property in Lemma 18.5.1 is a strong security guarantee: any bounded-query adversary that opens a commitment must have “known” the opening at commitment time. In fact, *extractability is stronger than binding*: we show that if MT satisfies extractability with error κ_{MT} then MT satisfies binding with error $\epsilon_{\text{MT}} \leq 2 \cdot \kappa_{\text{MT}}$ (a closely related error).³

Let A be a t -query algorithm that opens a commitment in two ways with probability ϵ_{MT} :

$$\epsilon_{\text{MT}} := \Pr \left[\begin{array}{l} \exists i \in I_0 \cap I_1 \text{ s.t. } \mathbf{a}_0[i] \neq \mathbf{a}_1[i] \\ \wedge \text{MT}.\text{Check}^f(\mathbf{rt}, I_0, \mathbf{a}_0, \mathbf{pf}_0) = 1 \\ \wedge \text{MT}.\text{Check}^f(\mathbf{rt}, I_1, \mathbf{a}_1, \mathbf{pf}_1) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{rt}, I_0, \mathbf{a}_0, \mathbf{pf}_0, I_1, \mathbf{a}_1, \mathbf{pf}_1) \leftarrow A^f \end{array} \right].$$

Let A_e be the t -query algorithm for the extraction game that is defined as follows.

- A_e^f : Run A^f to get $(\mathbf{rt}, I_0, \mathbf{a}_0, \mathbf{pf}_0, I_1, \mathbf{a}_1, \mathbf{pf}_1)$, set $\mathbf{aux} := (I_0, \mathbf{a}_0, \mathbf{pf}_0, I_1, \mathbf{a}_1, \mathbf{pf}_1)$, and output $(\mathbf{rt}, \mathbf{aux})$.

³This implication loses a factor of 2, and we present it merely for didactic purposes. Indeed, for MT we prove almost the same errors in Lemma 18.4.1 (binding error) and Lemma 18.5.1 (extraction error).

- $A_e^f(\text{aux})$: Sample a random bit b (by using private randomness or via a fresh additional query to the random oracle) and output the tuple $(I_b, \mathbf{a}_b, \text{pf}_b)$ stored in aux .

Fix any candidate extractor MT.Extract . Fix any output $(\text{rt}, I_0, \mathbf{a}_0, \text{pf}_0, I_1, \mathbf{a}_1, \text{pf}_1)$ of A that breaks the binding property (i.e., satisfies the conditions in the probability statement above). If we run MT.Extract on rt (the commitment output by A_e in the first step) and tr (the query-answer trace by A_e in the first step) then we obtain an output (\mathbf{m}, td) that, with probability at least $1/2$, satisfies $\mathbf{m}[I] \neq \mathbf{a}_b \vee \text{pf} \neq \text{pf}_b$, where $\text{pf} := \text{MT.Open}^f(\text{td}, I_b)$. This is because the inputs to the extractor MT.Extract are independent of the bit b (which is sampled by A_e after the inputs to MT.Extract are determined). Moreover, by the definition of A_e , we know that $\text{MT.Check}^f(\text{rt}, I_b, \mathbf{a}_b, \text{pf}_b) = 1$ regardless of the choice of bit b . We deduce that

$$\frac{1}{2} \cdot \epsilon_{\text{MT}} \leq \Pr \left[\begin{array}{l} \text{MT.Check}^f(\text{rt}, I_b, \mathbf{a}_b, \text{pf}_b) = 1 \\ \wedge (\mathbf{m}[I] \neq \mathbf{a}_b \vee \text{pf} \neq \text{pf}_b) \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\text{rt}, \text{aux}) \xleftarrow{\text{tr}} A_e^f \\ (\mathbf{m}, \text{td}) \leftarrow \text{MT.Extract}(\text{rt}, \text{tr}) \\ (I_b, \mathbf{a}_b, \text{pf}_b) \leftarrow A_e^f(\text{aux}) \\ \text{pf} \leftarrow \text{MT.Open}^f(\text{td}, I_b) \end{array} \right].$$

We conclude that any extraction error κ_{MT} that can be proved must satisfy the relation $\frac{1}{2} \cdot \epsilon_{\text{MT}} \leq \kappa_{\text{MT}}$ which gives us the desired bound on the binding error ϵ_{MT} .

18.6 Hiding

The binding and extractability properties of Merkle commitments protect the receiver from malicious senders. Merkle commitments additionally satisfy certain *hiding* properties, which instead protect the sender. Informally, hiding ensures that information about the committed message that the sender did not open is hidden from the receiver. The mechanism that enables this is the random salts used to commit to each message entry at the leaf layer of the tree (see Item 1 in MT.Commit); in fact, this mechanism can be viewed as using the basic commitment scheme CM (from Chapter 17) to commit to each message entry.

We discuss hiding properties of Merkle commitments in two steps. In Section 18.6.1 we explain how a Merkle commitment reveals no information about the committed message. Then in Section 18.6.2 we explain how additionally providing an opening proof reveals no information about the message beyond the opened entries of the message.

18.6.1 Merkle commitments hide the message

We prove that a Merkle commitment rt reveals essentially no information about the underlying committed message \mathbf{m} . Formally this is captured via the simulation paradigm: there exists a polynomial-time probabilistic algorithm MT.SimulateRoot that samples a string, without any information about the message \mathbf{m} , whose distribution is statistically close to rt . This is analogous to the hiding property for the basic commitment scheme CM in Chapter 17 (see Lemma 17.3.1 in Section 17.3). Below we state the hiding property and prove the statistical error bound. Note that the statistical error depends on several parameters: (a) the output size $\sigma \in \mathbb{N}$ of the random oracle; (b) the salt size $s \in \mathbb{N}$; (c) the message size $\ell \in \mathbb{N}$; (d) the number of queries t to the random oracle made by an algorithm trying to distinguish between the two distributions.

Lemma 18.6.1 (rt is hiding). *Let $\text{MT} := \text{MT}[\sigma, \Sigma, \ell, s]$. There exists a polynomial-time probabilistic algorithm $\text{MT}.\text{SimulateRoot}$ such that for every query bound $t \in \mathbb{N}$ and t -query algorithm A , the following two distributions are $z_{\text{rt}}(\sigma, \ell, s, t)$ -close in statistical distance:*

$$\left\{ A^f(\text{aux}, \text{rt}) \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{m} \in \Sigma^\ell, \text{aux}) \leftarrow A^f \\ (\text{rt}, \text{td}) \leftarrow \text{MT}.\text{Commit}^f(\mathbf{m}) \end{array} \right\} \text{ and } \left\{ A^f(\text{aux}, \text{rt}) \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{m} \in \Sigma^\ell, \text{aux}) \leftarrow A^f \\ \text{rt} \leftarrow \text{MT}.\text{SimulateRoot}^f \end{array} \right\}.$$

Above,

$$z_{\text{rt}}(\sigma, \ell, s, t) \leq \begin{cases} \frac{\ell \cdot t}{2^s} + \frac{\ell \cdot t}{2^{2\sigma}} & \text{if } t < \infty \\ \ell \cdot z_{\text{CM}}(\sigma, s, \infty) + \ell \cdot z_{\text{CM}}(\sigma, \sigma, \infty) & \text{if } t = \infty \end{cases}$$

where z_{CM} is the hiding error of $\text{CM} = \text{CM}[\sigma, \ell, s]$ from Lemma 17.3.1.

We sketch a straightforward way to upper bound $z_{\text{rt}}(\sigma, \ell, s, t)$. Each inner vertex in the Merkle tree can be viewed as a basic commitment scheme, as it is obtained by hashing a left child that acts as the message and the right child that acts as the salt. Therefore, going from the leaf layer towards the root, we can use the basic commitment's simulator to replace each commitment with a simulated value. This incurs an error of $z_{\text{CM}}(\sigma, s, t)$ for each leaf, and an error of $z_{\text{CM}}(\sigma, \sigma, t)$ for each inner vertex. This adds up to at most an error of $\ell \cdot z_{\text{CM}}(\sigma, s, t) + \ell \cdot z_{\text{CM}}(\sigma, \sigma, t)$. We essentially follow this approach for the case of $t = \infty$ (and hence the similar bound). However for the case of $t < \infty$ we provide an improved analysis that achieves a better bound (that in particular avoids the multiplicative security loss of ℓ); this is useful for setting parameters in practice.

Proof for $t < \infty$. The adversary A performs queries in two phases: the first phase leads to A choosing the message \mathbf{m} ; and the second phase is after A receives the commitment rt . Let tr_1 be the query-answer trace of the first phase and tr_2 the query-answer trace of the second phase. Define $t_1 := |\text{tr}_1|$ and $t_2 := |\text{tr}_2|$. Note that $t_1 + t_2 \leq t$. The final output of rt is a deterministic function of the query-answer pairs in the traces tr_1 and tr_2 and the commitment rt . In one experiment, rt is computed as $\text{MT}.\text{Commit}^f(\mathbf{m})$ (throughout this analysis, we drop the opening trapdoor td that is also output by $\text{MT}.\text{Commit}$) and in the other experiment it is computed as $\text{MT}.\text{SimulateRoot}^f$. By Claim 1.2.8, the statistical distance between the final output of A in the two experiments is at most the statistical distance between the following two random variables:

$$(\text{tr}_1, \text{MT}.\text{Commit}^f(\mathbf{m}), \text{tr}_2) \text{ and } (\text{tr}_1, \text{MT}.\text{SimulateRoot}^f, \text{tr}_2).$$

The simulator $\text{MT}.\text{SimulateRoot}$ outputs a random string in $\{0, 1\}^\sigma$.

The root $\text{rt} = c_{(0,1)}$ is the result of querying f at $(c_{(1,1)}, c_{(1,2)})$. Let E be the event that $((c_{(1,1)}, c_{(1,2)}), \text{rt})$ appears as a query-answer pair in tr_1 or tr_2 . We argue that if E does not happen, the statistical distance between the two random variables above is zero. Without any other queries, the distributions of $\text{rt} := \text{MT}.\text{Commit}^f(\mathbf{m})$ and $\text{rt} := \text{MT}.\text{SimulateRoot}^f$ are identical (both are uniformly random). In fact, if the adversary does not query f at $(c_{(1,1)}, c_{(1,2)})$ then all query answers are uniform and independent of rt (which is also the case when given the simulated root).

We conclude that the statistical distance between the two random variables is at most $\Pr[E]$. We give an upper bound on $\Pr[E]$ that, while we believe is not tight, suffices for the lemma.

Let S be the trace of queries performed in the execution of $\text{rt} \leftarrow \text{MT}.\text{Commit}^f(\mathbf{m})$, and let E_{any} be the event that there exists a query-answer pair in S that appears in tr_1 or tr_2 . Note

that $\Pr[E] \leq \Pr[E_{\text{any}}]$. We upper bound $\Pr[E_{\text{any}}]$ by separately considering the case of queries in S corresponding to leaf vertices and queries in S corresponding to internal vertices.

- *Leaf vertices.* Let E_d be the event that a leaf query in S appears in tr_1 or tr_2 . We upper bound $\Pr[E_d]$. Fix $i \in [\ell]$. The i -th leaf query has the form (m_i, τ_i) , where τ_i is a random salt string in $\{0, 1\}^s$. The probability that (m_i, τ_i) equals one of the t_1 queries in tr_1 is at most $\frac{t_1}{2^s}$; this addresses the queries made by the adversary before receiving the input rt . After receiving the input rt , the adversary makes further queries, resulting in the query-answer trace tr_2 . The probability that any of the t_2 queries in tr_2 equals (m_i, τ_i) is at most $\frac{t_2}{2^s}$. Taking a union bound over all $i \in [\ell]$,

$$\Pr[E_d] \leq \ell \cdot \left(\frac{t_1}{2^s} + \frac{t_2}{2^s} \right) \leq \frac{\ell \cdot t}{2^s}.$$

- *Internal vertices.* Let $((c_{(j,i)})_{i \in [2^j]})_{j=0}^d$ be the query answers in S . For every layer $j \in \{0, 1, \dots, d-1\}$, let E_j be the event that there exists a vertex index $i \in [2^j]$ such that $((c_{(j+1,2i-1)}, c_{(j+1,2i)}), c_{(j,i)})$ appears as a query-answer pair in tr_1 or tr_2 . We upper bound $\Pr[\vee_{j \in \{0, 1, \dots, d-1\}} E_j \mid \overline{E_d}]$.

For every $j \in \{0, 1, \dots, d-1\}$, we argue that

$$\Pr[E_j = 1 \mid \overline{E_d} \wedge \overline{E_{d-1}} \wedge \dots \wedge \overline{E_{j+1}}] \leq 2^j \cdot \frac{t}{2^{2\sigma}}.$$

First, we consider the trace tr_1 . Conditioning on $\overline{E_d} \wedge \overline{E_{d-1}} \wedge \dots \wedge \overline{E_{j+1}}$, none of the queries in S corresponding to layers $d, d-1, \dots, j+1$ appears in tr_1 , and so the answer to such queries is uniformly random (independent of query answers in tr_1). Hence for every $i \in [2^j]$ the pair $(c_{(j+1,2i-1)}, c_{(j+1,2i)})$ is uniformly random in $\{0, 1\}^{2\sigma}$ (each commitment is σ bits long). The probability that $(c_{(j+1,2i-1)}, c_{(j+1,2i)})$ equals one of the t_1 queries in tr_1 is at most $\frac{t_1}{2^{2\sigma}}$. The probability that such an index $i \in [2^j]$ exists is at most $2^j \cdot \frac{t_1}{2^{2\sigma}}$.

Next, we consider the trace tr_2 . Conditioning on $\overline{E_d} \wedge \overline{E_{d-1}} \wedge \dots \wedge \overline{E_{j+1}}$, none of the queries in S corresponding to layers $d, d-1, \dots, j+1$ appears in tr_2 . Thus, the answers to queries corresponding to layer $j+1$, which are the queries corresponding to layer j , are uniformly random (independent of tr_2). The probability that any of the queries in tr_2 equals any of the queries in S corresponding to layer j is at most $2^j \cdot \frac{t_2}{2^{2\sigma}}$.

We establish that

$$\Pr[E_j = 1 \mid \overline{E_d} \wedge \overline{E_{d-1}} \wedge \dots \wedge \overline{E_{j+1}}] \leq 2^j \cdot \frac{t_1}{2^{2\sigma}} + 2^j \cdot \frac{t_2}{2^{2\sigma}} = 2^j \cdot \frac{t_1 + t_2}{2^{2\sigma}} \leq 2^j \cdot \frac{t}{2^{2\sigma}}.$$

Finally, taking a union bound over every layer $j \in \{0, 1, \dots, d-1\}$,

$$\begin{aligned} \Pr[\vee_{j \in \{0, 1, \dots, d-1\}} E_j \mid \overline{E_d}] &\leq \sum_{j \in \{0, 1, \dots, d-1\}} \Pr[E_j = 1 \mid \overline{E_d} \wedge \overline{E_{d-1}} \wedge \dots \wedge \overline{E_{j+1}}] \\ &\leq \sum_{j \in \{0, 1, \dots, d-1\}} 2^j \cdot \frac{t}{2^{2\sigma}} \\ &= (2^d - 1) \cdot \frac{t}{2^{2\sigma}} \\ &\leq \frac{\ell \cdot t}{2^{2\sigma}}. \end{aligned}$$

The event E_{any} equals the event $\vee_{j \in \{0,1,\dots,d\}} E_j$, so we deduce that

$$\Pr[E_{\text{any}}] = \Pr[\vee_{j \in \{0,1,\dots,d\}} E_j] \leq \Pr[E_d] + \Pr[\vee_{j \in \{0,1,\dots,d-1\}} E_j \mid \overline{E_d}] \leq \frac{\ell \cdot t}{2^s} + \frac{\ell \cdot t}{2^{2\sigma}}.$$

This gives us the upper bound stated in the lemma. Finally, note that if $s \leq 2\sigma$ (a mild condition) then we get the following simplified upper bound:

$$\Pr[E_{\text{any}}] \leq \frac{2\ell t}{2^s}.$$

□

Proof for $t = \infty$. The simulator $\text{MT}.\text{SimulateRoot}$ outputs a random string in $\{0,1\}^\sigma$.

Consider the leaf layer. Each leaf is a basic commitment as defined in Chapter 17. Using Lemma 17.3.1, we can replace a leaf by a uniform random string over $\{0,1\}^\sigma$ and lose an additive term of $z_{\text{CM}}(\sigma, s, \infty)$ in the statistical distance. Thus, replacing all leaves with random string gives us the error of $\ell \cdot z_{\text{CM}}(\sigma, s, \infty)$.

We continue layer by layer up to the root. Each vertex of the tree can be viewed as a basic commitment where the left child is the message, and the right child is considered as the salt (of size σ). Thus, replacing it with a random strings gives us an error of $z_{\text{CM}}(\sigma, \sigma, \infty)$. We replace each vertex, one by one, with a random string, until we reach the root. The number of vertices is at most ℓ and thus we have an error of $\ell \cdot z_{\text{CM}}(\sigma, \sigma, \infty)$.

Overall, the total error accumulated is upper bounded by $\ell \cdot z_{\text{CM}}(\sigma, s, \infty) + \ell \cdot z_{\text{CM}}(\sigma, \sigma, \infty)$. □

Remark 18.6.2. In many cases, it is useful to have the simulator output a uniformly random string, which is the case of our simulator. However, the second term in the error expression for $t = \infty$ can be avoided at the price of changing the simulator to output the root of a Merkle commitment starting from random leaves instead of a uniformly random string. In this case, we only need to replace the leaves with simulated ones and incur only an error of $\ell \cdot z_{\text{CM}}(\sigma, s, \infty)$.

18.6.2 Merkle opening proofs hide the rest of the message

We prove that a Merkle opening proof pf , which is used to authenticate the entries of a message in a subset $I \subseteq [\ell]$ of locations, hides the entries of the message outside of I (i.e., with locations in $[\ell] \setminus I$). Formally, this is captured via the simulation paradigm: there exists a polynomial-time probabilistic algorithm $\text{MT}.\text{Simulate}$ that, given as input the entries of the message in I , samples a string whose distribution is statistically close to pf . For convenience to later sections, the property that we consider below additionally includes the Merkle commitment rt in order to provide a *joint* hiding property across rt and pf . In particular, the property below implies the one above (Lemma 18.6.1) by taking $I = \emptyset$ (as in this case the Merkle opening proof is empty and the simulator receives no entries of the message).

Lemma 18.6.3 (rt and pf are hiding). *Let $\text{MT} := \text{MT}[\sigma, \Sigma, \ell, s]$. There exists a polynomial-time probabilistic algorithm $\text{MT}.\text{Simulate}$ such that for every query bound $t \in \mathbb{N}$, t -query algorithm A , and opening size $q \in \mathbb{N}$, the following two distributions are $z_{\text{MT}}(\sigma, \ell, s, q, t)$ -close in statistical distance:*

$$\left\{ A^f(\text{aux}, \text{rt}, \mathbf{m}[I], \text{pf}) \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{m} \in \Sigma^\ell, I \in \binom{[\ell]}{q}, \text{aux}) \leftarrow A^f \\ (\text{rt}, \text{td}) \leftarrow \text{MT}.\text{Commit}^f(\mathbf{m}) \\ \text{pf} \leftarrow \text{MT}.\text{Open}^f(\text{td}, I) \end{array} \right\}$$

and

$$\left\{ A^f(\text{aux}, \text{rt}, \mathbf{m}[I], \text{pf}) \mid \begin{array}{l} f \leftarrow \mathcal{U}(\sigma) \\ (\mathbf{m} \in \Sigma^\ell, I \in \binom{[\ell]}{q}, \text{aux}) \leftarrow A^f \\ (\text{rt}, \text{pf}) \leftarrow \text{MT.Simulate}^f(I, \mathbf{m}[I]) \end{array} \right\}.$$

Letting z_{rt} be the error defined and upper bounded in Lemma 18.6.1, we define the error function z_{MT} used above as follows:

- if $q = 0$ then $z_{\text{MT}}(\sigma, \ell, s, q, t) := z_{\text{rt}}(\sigma, \ell, s, t)$; and
- if $q > 0$ then $z_{\text{MT}}(\sigma, \ell, s, q, t) := \max_{I \in \binom{[\ell]}{q}} \sum_{(j,i) \in V^*(T_\ell, I)} z_{\text{rt}}(\sigma, 2^{d-j}, s, t)$, where $V^*(T_\ell, I)$ is the set of vertices in the copaths but not the paths for I , i.e.,

$$V^*(T_\ell, I) := \text{copath}(I) \setminus \text{path}(I) = (\cup_{i \in I} \text{copath}(i)) \setminus (\cup_{i \in I} \text{path}(i)).$$

In particular, if $q > 0$ then taking $z_{\text{MT}}(\sigma, \ell, s, q, t) := q \cdot \sum_{j \in [d]} z_{\text{rt}}(\sigma, 2^{d-j}, s, t)$ suffices.⁴

Note that if $q = 0$ then the adversary receives only a Merkle commitment rt ; the adversary receives no message entries or opening proof pf (it is empty). This degenerate case corresponds to the setting of Lemma 18.6.1, so the statistical distance is upper bounded by $z_{\text{rt}}(\sigma, \ell, s, t) = z_{\text{rt}}(\sigma, 2^d, s, t)$. Henceforth we assume that $q > 0$, which is the “interesting” case of Lemma 18.6.3.

First we study the case of simulating (the Merkle commitment and) the opening proof for a single leaf and then study the case of multiple leaves.

Single leaf In the case of a single leaf, the simulator receives a leaf index $i \in [\ell]$ and a message entry $\mathbf{a} \in \Sigma$, and is tasked to sample a Merkle commitment rt and authentication path auth whose distribution is close to that obtained by honestly committing to any message whose i -th entry is \mathbf{a} and then opening that location. Informally, the simulator does this by sampling the salt for the leaf at random (just like MT.Commit does), sampling the commitments along $\text{copath}(i)$ via MT.SimulateRoot (rather than obtaining them from a computation on a message), and computing from these the implied Merkle commitment rt according to the way an authentication path is checked. In more detail, the simulator is constructed as follows.

Construction 18.6.4. The simulator for the case of a single leaf works as follows.

$\text{MT.Simulate}^f(i, \mathbf{a})$:

1. Sample a random salt $\tau_i \in \{0, 1\}^s$.
2. Compute the commitment $c_{(d,i)} := f(\mathbf{a}, \tau_i)$.
3. For $j = d-1, \dots, 1, 0$:
 - a) Sample the commitment $c_{\bar{p}(i,j+1)} \leftarrow \text{MT.SimulateRoot}^f$.
 - b) If $p(i, j+1)$ is an odd vertex then set $c_L := c_{p(i,j+1)}$ and $c_R := c_{\bar{p}(i,j+1)}$.
 - c) If $p(i, j+1)$ is an even vertex then set $c_L := c_{\bar{p}(i,j+1)}$ and $c_R := c_{p(i,j+1)}$.
 - d) Compute the commitment $c_{p(i,j)} := f(c_L, c_R)$.
4. Set $\text{rt} := c_{(0,1)}$.
5. Set $\text{auth} := (\tau_i, (c_{\bar{p}(i,j)})_{j \in \{1, \dots, d\}})$. (For convenience j now ranges from 1 to d .)

⁴Indeed, $\sum_{(j,i) \in \text{copath}(I) \setminus \text{path}(I)} z_{\text{rt}}(\sigma, 2^{d-j}, s, t) \leq \sum_{(j,i) \in \text{copath}(I)} z_{\text{rt}}(\sigma, 2^{d-j}, s, t) \leq \sum_{i \in I} \sum_{j \in [d]} z_{\text{rt}}(\sigma, 2^{d-j}, s, t) = q \cdot \sum_{j \in [d]} z_{\text{rt}}(\sigma, 2^{d-j}, s, t)$. Here we use the fact that every copath contains one vertex per layer (except the root layer).

6. Output (rt, auth) .

The output of this simulator always validates, that is, is such that $\text{MT.Check}^f(\text{rt}, i, \mathbf{a}, \text{auth}) = 1$. The number of queries to the random oracle is $d + 1$. Intuitively, the statistical distance from the real distribution is at most $\sum_{j \in [d]} z_{\text{rt}}(\sigma, 2^{d-j}, s, t)$ because of the following reasons.

- The salt τ_i in auth is distributed as in the real distribution.
- For every $j \in [d]$, the commitment $c_{\bar{p}(i,j)}$ in auth can be viewed as a root of a subtree of size 2^{d-j} , and simulating it via MT.SimulateRoot^f contributes a statistical error of $z_{\text{rt}}(\sigma, 2^{d-j}, s, t)$.
- The Merkle commitment rt is determined by the other values and the random oracle.

Multiple leaves The case of simulating a Merkle commitment and opening proof for multiple leaves I may superficially look rather straightforward: for each leaf index $i \in I$, sample an authentication path for i by following the simulator for a single leaf in Construction 18.6.4.

While intuitive, the above strategy does *not* work. The problem is that the commitments across multiple authentication paths in pf may not be consistent with one another. First, the same vertex may appear in multiple copaths, and thus its commitment is sampled multiple times inconsistently; this problem is straightforwardly addressed by not sampling the commitment for a vertex if it was already sampled.⁵ Second, and this is more subtle, a commitment sampled as part of the copath for a leaf index may be the result of an oracle query to check the copath of another leaf index.⁶ These considerations imply that the simulator should produce the opening proof pf in a more global way: sample all the commitments in pf that do not depend on other commitments, and then derive from these any remaining commitments in pf using the random oracle.

In more detail, an opening proof pf for an index set I contains commitments for vertices in $\text{copath}(I)$ (vertices that belongs to the copath corresponding to some location in I). For every $i \in I$, checking the authentication path corresponding to i involves computing the commitments for vertices in $\text{path}(i)$ from the commitments for vertices in $\text{copath}(i)$ (and the message entry and salt). Hence, the commitments in pf that are derived from other commitments are those in $\text{path}(I)$. We conclude that the commitments in pf that are to be sampled are those corresponding to vertices in $\text{copath}(I) \setminus \text{path}(I)$. These considerations lead to the following simulator.

Construction 18.6.5. Given query access to a random oracle $f \in \mathcal{U}(\sigma)$, the simulator MT.Simulate receives as input a subset I and corresponding entries $(m_i)_{i \in I}$ of a message, and works as follows.

$\text{MT.Simulate}^f(I, (m_i)_{i \in I})$:

1. If $I = \emptyset$, set $\text{rt} \leftarrow \text{MT.SimulateRoot}^f$ and $\text{pf} := \perp$, and output (rt, pf) .
2. For every $i \in I$, sample a random salt $\tau_i \in \{0, 1\}^s$.
3. For every $i \in I$, compute the commitment $c_{(d,i)} := f(m_i, \tau_i)$.
4. For every $(j, i) \in \text{copath}(I) \setminus \text{path}(I)$, set $c_{(j,i)} \leftarrow \text{MT.SimulateRoot}^f$.

⁵For example, for a message of size $\ell = 4$, both $\text{copath}(1)$ and $\text{copath}(2)$ contain the vertex $(1, 2)$.

⁶For example, for a message of size $\ell = 4$, $\text{copath}(1)$ contains the vertex $(1, 2)$, and if the opening proof contains authentication paths for entries 1 and 3, then the commitment of the vertex $(1, 2)$ must be the output of the random oracle applied to the commitment of vertices $(2, 3)$ (derived from entry 3 of the message and its salt) and of $(2, 4)$ (provided in $\text{copath}(3)$).

5. For $j = d, \dots, 1$ (in this order) and $i \in [2^j]$:
 - if $(j, i) \in \text{path}(I)$ then set $c_{(j,i)} := f(c_{(j+1,2i-1)}, c_{(j+1,2i)}) \in \{0, 1\}^\sigma$.
6. Set $\text{rt} := c_{(0,1)}$. (Note that $(0, 1) \in \text{path}(I)$.)
7. For every $i \in I$, set $\text{auth}_i := (\tau_i, (c_v)_{v \in \text{copath}(i)}) = (\tau_i, (c_{\bar{p}(i,j)})_{j \in \{1, \dots, d\}})$.
8. Output (rt, pf) where $\text{pf} := (\text{auth}_i)_{i \in I}$.

Definition 18.6.6. Two vertices v_1, v_2 in the tree T_ℓ are **independent** if the subtree rooted at v_1 is disjoint from the subtree rooted at v_2 .

Claim 18.6.7. The vertices in $\text{copath}(I) \setminus \text{path}(I)$ are pairwise independent.

Proof. Suppose there exist v_1 and v_2 in $\text{copath}(I) \setminus \text{path}(I)$ that are not independent. Since T_ℓ is a tree, one must be a descendant of the other; without loss of generality, v_2 is a descendant of v_1 . Since $v_2 \in \text{copath}(I)$, there exists $i \in I$ such that $v_2 \in \text{copath}(i)$, and hence the sibling v'_2 of v_2 is in $\text{path}(i)$. The vertex v'_2 is also a descendant of v_1 . This tells us that $v_1 \in \text{path}(i)$, a contradiction. \square

Claim 18.6.8. For every $(j, i) \in \text{path}(I)$, $(j + 1, 2i - 1), (j + 1, 2i) \in \text{copath}(I) \cup \text{path}(I)$.

Proof. If (j, i) is on some path from a leaf vertex to the root vertex, one of the two children $(j + 1, 2i - 1)$ and $(j + 1, 2i)$ of (j, i) is on that path and the other is on the corresponding copath. \square

Proof of Lemma 18.6.3. Fix $\mathbf{m} \in \Sigma^\ell$ and $I \in \binom{[\ell]}{q}$. We argue that if the adversary A outputs the message vector \mathbf{m} and index set I then the statistical distance between the two distributions (conditioned on this output) is at most $\sum_{(j,i) \in V^*(T_\ell, I)} z_{\text{rt}}(\sigma, 2^{d-j}, s, t)$ where $V^*(T_\ell, I) = \text{copath}(I) \setminus \text{path}(I)$. The lemma then follows by taking the maximum across all possible outputs \mathbf{m} and I .

The simulator `MT.Simulate` proceeds layer by layer from the leaf vertices towards the root vertex, setting the values of commitments in the opening proof pf either by sampling them via `MT.SimulateRoot` or by deriving them via the random oracle f from other commitments.

First we discuss the leaf layer. For every $i \in I$, the simulator samples a random salt $\tau_i \in \{0, 1\}^s$ and sets the commitment $c_{(d,i)} := f(m_i, \tau_i)$. This is distributed exactly as in the real commitment experiment, so no simulation errors are incurred in the leaf layer.

Next we discuss the commitments $((c_v)_{v \in \text{copath}(i)})_{i \in I} = ((c_{\bar{p}(i,j)})_{j \in \{1, \dots, d\}})_{i \in I}$.

- By Claim 18.6.7, vertices in $\text{copath}(I) \setminus \text{path}(I)$ are pairwise independent. Hence, for every $(j, i) \in \text{copath}(I) \setminus \text{path}(I)$, setting $c_{(j,i)} \leftarrow \text{MT.SimulateRoot}^f$ incurs a simulation error of $z_{\text{rt}}(\sigma, 2^{d-j}, s, t)$. Indeed, (j, i) is the root vertex of a perfect binary tree of depth $d - j$, and this perfect binary tree is disjoint from the perfect binary trees of any other vertex in $\text{copath}(I) \setminus \text{path}(I)$. Hence, the total simulation error here is $\sum_{(j,i) \in V^*(T_\ell, I)} z_{\text{rt}}(\sigma, 2^{d-j}, s, t)$.
- By Claim 18.6.8, for every $(j, i) \in \text{path}(I)$, $(j + 1, 2i - 1), (j + 1, 2i) \in \text{copath}(I) \cup \text{path}(I)$. Hence, the value of the commitment $c_{(j,i)}$ is determined by the values of the commitments $c_{(j+1,2i-1)}$ and $c_{(j+1,2i)}$, which the simulator has either previously sampled or computed. Specifically, for correctness of the simulator's output, the value of the commitment $c_{(j,i)}$ must equal the answer $f(c_{(j+1,2i-1)}, c_{(j+1,2i)})$. No simulation errors are incurred by setting values for these commitments.

Finally, for correctness of the simulator's output, the root commitment $\text{rt} := c_{(0,1)}$ must equal the answer $f(c_{(1,1)}, c_{(1,2)})$. The values for both commitments are set by the simulator because $(0, 1) \in \text{path}(I)$. No simulation error is incurred by setting the root commitment. \square

Remark 18.6.9 (expressions for z_{MT}). The expressions for z_{MT} in Lemma 18.6.3 are in terms of z_{rt} . Here we describe the expression that we obtain for z_{MT} in the case $t < \infty$ when we plug in the expression for z_{rt} in Lemma 18.6.1 (for $t < \infty$).

First we discuss the case of a single leaf. We get the expression $z_{\text{MT}}(\sigma, \ell, s, q = 1, t) = \sum_{j \in [d]} z_{\text{rt}}(\sigma, 2^{d-j}, s, t) \leq (\frac{t}{2^s} + \frac{t}{2^{2\sigma}}) \cdot \sum_{j \in [d]} 2^{d-j} = (\frac{t}{2^s} + \frac{t}{2^{2\sigma}}) \cdot (2^d - 1) \leq \frac{\ell \cdot t}{2^s} + \frac{\ell \cdot t}{2^{2\sigma}}$.

Next we discuss the case of multiple leaves, using the simplified upper bound at the end of Lemma 18.6.3. We get the expression $q \cdot \sum_{j \in [d]} z_{\text{rt}}(\sigma, 2^{d-j}, s, t) \leq q \cdot (\frac{t}{2^s} + \frac{t}{2^{2\sigma}}) \cdot \sum_{j \in [d]} 2^{d-j} = q \cdot (\frac{t}{2^s} + \frac{t}{2^{2\sigma}}) \cdot (2^d - 1) \leq \frac{q \cdot \ell \cdot t}{2^s} + \frac{q \cdot \ell \cdot t}{2^{2\sigma}}$.

Remark 18.6.10 (superadditive and monotonic z_{MT}). For convenience we assume that the error bound $z_{\text{MT}}(\sigma, \ell, s, q, t)$ is *superadditive* in the message length ℓ and opening size q , which means that

$$z_{\text{MT}}(\sigma, \ell_1, s, q_1, t) + z_{\text{MT}}(\sigma, \ell_2, s, q_2, t) \leq z_{\text{MT}}(\sigma, \ell_1 + \ell_2, s, q_1 + q_2, t).$$

In particular, z_{MT} is *monotonic* (more precisely, nondecreasing) in the message length and opening size. This holds for the upper bound $z_{\text{MT}}(\sigma, \ell, s, q, t) = q \cdot \sum_{j \in [d]} z_{\text{rt}}(\sigma, 2^{d-j}, s, t)$ in Lemma 18.6.3.

We use this to simplify certain expressions later on (in Sections 22.2 and 26.2). However, tighter upper bounds on z_{MT} might not be monotonic (and thus also not superadditive) in the message length ℓ or opening size q . If so one can consider the non-simplified expressions in our analyses.

For example, in the upper bound $\max_{I \in \binom{[\ell]}{q}} \sum_{(j,i) \in V^*(T_\ell, I)} z_{\text{rt}}(\sigma, 2^{d-j}, s, t)$, the size of the set $V^*(T_\ell, I)$ does *not* monotonically increase in the size q of I (informally, $V^*(T_\ell, I)$ first gets bigger and then gets smaller). In the extreme setting where $I = [\ell]$ (i.e., the size of I is the largest possible value $q = \ell$) the set $V^*(T_\ell, I)$ is empty, and the hiding error is zero ($z_{\text{MT}}(\sigma, \ell, s, q = \ell, t) = 0$).

Part V

SNARGs Based on PCPs

Overview

The non-interactive arguments constructed so far (in Parts II and III) are not succinct: the size of the argument string is at least the prover-to-verifier communication complexity of the underlying interactive proof (or sigma protocol). Indeed, we used the hash function (the random oracle) only to remove interaction, without any reduction in the prover-to-verifier communication complexity.

In this part we study how to construct (interactive and non-interactive) arguments that are succinct. We introduce the notion of probabilistically checkable proofs (PCPs), a type of probabilistic proof where the verifier reads a few locations of a long proof string. Then we explain how to use commitment schemes from Part IV (specifically, Merkle commitment schemes) to compile any PCP into a succinct argument. We consider first the case of a succinct interactive argument, and then build on this to additionally achieve a succinct non-interactive argument (a SNARG).

19 Probabilistically checkable proofs	199
19.1 Definition	199
19.2 State restoration	202
20 Warmup: succinct interactive arguments from PCPs	207
20.1 Construction	207
20.2 Non-adaptive soundness	209
20.3 Adaptive soundness	212
20.4 Additional security definitions	214
21 The Micali transformation	216
21.1 Construction	216
21.2 Soundness	218
22 Additional security definitions	225
22.1 Knowledge soundness	225
22.2 Zero knowledge	226

19 Probabilistically checkable proofs

We introduce notations and definitions for *probabilistically checkable proofs* (PCPs), which are proof strings that can be checked by a verifier that randomly reads a few locations of the proof string.

In subsequent chapters we study how to construct succinct arguments from PCPs, via transformations that are arguably among the simplest methods to construct succinct arguments. In Chapter 20 we study the *Kilian transformation*, which compiles any given PCP into a succinct *interactive* argument. Then in Chapters 21 and 22 we study the *Micali transformation*, which compiles any given PCP into a succinct *non-interactive* argument.

19.1 Definition

A probabilistically checkable proof (PCP) is a tuple of algorithms

$$\text{PCP} = (\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}})$$

that works as follows. The PCP prover \mathbf{P}_{PCP} receives as input an instance \mathbf{x} and a witness \mathbf{w} , and outputs a PCP string Π (possibly using some randomness); the PCP string is a list of symbols, each taken from a finite-size alphabet. The PCP verifier \mathbf{V}_{PCP} receives as input the instance \mathbf{x} and is granted query access to the PCP string Π . The PCP verifier \mathbf{V}_{PCP} probabilistically queries locations of Π , with each query answered with the corresponding symbol at that location; then \mathbf{V}_{PCP} outputs a bit denoting whether to accept or reject. See Figure 19.1 for a diagram of a PCP.

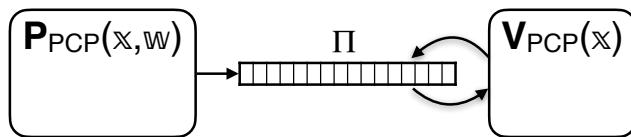


Figure 19.1: Diagram of a PCP.

The tuple $\text{PCP} = (\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}})$ is a PCP for a relation \mathcal{R} with (*perfect*) *completeness* and *soundness error* ϵ_{PCP} if it satisfies the two properties stated below.

Definition 19.1.1. $\text{PCP} = (\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}})$ for a relation \mathcal{R} has **perfect completeness** if for every $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$,

$$\Pr [\mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}) = 1 \mid \Pi \leftarrow \mathbf{P}_{\text{PCP}}(\mathbf{x}, \mathbf{w})] = 1.$$

In other words, for every true statement, the probability that the (honest) PCP prover produces a PCP string that convinces the (honest) PCP verifier is 1. The probability is taken over the randomness of the PCP verifier, as well as any randomness used by the PCP prover. (A PCP prover may use randomness, e.g., to achieve zero-knowledge, a property discussed further below.)

Definition 19.1.2. $\text{PCP} = (\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}})$ for a relation \mathcal{R} has **soundness error** ϵ_{PCP} if for every $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ and (unbounded) malicious PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}$,

$$\Pr [\mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}) = 1 \mid \Pi \leftarrow \tilde{\mathbf{P}}_{\text{PCP}}] \leq \epsilon_{\text{PCP}}(\mathbf{x}).$$

In other words, for every false statement, the probability that a computationally unbounded malicious PCP prover outputs a PCP string that convinces the (honest) PCP verifier is at most ϵ_{PCP} . The probability is taken over the randomness of the PCP verifier, as well as any randomness of the malicious PCP prover. The malicious PCP prover is computationally unbounded, so randomness does not help (the PCP prover can have the best choice of randomness hardcoded). Nevertheless we provide the more general definition for convenience in later discussions.

The PCP soundness error ϵ_{PCP} depends on the instance \mathbf{x} given as input to the PCP verifier. For convenience, we additionally define notation for the largest soundness error across any instance \mathbf{x} (not in the language) whose size is at most a bound n :

$$\epsilon_{\text{PCP}}(n) := \max \{ \epsilon_{\text{PCP}}(\mathbf{x}) \mid \mathbf{x} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \text{ and } \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \}.$$

In particular, we can use $\epsilon_{\text{PCP}}(n)$ to upper bound the winning probability of a malicious PCP prover that chooses the instance (as long as the instance is not in the language). That is, Definition 19.1.2 implies that for every instance size bound $n \in \mathbb{N}$ and (unbounded) malicious PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}$,

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}) = 1 \end{array} \mid (\mathbf{x}, \Pi) \leftarrow \tilde{\mathbf{P}}_{\text{PCP}} \right] \leq \epsilon_{\text{PCP}}(n).$$

Efficiency measures We are interested in several efficiency measures of a PCP.

- *Alphabet*, denoted Σ , is the alphabet used to write the PCP string.
- *Proof length*, denoted l , is the number of symbols in the PCP string. The total number of bits in a PCP string is $l \cdot \log |\Sigma|$.
- *Query complexity*, denoted q , is the number of locations in the PCP string read by the PCP verifier. Each query returns a symbol of $\log |\Sigma|$ bits.
- *Randomness complexity*, denoted r , is the number of random bits used by the PCP verifier.
- *Prover time and verifier time*, denoted pt and vt , are the time complexities of the PCP prover (to output the PCP string) and PCP verifier (to output its decision bit), respectively.

The above efficiency measures may be functions of the instance \mathbf{x} (and possibly other parameters associated to the construction of a PCP).

Knowledge soundness The soundness notion can be strengthened to a knowledge soundness notion, which informally means that whenever the PCP verifier accepts a PCP string then, up to some error, an efficient deterministic extractor outputs a witness when given the instance and the PCP string. (Note that the extractor does not receive as input the randomness of the PCP verifier.)

Definition 19.1.3. $\text{PCP} = (\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}})$ for a relation \mathcal{R} has **knowledge soundness error** κ_{PCP} if there exists a polynomial-time deterministic algorithm \mathbf{E}_{PCP} such that for every instance \mathbf{x} and malicious prover $\tilde{\mathbf{P}}_{\text{PCP}}$ the following holds:

$$\Pr \left[\begin{array}{l} \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}) = 1 \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \end{array} \mid \begin{array}{l} \Pi \leftarrow \tilde{\mathbf{P}}_{\text{PCP}} \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{PCP}}(\mathbf{x}, \Pi) \end{array} \right] \leq \kappa_{\text{PCP}}(\mathbf{x}).$$

We additionally define $\kappa_{\text{PCP}}(n) := \max \{ \kappa_{\text{PCP}}(\mathbf{x}) \mid \mathbf{x} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \}$.

If $\text{PCP} = (\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}})$ has knowledge soundness error κ_{PCP} then it has soundness error $\epsilon_{\text{PCP}} \leq \kappa_{\text{PCP}}$: if $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ then the extractor cannot output a witness \mathbf{w} such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, in which case the event bounded by the knowledge soundness condition equals the event bounded by the soundness condition.

Zero knowledge A notion of zero knowledge for PCPs against honest PCP verifiers suffices for the applications that we study. Informally, zero knowledge states that the *verifier's view* can be efficiently simulated by a probabilistic algorithm that receives as input the instance (but not a corresponding witness). The view consists of the instance, the randomness, and the query answers from the given PCP string. Honest-verifier zero knowledge then requires that the simulated view is statistically close to a view in a real execution, provided that the proved statement is true.

Definition 19.1.4. The PCP verifier's **view** in $\text{PCP} = (\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}})$ on the instance-witness pair (\mathbf{x}, \mathbf{w}) , denoted $\text{View}_{\text{PCP}}(\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}}, \mathbf{x}, \mathbf{w})$, is the random variable $(\mathbf{x}, \rho, Q, \mathbf{a})$ where:

- $\rho \in \{0, 1\}^r$ is a random choice of randomness for the PCP verifier \mathbf{V}_{PCP} ;
- $Q \subseteq [l]$ and $\mathbf{a} \in \Sigma^Q$ are the queries and answers when running $\mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho)$ for $\Pi \leftarrow \mathbf{P}_{\text{PCP}}(\mathbf{x}, \mathbf{w})$.

The PCP prover $\mathbf{P}_{\text{PCP}}(\mathbf{x}, \mathbf{w})$ may use private randomness (not part of the PCP verifier's view).

Definition 19.1.5. $\text{PCP} = (\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}})$ for a relation \mathcal{R} has **honest-verifier zero-knowledge error** z_{PCP} if there exists a polynomial-time probabilistic algorithm \mathbf{S}_{PCP} such that for every instance-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ the following random variables are $z_{\text{PCP}}(\mathbf{x})$ -close in statistical distance:

$$\text{View}_{\text{PCP}}(\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}}, \mathbf{x}, \mathbf{w}) \text{ and } \mathbf{S}_{\text{PCP}}(\mathbf{x}).$$

We additionally define $z_{\text{PCP}}(n) := \max \{ z_{\text{PCP}}(\mathbf{x}) \mid \mathbf{x}, \mathbf{w} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \text{ and } (\mathbf{x}, \mathbf{w}) \in \mathcal{R} \}$.

Verification on local views We sometimes run a PCP verifier with query access to a *partial* PCP string defined on a subset of locations, and it is useful to introduce notation for this setting.

Definition 19.1.6. Let $Q \subseteq [l]$ be a query set, $\mathbf{a} \in \Sigma^Q$ query answers, \mathbf{x} an instance, and $\rho \in \{0, 1\}^r$ a choice of PCP randomness. We denote by

$$\mathbf{V}_{\text{PCP}}^{[Q, \mathbf{a}]}(\mathbf{x}, \rho)$$

the decision bit of the PCP verifier \mathbf{V}_{PCP} , given instance \mathbf{x} and PCP randomness ρ , when each query $j \in Q$ is answered with $\mathbf{a}[j] \in \Sigma$. (If \mathbf{V}_{PCP} queries outside the set Q then $\mathbf{V}_{\text{PCP}}^{[Q, \mathbf{a}]}(\mathbf{x}, \rho) = 0$.)

19.2 State restoration

We introduce the *PCP state restoration game*, where a PCP prover attacks a PCP multiple times. Then we discuss notions of soundness and knowledge soundness associated to this game. These are stronger notions of soundness and knowledge soundness that we use, in later sections, to describe the security properties of the Micali transformation (which we study in Chapter 21).

These notions are analogous to the notions of state-restoration soundness and knowledge soundness for SPs discussed in Chapter 12. Similarly to the case of SPs, good standard soundness implies good state-restoration soundness; specifically, the state-restoration soundness error of a PCP is $\Theta(t)$ times the standard soundness error of the PCP. Similar considerations hold for state-restoration knowledge soundness compared to standard knowledge soundness.

In this chapter we use state-restoration soundness (and knowledge soundness) as a convenient intermediate step towards the security of the non-interactive argument obtained by applying the Micali transformation to a given PCP, as discussed in Chapter 21.

SR game We consider a game where a PCP prover attacks a PCP multiple times. A move consists of proposing an instance \mathbf{x} and a PCP string Π , and the game responds with a fresh choice of PCP randomness ρ . The PCP prover can propose the same instance and PCP string more times (to see multiple fresh choices of PCP randomness), but up to some limit (as determined by a salt size). The goal of the PCP prover is to output an instance \mathbf{x} and a PCP string Π such that $\mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1$ where ρ is the PCP randomness chosen for that move. (The PCP prover can also output an instance and PCP string that was not one of its moves.)

We refer to this as a *PCP state-restoration game*, because the adversary is “attacking” the PCP by trying to sample multiple PCP verifier states.

The salt size $s \in \mathbb{N}$ in the definition below implies that a PCP prover can see at most 2^s fresh choices of PCP randomness for the same instance and PCP string. The game’s randomness, used to consistently answer the PCP prover’s moves, is captured by the random function $\text{rnd} \in \mathcal{U}(r)$.

Definition 19.2.1. *The PCP state-restoration game for $\text{PCP} = (\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}})$ with salt size $s \in \mathbb{N}$, function $\text{rnd} \in \mathcal{U}(r)$, and PCP state-restoration prover $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ is defined below.*

$\text{Game}_{\text{PCP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}})$:

1. Repeat the following until $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ decides to exit the loop.
 - a) $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ outputs $(\mathbf{x}, \Pi, \sigma)$, where \mathbf{x} is an instance, $\Pi \in \Sigma^l$ is a PCP string, and $\sigma \in \{0, 1\}^s$ is a salt string.
 - b) Set $\rho := \text{rnd}(\mathbf{x}, \Pi, \sigma)$.
 - c) Send ρ to $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$.
2. $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ outputs $(\mathbf{x}, \Pi, \sigma)$, where \mathbf{x} is an instance, $\Pi \in \Sigma^l$ is a PCP string, and $\sigma \in \{0, 1\}^s$ is a salt string.
3. Set $\rho := \text{rnd}(\mathbf{x}, \Pi, \sigma)$.
4. Output $(\mathbf{x}, \Pi, \sigma, \rho)$.

We denote by tr^{sr} the list of move-response pairs of the form $((\mathbf{x}, \Pi, \sigma), \rho)$ performed in the loop. We show tr^{sr} in an execution of the PCP state-restoration game using the following notation:

$$(\mathbf{x}, \Pi, \sigma, \rho) \xleftarrow{\text{tr}^{\text{sr}}} \text{Game}_{\text{PCP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}).$$

We say that $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ is **t-move** if $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ exits the loop after at most t iterations.

SR soundness State-restoration soundness captures the maximum probability that a prover playing the PCP state-restoration game finds an instance and PCP string such that the instance is not in the language and the PCP string convinces the PCP verifier.

Definition 19.2.2. $\text{PCP} = (\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}})$ has state-restoration soundness error $\epsilon_{\text{PCP}}^{\text{sr}}$ if for every salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move malicious PCP state-restoration prover $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$, and instance size bound $n \in \mathbb{N}$:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} \mathsf{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \Pi, \sigma, \rho) \leftarrow \text{Game}_{\text{PCP}}^{\text{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \end{array} \right] \leq \epsilon_{\text{PCP}}^{\text{sr}}(s, t, n).$$

The PCP state-restoration game gives the cheating prover $t + 1$ attempts at convincing the PCP verifier (once per loop and once after all loops), each time possibly with a different instance and proof. Intuitively, the maximum winning probability of any cheating prover, for instances not in the language, is at most $t + 1$ times the PCP soundness error. We prove this next.

Theorem 19.2.3. Let $\text{PCP} = (\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}})$ be a PCP. If PCP has soundness error ϵ_{PCP} then PCP has state-restoration soundness error $\epsilon_{\text{PCP}}^{\text{sr}}$ such that

$$\epsilon_{\text{PCP}}^{\text{sr}}(s, t, n) \leq (t + 1) \cdot \epsilon_{\text{PCP}}(n).$$

Note that the salt size s does not affect the upper bound.

In more detail, there exists a PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}$ such that for every salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move malicious PCP state-restoration prover $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$, and instance size bound $n \in \mathbb{N}$:

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} \mathsf{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \Pi, \sigma, \rho) \leftarrow \text{Game}_{\text{PCP}}^{\text{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \end{array} \right] \\ & \leq (t + 1) \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \Pi) \leftarrow \tilde{\mathbf{P}}_{\text{PCP}}(s, t, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \\ \rho \leftarrow \{0, 1\}^r \end{array} \right]. \end{aligned}$$

Construction 19.2.4. The PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}$ receives as input a salt size $s \in \mathbb{N}$ and move budget $t \in \mathbb{N}$ and black-box access to a PCP state-restoration prover $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$, and works as follows.

1. Lazily sample an oracle $\mathsf{rnd} \leftarrow \mathcal{U}(\mathbf{r})$.
2. Sample a random index $j \in [t + 1]$.
3. Emulate the execution of $\text{Game}_{\text{PCP}}^{\text{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}})$ up to the j -th output $(\mathbf{x}, \Pi, \sigma)$ of $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$. (There are at most t outputs in the loop in Item 1, and an output in Item 2.) Note that black-box access to $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ suffices for this.
4. Output (\mathbf{x}, Π) .

Proof. The PCP state-restoration prover $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ produces at most $t + 1$ outputs: one output per iteration of the loop (there are at most t iterations), and a final output after the loop. The final output either equals an earlier output or not. We denote by I the set of indices of any appearance of the final output: I contains the index of any iteration in which the final output appears, or (if the final output first appears after the loop) I is a singleton containing the

number of iterations plus one. Note that I is a random variable over $\{1, \dots, t+1\}$ that depends on rnd . Whenever $j \in I$, the j -th output of $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ equals the final output of $\mathbf{P}_{\text{PCP}}^{\text{sr}}$, so in this case $\tilde{\mathbf{P}}_{\text{PCP}}$ and $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ output the same instance \mathbf{x} and PCP string Π . Moreover, the PCP verifier randomness ρ is sampled from $\{0, 1\}^r$, the same distribution as $\rho := \text{rnd}(\mathbf{x}, \Pi, \sigma)$ for a random choice of $\text{rnd} \in \mathcal{U}(r)$. Hence we can write:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \Pi) \leftarrow \tilde{\mathbf{P}}_{\text{PCP}}(s, t, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \\ \rho \leftarrow \{0, 1\}^r \end{array} \right] \\ \geq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \\ \wedge j \in I \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \Pi, \sigma, \rho) \leftarrow \text{Game}_{\text{PCP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \\ j \leftarrow [t+1] \end{array} \right]. \right]$$

Since j and I are independent, $\Pr[j \in I] \geq \frac{1}{t+1}$. This lets us lower bound the previous probability:

$$\geq \frac{1}{t+1} \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \Pi, \sigma, \rho) \leftarrow \text{Game}_{\text{PCP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \end{array} \right].$$

We conclude that:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \Pi, \sigma, \rho) \leftarrow \text{Game}_{\text{PCP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \end{array} \right] \\ \leq (t+1) \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \Pi) \leftarrow \tilde{\mathbf{P}}_{\text{PCP}}(s, t, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \\ \rho \leftarrow \{0, 1\}^r \end{array} \right] \\ \leq (t+1) \cdot \epsilon_{\text{PCP}}(n). \right]$$

Salts play a meaningful role in the PCP state-restoration game (in that they lead to a more general class of games) but do not affect the above argument. \square

SR knowledge soundness State-restoration knowledge soundness captures the maximum probability that a prover playing the PCP state-restoration game finds an instance and PCP string such that the given extractor cannot find a witness for the instance and the PCP string convinces the PCP verifier.

Definition 19.2.5. $\text{PCP} = (\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}})$ has **state-restoration knowledge soundness error** $\kappa_{\text{PCP}}^{\text{sr}}$ if there exists a polynomial-time deterministic algorithm $\mathbf{E}_{\text{PCP}}^{\text{sr}}$ (the extractor) such that for every salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move deterministic PCP state-restoration prover $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$, and instance size bound n :

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \Pi, \sigma, \rho) \leftarrow \text{Game}_{\text{PCP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{PCP}}^{\text{sr}}(\mathbf{x}, \Pi) \end{array} \right] \leq \kappa_{\text{PCP}}^{\text{sr}}(s, t, n).$$

Similarly to Theorem 19.2.3, $(t + 1)$ times the knowledge soundness error of a PCP is an upper bound on its state-restoration knowledge soundness error.

Theorem 19.2.6. *Let $\text{PCP} = (\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}})$ be a PCP. If PCP has knowledge soundness error κ_{PCP} then PCP has state-restoration knowledge soundness error $\kappa_{\text{PCP}}^{\text{sr}}$ such that*

$$\kappa_{\text{PCP}}^{\text{sr}}(s, t, n) \leq (t + 1) \cdot \kappa_{\text{PCP}}(n).$$

Note that the salt size s does not affect the upper bound.

In more detail, letting \mathbf{E}_{PCP} be the knowledge extractor for PCP , there exists a PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}$ and a PCP state-restoration extractor $\mathbf{E}_{\text{PCP}}^{\text{sr}}$ such that for every salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move malicious PCP state-restoration prover $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$, and instance size bound $n \in \mathbb{N}$:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \Pi, \sigma, \rho) \leftarrow \text{Game}_{\text{PCP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{PCP}}^{\text{sr}}(\mathbf{x}, \Pi) \end{array} \right] \\ \leq (t + 1) \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \Pi) \leftarrow \tilde{\mathbf{P}}_{\text{PCP}}(s, t, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \\ \rho \leftarrow \{0, 1\}^r \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{PCP}}(\mathbf{x}, \Pi) \end{array} \right]. \right]$$

Proof. The proof is similar to the soundness proof in Theorem 19.2.3. Let $\tilde{\mathbf{P}}_{\text{PCP}}$ be the PCP prover in Construction 19.2.4. The extractor $\mathbf{E}_{\text{PCP}}^{\text{sr}}$ applies the extractor \mathbf{E}_{PCP} of the underlying PCP: $\mathbf{E}_{\text{PCP}}^{\text{sr}}(\mathbf{x}, \Pi) := \mathbf{E}_{\text{PCP}}(\mathbf{x}, \Pi)$. We are left to argue the success probability of this extractor.

The PCP state-restoration prover $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ produces at most $t + 1$ outputs: one output per iteration of the loop (there are at most t iterations), and a final output after the loop. The final output either equals an earlier output or not. We denote by I the set of indices of any appearance of the final output: I contains the index of any iteration in which the final output appears, or (if the final output first appears after the loop) I is a singleton containing the number of iterations plus one. Note that I is a random variable over $\{1, \dots, t + 1\}$ that depends on \mathbf{rnd} . Whenever $j \in I$, the j -th output of $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ equals the final output of $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$, so in this case $\tilde{\mathbf{P}}_{\text{PCP}}$ and $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ output the same instance \mathbf{x} and PCP string Π . Moreover, the PCP verifier randomness ρ is sampled from $\{0, 1\}^r$, the same distribution as $\rho := \mathbf{rnd}(\mathbf{x}, \Pi, \sigma)$ for a random choice of $\mathbf{rnd} \in \mathcal{U}(\mathbf{r})$. Hence we can write:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \Pi) \leftarrow \tilde{\mathbf{P}}_{\text{PCP}}(s, t, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \\ \rho \leftarrow \{0, 1\}^r \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{PCP}}(\mathbf{x}, \Pi) \end{array} \right] \\ \geq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \\ \wedge j \in I \end{array} \middle| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \Pi, \sigma, \rho) \leftarrow \text{Game}_{\text{PCP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{PCP}}^{\text{sr}}(\mathbf{x}, \Pi) \\ j \leftarrow [t + 1] \end{array} \right]. \right]$$

Since j and I are independent, $\Pr[j \in I] \geq \frac{1}{t+1}$. This lets us lower bound the previous probability:

$$\geq \frac{1}{t+1} \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \Pi, \sigma, \rho) \leftarrow \text{Game}_{\text{PCP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{PCP}}^{\text{sr}}(\mathbf{x}, \Pi) \end{array} \right].$$

We conclude that:

$$\begin{aligned}
 & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \Pi, \sigma, \rho) \leftarrow \text{Game}_{\text{PCP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{PCP}}^{\text{sr}}(\mathbf{x}, \Pi) \end{array} \right] \\
 & \leq (t+1) \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \Pi) \leftarrow \tilde{\mathbf{P}}_{\text{PCP}}(s, t, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}) \\ \rho \leftarrow \{0, 1\}^r \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{PCP}}(\mathbf{x}, \Pi) \end{array} \right] \\
 & \leq (t+1) \cdot \kappa_{\text{PCP}}(n).
 \end{aligned}$$

□

20 Warmup: succinct interactive arguments from PCPs

We describe how to construct a succinct *interactive* argument, starting from any probabilistically checkable proof (PCP). This is known as the *Kilian transformation* [Kil92]. Later, in Chapter 21, we build on this to construct a succinct *non-interactive* argument (again starting from any PCP). Throughout, we assume familiarity with the Merkle commitment scheme (which we study in Chapter 18).

20.1 Construction

The PCP model envisages a setting where the PCP verifier has query access to a proof string. The succinct interactive argument that we describe here relies on the Merkle commitment scheme to “implement” this query access interface, as we now elaborate.

The argument verifier wishes to query the PCP string but can neither receive it (the PCP string is too long) nor trust the argument prover to answer queries (the argument prover could choose how to answer based on the received queries). The Kilian transformation resolves this via a commit-challenge-response structure that is a common paradigm in cryptographic protocols (e.g., many SPs follow this structure); in particular, the transformation relies on the Merkle commitment scheme, which provides succinct commitments with local openings (in the random oracle model).

1. *Commit* ($\mathcal{P} \rightarrow \mathcal{V}$). The argument prover sends to the argument verifier a short Merkle commitment to the PCP string.
2. *Challenge* ($\mathcal{V} \rightarrow \mathcal{P}$). The argument verifier sends PCP randomness to the argument prover, which determines a set of queries.
3. *Response* ($\mathcal{P} \rightarrow \mathcal{V}$). The argument prover opens the queried locations of the PCP string.

The first message ensures that the argument prover is (computationally) bound to a particular PCP string via the Merkle commitment, and so the argument verifier does not have to worry about sending the PCP randomness to the argument prover in the second message. The third (and final) message authenticates the claimed answers relative to the Merkle commitment. This sketch is merely an intuition, and we will see that establishing a formal security proof requires some work.

Below we describe the construction in detail.

Construction 20.1.1: Kilian transformation

Let $\text{PCP} = (\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}})$ be a PCP with proof length l over alphabet Σ , query complexity q , and randomness complexity r . Let $\lambda \in \mathbb{N}$ be a security parameter and $s \in \mathbb{N}$ be a privacy parameter. Define $\text{MT} := \text{MT}[\lambda, \Sigma, l, s]$ to be the Merkle commitment scheme with the stated parameters. (The output size of the random oracle for the Merkle commitment scheme equals the security parameter λ .)

We define $\text{IARG} := \text{Kilian}[\text{PCP}, \lambda, s]$ to be the 3-message public-coin interactive argument $\text{IARG} = (\mathcal{P}, \mathcal{V})$ constructed as follows. The argument prover \mathcal{P} receives as input an instance \mathbf{x} and witness w , and the argument verifier \mathcal{V} receives as input the instance \mathbf{x} . Both receive query access to a random oracle $f \in \mathcal{U}(\lambda)$ and they interact as below.

1. *Commit.* The argument prover \mathcal{P} computes its first message as follows.
 - a) Compute the PCP string $\Pi := \mathbf{P}_{\text{PCP}}(\mathbf{x}, w) \in \Sigma^l$.
 - b) Commit to Π via the Merkle commitment scheme: $(\text{rt}, \text{td}) := \text{MT}.\text{Commit}^f(\Pi)$.
 - c) Send the Merkle commitment $\text{rt} \in \{0, 1\}^\lambda$.
2. *Challenge.* The argument verifier \mathcal{V} sends randomness $\rho \in \{0, 1\}^r$ for the PCP verifier \mathbf{V}_{PCP} .
3. *Response.* The argument prover \mathcal{P} computes its second message as follows.
 - a) Simulate the PCP verifier $\mathbf{V}_{\text{PCP}}^\Pi(\mathbf{x}, \rho)$, yielding a query set $Q \subseteq [l]$ with $|Q| \leq q$.
 - b) Set $\mathbf{a} := \Pi[Q] \in \Sigma^Q$ to be the answers for the query set.
 - c) Compute an opening proof for \mathbf{a} : $\text{pf} := \text{MT}.\text{Open}^f(\text{td}, Q)$.
 - d) Send $(Q, \mathbf{a}, \text{pf})$.
4. *Verification.* The argument verifier \mathcal{V} checks the following.
 - a) Check that $\mathbf{V}_{\text{PCP}}^{[Q, \mathbf{a}]}(\mathbf{x}, \rho) = 1$. (The PCP verifier $\mathbf{V}_{\text{PCP}}(\mathbf{x}, \rho)$ accepts if each query $j \in Q$ is answered with $\mathbf{a}[j] \in \Sigma$. If any query falls outside the set Q then reject.)
 - b) Check that $\text{MT}.\text{Check}^f(\text{rt}, Q, \mathbf{a}, \text{pf}) = 1$. (The query answers $\mathbf{a} \in \Sigma^Q$ are authenticated with respect to the Merkle commitment rt .)

Efficiency We discuss efficiency properties of the above construction.

- *Communication complexity.* The argument prover sends to the argument verifier:
 - one Merkle commitment rt , consisting of λ bits;
 - q query-answer pairs, consisting of $q \cdot (\log l + \log |\Sigma|)$ bits; and
 - a Merkle opening proof for q locations, consisting of at most $q \cdot (s + \lambda \cdot \log l)$ bits.¹

Hence the number of bits sent by the argument prover is at most:

$$\lambda + q \cdot (\log l + \log |\Sigma| + s + \lambda \cdot \log l) = O(q \cdot (\log |\Sigma| + s + \lambda \cdot \log l)) . \quad (20.1)$$

¹See Section 29.2 for more on the size of Merkle opening proofs.

In the other direction, the argument verifier sends to the argument prover the randomness ρ , which consists of r bits. If zero knowledge is not a goal, the privacy parameter s equals 0.

Simply sending the (uncommitted) PCP string Π would have cost $|l| \cdot \log |\Sigma|$ bits. Therefore, using the Merkle commitment scheme to succinctly commit to the PCP string and then locally open the queried locations reduces the communication complexity to $q \cdot (\log l + \log |\Sigma|)$ plus the information that is used to commit and authenticate. For most parameter settings, this is significantly smaller than the number of bits in the PCP string, achieving the desired “succinct communication complexity”.

- *Prover complexity.* The complexity of the argument prover is typically dominated by the complexity of the underlying PCP prover. Moreover, the number of queries to the random oracle is $q_p = O(l)$ (to commit to the PCP string).
- *Verifier complexity.* The complexity of the argument verifier is typically dominated by the complexity of the underlying PCP verifier. Moreover, the number of queries to the random oracle is $q_v = O(q \cdot \log l)$ (to check the Merkle opening proof for q answers).

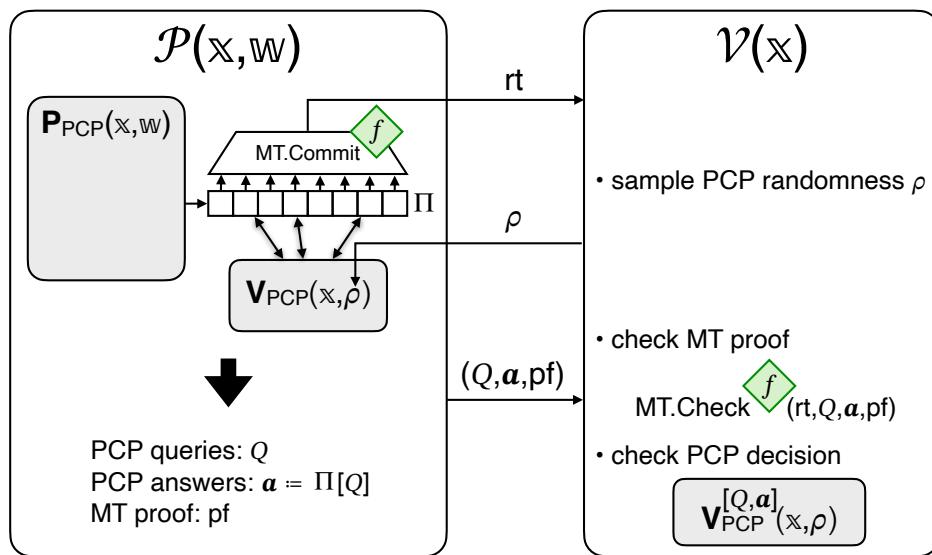


Figure 20.1: Diagram of the Kilian transformation (**Kilian** in Construction 20.1.1).

20.2 Non-adaptive soundness

We prove that Construction 20.1.1 is *non-adaptively* sound (Definition 4.2.2): for every instance that is not in the language, the probability that a bounded-query adversary convinces the argument verifier is small. Theorem 20.2.1 below provides an upper bound on this error, which is the soundness error of the underlying PCP plus a small term. This term represents the error incurred by “breaking” (the extractability property of) the Merkle commitment scheme used to commit to the PCP string. Later in Section 20.3 we prove that Construction 20.1.1 is also *adaptively* sound.

Theorem 20.2.1

Let PCP be a PCP for a relation \mathcal{R} with soundness error ϵ_{PCP} and proof length l . For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{IARG} := \text{Kilian}[\text{PCP}, \lambda, s]$ in Construction 20.1.1 is an interactive argument for \mathcal{R} with non-adaptive soundness error ϵ_{ARG} (see Definition 4.2.2) such that

$$\epsilon_{\text{ARG}}(\lambda, t, \mathbf{x}) \leq \epsilon_{\text{PCP}}(\mathbf{x}) + \kappa_{\text{MT}}(\lambda, t, l).$$

Above κ_{MT} is the MT extraction error from Lemma 18.5.1, and $\kappa_{\text{MT}}(\lambda, t, l) \leq \frac{1}{2} \cdot \frac{t^2}{2^\lambda}$ if $t \geq 4 \cdot (\log l + 1) \cdot l$.

We prove the theorem via a security reduction of the following form. For every fixed instance \mathbf{x} , any malicious argument prover $\tilde{\mathcal{P}}$ that convinces the argument verifier $\mathcal{V}(\mathbf{x})$ with probability δ can be transformed into a corresponding malicious PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}$ that convinces the PCP verifier $\mathbf{V}_{\text{PCP}}(\mathbf{x})$ with probability at least δ minus a small term. Theorem 20.2.1 follows by noting that if $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ then the PCP verifier accepts \mathbf{x} with probability at most $\epsilon_{\text{PCP}}(\mathbf{x})$.

Below we state the security reduction and then prove it.

Lemma 20.2.2. *There exists a PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}$ such that for every t -query argument prover $\tilde{\mathcal{P}}$ the following holds for every instance \mathbf{x} :*

$$\Pr \left[\langle \tilde{\mathcal{P}}^f, \mathcal{V}^f(\mathbf{x}) \rangle_{\text{ARG}} = 1 \mid f \leftarrow \mathcal{U}(\lambda) \right] \leq \Pr \left[\mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \mid \begin{array}{l} \Pi \leftarrow \tilde{\mathbf{P}}_{\text{PCP}}(\mathcal{P}) \\ \rho \leftarrow \{0, 1\}^r \end{array} \right] + \kappa_{\text{MT}}(\lambda, t, l).$$

Moreover, the running time of $\tilde{\mathbf{P}}_{\text{PCP}}$ equals the running time of $\tilde{\mathcal{P}}$ plus $\text{et}_{\text{MT}}(\lambda, \Sigma, l, s, t)$ where et_{MT} is the time complexity of the MT extractor in Lemma 18.5.1.

Proof. Let MT.Extract be the MT extractor from Lemma 18.5.1 in Section 18.5.1. The PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}$ works as follows.

$\tilde{\mathbf{P}}_{\text{PCP}}(\mathcal{P})$:

1. Run the argument prover $\tilde{\mathcal{P}}$, simulating its random oracle f , until $\tilde{\mathcal{P}}$ outputs a Merkle commitment rt ; let tr be the query-answer trace of this partial execution. (The argument prover $\tilde{\mathcal{P}}$ is an interactive algorithm, whose full execution would also involve giving PCP randomness ρ to $\tilde{\mathcal{P}}$ and then obtaining the next message of $\tilde{\mathcal{P}}$. Here we use a partial execution.)
2. Run the MT extractor to obtain a PCP string: $(\Pi, \text{td}) \leftarrow \text{MT.Extract}(\text{rt}, \text{tr})$. (MT.Extract also outputs an opening trapdoor td , but we do not use that here.)
3. Output the PCP string Π .

The PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}$ uses the argument prover $\tilde{\mathcal{P}}$ as a black box. The running time of $\tilde{\mathbf{P}}_{\text{PCP}}$ is the running time of (a partial execution of) $\tilde{\mathcal{P}}$ plus the running time of the (efficient) extractor MT.Extract on a query-answer trace of size at most t .

We argue that the PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}$ satisfies the claimed property.

By construction, the argument verifier \mathcal{V} accepts if and only if the PCP verifier \mathbf{V}_{PCP} accepts and the Merkle commitment checking algorithm $\text{MT}.\text{Check}$ accepts:

$$\Pr \left[\langle \tilde{\mathcal{P}}^f, \mathcal{V}^f(\mathbf{x}) \rangle_{\text{ARG}} = 1 \mid f \leftarrow \mathcal{U}(\lambda) \right] \\ = \Pr \left[\begin{array}{l} \mathbf{V}_{\text{PCP}}^{[Q, \mathbf{a}]}(\mathbf{x}, \rho) = 1 \\ \wedge \text{MT}.\text{Check}^f(\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}) = 1 \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{rt}, \mathbf{aux}) \leftarrow \tilde{\mathcal{P}}^f \\ \rho \leftarrow \{0, 1\}^r \\ (Q, \mathbf{a}, \mathbf{pf}) \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}, \rho) \end{array} \right].$$

We split the above probability as a sum of two probabilities that separately consider the disjoint cases $\Pi[Q] = \mathbf{a}$ and $\Pi[Q] \neq \mathbf{a}$ where Π is the PCP string output by $\text{MT}.\text{Extract}$:

$$\Pr \left[\begin{array}{l} \mathbf{V}_{\text{PCP}}^{[Q, \mathbf{a}]}(\mathbf{x}, \rho) = 1 \\ \wedge \text{MT}.\text{Check}^f(\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}) = 1 \\ \wedge \Pi[Q] = \mathbf{a} \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{rt}, \mathbf{aux}) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ (\Pi, \mathbf{td}) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}, \mathbf{tr}) \\ \rho \leftarrow \{0, 1\}^r \\ (Q, \mathbf{a}, \mathbf{pf}) \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}, \rho) \end{array} \right] \quad (20.2)$$

$$+ \Pr \left[\begin{array}{l} \mathbf{V}_{\text{PCP}}^{[Q, \mathbf{a}]}(\mathbf{x}, \rho) = 1 \\ \wedge \text{MT}.\text{Check}^f(\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}) = 1 \\ \wedge \Pi[Q] \neq \mathbf{a} \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{rt}, \mathbf{aux}) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ (\Pi, \mathbf{td}) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}, \mathbf{tr}) \\ \rho \leftarrow \{0, 1\}^r \\ (Q, \mathbf{a}, \mathbf{pf}) \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}, \rho) \end{array} \right]. \quad (20.3)$$

We conclude the proof by upper bounding each of the two terms.

- The term in Equation 20.2 is upper bounded by the PCP error, as we now explain. If $\Pi[Q] = \mathbf{a}$ (the extracted PCP string agrees with the query answers) then $\mathbf{V}_{\text{PCP}}^{[Q, \mathbf{a}]}(\mathbf{x}, \rho) = \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho)$ (the PCP verifier returns the same output). Moreover, the probability does not decrease if we omit the condition on $\text{MT}.\text{Check}$ (and thus can ignore the second output of $\tilde{\mathcal{P}}$). Therefore the term in Equation 20.2 can be upper bounded by the following expression:

$$\Pr \left[\mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{rt}, \mathbf{aux}) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ (\Pi, \mathbf{td}) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}, \mathbf{tr}) \\ \rho \leftarrow \{0, 1\}^r \end{array} \right].$$

The first three lines of the (right-side) experiment correspond to an execution of the PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}(\mathcal{P})$. Hence the above probability equals the following probability:

$$\Pr \left[\mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \mid \begin{array}{l} \Pi \leftarrow \tilde{\mathbf{P}}_{\text{PCP}}(\mathcal{P}) \\ \rho \leftarrow \{0, 1\}^r \end{array} \right].$$

- The term in Equation 20.3 is upper bounded via the MT extraction error (Lemma 18.5.1), as we now explain. The probability does not decrease if we omit the condition on \mathbf{V}_{PCP} . Therefore the term in Equation 20.3 can be upper bounded by the following expression:

$$\Pr \left[\begin{array}{l} \text{MT}.\text{Check}^f(\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}) = 1 \\ \wedge \Pi[Q] \neq \mathbf{a} \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{rt}, \mathbf{aux}) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ (\Pi, \mathbf{td}) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}, \mathbf{tr}) \\ \rho \leftarrow \{0, 1\}^r \\ (Q, \mathbf{a}, \mathbf{pf}) \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}, \rho) \end{array} \right].$$

In turn, the above probability is at most $\kappa_{\text{MT}}(\lambda, t, \mathsf{l})$ by Lemma 18.5.1. (Here we do not use the fact that Lemma 18.5.1 guarantees that \mathbf{td} can be used to extract a Merkle opening proof.)

□

20.3 Adaptive soundness

We prove that Construction 20.1.1 is *adaptively* sound (Definition 4.2.3): the probability that a bounded-query adversary outputs an instance not in the language and convinces the argument verifier is small. Theorem 20.3.1 below provides an upper bound on this error (which happens to be the same as the non-adaptive case in Theorem 20.2.1).

Theorem 20.3.1

Let \mathbf{PCP} be a PCP for a relation \mathcal{R} with soundness error $\epsilon_{\mathbf{PCP}}$ and proof length l . For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\mathbf{IARG} := \mathbf{Kilian}[\mathbf{PCP}, \lambda, s]$ in Construction 20.1.1 is an interactive argument for \mathcal{R} with adaptive soundness error $\epsilon_{\mathbf{ARG}}$ (see Definition 4.2.3) such that

$$\epsilon_{\mathbf{ARG}}(\lambda, t, n) \leq \epsilon_{\mathbf{PCP}}(n) + \kappa_{\text{MT}}(\lambda, t, \mathsf{l}).$$

Above κ_{MT} is the MT extraction error from Lemma 18.5.1, and $\kappa_{\text{MT}}(\lambda, t, \mathsf{l}) \leq \frac{1}{2} \cdot \frac{t^2}{2^\lambda}$ if $t \geq 4 \cdot (\log \mathsf{l} + 1) \cdot \mathsf{l}$.

We cannot use the security reduction in Lemma 20.2.2 to prove the adaptive case, because now the choice of instance by the malicious argument prover depends on the random oracle.

Nevertheless, we can adapt the statement of the lemma and its proof in a straightforward way to accommodate instances chosen by the malicious argument prover. Theorem 20.3.1 follows by noting that, for every instance \mathbf{x} such that $|\mathbf{x}| \leq n$ and $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$, the PCP verifier accepts \mathbf{x} with probability at most $\epsilon_{\mathbf{PCP}}(n)$.

Lemma 20.3.2. *There exists a PCP prover $\tilde{\mathbf{P}}_{\mathbf{PCP}}$ such that for every t -query argument prover $\tilde{\mathcal{P}}$ the following holds for every instance size bound $n \in \mathbb{N}$:*

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \langle \tilde{\mathcal{P}}^f(\mathbf{aux}), \mathcal{V}^f(\mathbf{x}) \rangle_{\mathbf{ARG}} = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \mathbf{aux}) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ & \leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\mathbf{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \Pi) \leftarrow \tilde{\mathbf{P}}_{\mathbf{PCP}}(\mathcal{P}) \\ \rho \leftarrow \{0, 1\}^r \end{array} \right] + \kappa_{\text{MT}}(\lambda, t, \mathsf{l}). \end{aligned}$$

Moreover, the running time of $\tilde{\mathbf{P}}_{\mathbf{PCP}}$ equals the running time of $\tilde{\mathcal{P}}$ plus $\mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathsf{l}, s, t)$ where \mathbf{et}_{MT} is the time complexity of the MT extractor in Lemma 18.5.1.

Proof. Below, since the argument prover $\tilde{\mathcal{P}}$ moves first, we can write that $\tilde{\mathcal{P}}$ outputs simultaneously the choice of instance \mathbf{x} , its first message \mathbf{rt} for the argument verifier \mathcal{V} , and an auxiliary state \mathbf{aux} for after receiving \mathcal{V} 's message ρ .

Let MT.Extract be the MT extractor from Lemma 18.5.1 in Section 18.5.1. The PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}$ works as follows.

$\tilde{\mathbf{P}}_{\text{PCP}}(\mathcal{P})$:

1. Run the argument prover $\tilde{\mathcal{P}}$, simulating its random oracle f , until $\tilde{\mathcal{P}}$ outputs $(\mathbf{x}, \mathbf{rt}, \mathbf{aux})$; let \mathbf{tr} be the query-answer trace of this partial execution. (The argument prover $\tilde{\mathcal{P}}$ is an interactive algorithm, whose full execution would also involve giving PCP randomness ρ to $\tilde{\mathcal{P}}$ and then obtaining the next message of $\tilde{\mathcal{P}}$. Here we use a partial execution.)
2. Run the MT extractor to obtain a PCP string: $(\Pi, \mathbf{td}) \leftarrow \text{MT.Extract}(\mathbf{rt}, \mathbf{tr})$. (MT.Extract also outputs an opening trapdoor \mathbf{td} , but we do not use that here.)
3. Output the instance \mathbf{x} and PCP string Π .

The PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}$ uses the argument prover $\tilde{\mathcal{P}}$ as a black box. The running time of $\tilde{\mathbf{P}}_{\text{PCP}}$ is the running time of (a partial execution of) $\tilde{\mathcal{P}}$ plus the running time of the (efficient) extractor MT.Extract on a query-answer trace of size at most t .

We argue that the PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}$ satisfies the claimed property.

By construction, the argument verifier \mathcal{V} accepts if and only if the PCP verifier \mathbf{V}_{PCP} accepts and the Merkle commitment checking algorithm MT.Check accepts:

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \langle \tilde{\mathcal{P}}^f(\mathbf{aux}), \mathcal{V}^f(\mathbf{x}) \rangle_{\text{ARG}} = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \mathbf{aux}) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ &= \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^{[Q, \mathbf{a}]}(\mathbf{x}, \rho) = 1 \\ \wedge \text{MT.Check}^f(\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \mathbf{rt}, \mathbf{aux}) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ \rho \leftarrow \{0, 1\}^r \\ (Q, \mathbf{a}, \mathbf{pf}) \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}, \rho) \end{array} \right]. \end{aligned}$$

We split the above probability as a sum of two probabilities that separately consider the disjoint cases $\Pi[Q] = \mathbf{a}$ and $\Pi[Q] \neq \mathbf{a}$ where Π is the PCP string output by MT.Extract :

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^{[Q, \mathbf{a}]}(\mathbf{x}, \rho) = 1 \\ \wedge \text{MT.Check}^f(\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}) = 1 \\ \wedge \Pi[Q] = \mathbf{a} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \mathbf{rt}, \mathbf{aux}) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ (\Pi, \mathbf{td}) \leftarrow \text{MT.Extract}(\mathbf{rt}, \mathbf{tr}) \\ \rho \leftarrow \{0, 1\}^r \\ (Q, \mathbf{a}, \mathbf{pf}) \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}, \rho) \end{array} \right] \quad (20.4)$$

$$+ \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^{[Q, \mathbf{a}]}(\mathbf{x}, \rho) = 1 \\ \wedge \text{MT.Check}^f(\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}) = 1 \\ \wedge \Pi[Q] \neq \mathbf{a} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \mathbf{rt}, \mathbf{aux}) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ (\Pi, \mathbf{td}) \leftarrow \text{MT.Extract}(\mathbf{rt}, \mathbf{tr}) \\ \rho \leftarrow \{0, 1\}^r \\ (Q, \mathbf{a}, \mathbf{pf}) \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}, \rho) \end{array} \right]. \quad (20.5)$$

We conclude the proof by upper bounding each of the two terms.

- The term in Equation 20.4 is upper bounded by the PCP error, as we now explain. If $\Pi[Q] = \mathbf{a}$ (the extracted PCP string agrees with the query answers) then $\mathbf{V}_{\text{PCP}}^{[Q, \mathbf{a}]}(\mathbf{x}, \rho) =$

$\mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho)$ (the PCP verifier returns the same output). Moreover, the probability does not decrease if we omit the condition on $\text{MT}.\text{Check}$ (and thus can ignore the second output of $\tilde{\mathcal{P}}$). Therefore the term in Equation 20.4 can be upper bounded by the following expression:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \mathbf{rt}, \mathbf{aux}) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ (\Pi, \mathbf{td}) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}, \mathbf{tr}) \\ \rho \leftarrow \{0, 1\}^r \end{array} \right].$$

The first three lines of the (right-side) experiment correspond to an execution of the PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}(\mathcal{P})$. Hence the above probability equals the following probability:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \Pi) \leftarrow \tilde{\mathbf{P}}_{\text{PCP}}(\mathcal{P}) \\ \rho \leftarrow \{0, 1\}^r \end{array} \right].$$

- The term in Equation 20.5 is upper bounded via the MT extraction error (Lemma 18.5.1), as we now explain. The probability does not decrease if we omit the condition on \mathbf{V}_{PCP} . Therefore the term in Equation 20.5 can be upper bounded by the following expression:

$$\Pr \left[\begin{array}{l} \text{MT}.\text{Check}^f(\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}) = 1 \\ \wedge \Pi[Q] \neq \mathbf{a} \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \mathbf{rt}, \mathbf{aux}) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ (\Pi, \mathbf{td}) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}, \mathbf{tr}) \\ \rho \leftarrow \{0, 1\}^r \\ (Q, \mathbf{a}, \mathbf{pf}) \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}, \rho) \end{array} \right].$$

In turn, the above probability is at most $\kappa_{\text{MT}}(\lambda, t, l)$ by Lemma 18.5.1. (Here we do not use the fact that Lemma 18.5.1 guarantees that \mathbf{td} can be used to extract a Merkle opening proof.)

□

20.4 Additional security definitions

In this book, interactive arguments serve as a warmup towards non-interactive counterparts. Therefore, here we only outline what happens for some security definitions beyond (non-adaptive and adaptive) soundness. We informally discuss how Construction 20.1.1 satisfies knowledge soundness and zero knowledge.

Knowledge soundness The interactive argument in Construction 20.1.1 satisfies (adaptive) knowledge soundness, assuming that the underlying PCP has knowledge soundness. A knowledge extractor receives as input an instance (chosen by the argument prover), the transcript of interaction, and the query-answer trace of the argument prover, and outputs a candidate witness. For this construction, the extractor outputs a candidate witness, which is the result of applying the PCP extractor to the PCP string output by the Merkle commitment extractor. One can show that if the PCP has knowledge soundness error κ_{PCP} , the knowledge soundness error of the interactive argument is $\kappa_{\text{ARG}}(\lambda, t, n) \leq \kappa_{\text{PCP}}(n) + \kappa_{\text{MT}}(\lambda, t, l)$ (which is analogous to the case of adaptive soundness in Theorem 20.3.1).

Zero knowledge The interactive argument in Construction 20.1.1 is *honest-verifier* zero knowledge, assuming the underlying PCP satisfies honest-verifier zero knowledge (and the privacy parameter s of the construction is large enough). Briefly, the zero knowledge simulator uses the PCP simulator to sample a simulated view of the PCP verifier and then uses the Merkle commitment simulator to sample a corresponding Merkle commitment. For an instance \mathbf{x} in the language, the simulation error is at most $z_{\text{PCP}}(\mathbf{x}) + z_{\text{MT}}(\lambda, \mathbf{l}(\mathbf{x}), s, \mathbf{q}(\mathbf{x}), t)$, where z_{MT} is the hiding error of the Merkle commitment scheme from Lemma 18.6.3.

The notion of honest-verifier zero-knowledge for an interactive argument suffices for many cryptographic applications and, in particular, suffices to obtain zero-knowledge non-interactive arguments via the Fiat–Shamir transformation (when applied to an interactive argument rather than an interactive proof).

One can also consider the stronger notion of *malicious-verifier* zero-knowledge for an interactive argument. In Construction 20.1.1 this means that a malicious argument verifier sends a message $\rho \in \{0, 1\}^r$ that may not be random; rather ρ depends arbitrarily on the argument prover's first message \mathbf{rt} . Simulating the verifier's view in this case involves some changes: (a) the simulator sends a dummy Merkle commitment \mathbf{rt} to the malicious verifier, who replies with a message $\rho \in \{0, 1\}^r$; (b) the PCP simulator must sample a PCP verifier view for the given ρ (rather than for a random ρ); (c) we allow programming the random oracle so that the simulator can construct a Merkle commitment on the local view that matches the previously sent commitment \mathbf{rt} . The simulation error is at most $z_{\text{PCP}}(\mathbf{x}) + z_{\text{MT}}(\lambda, \mathbf{l}(\mathbf{x}), s, \mathbf{q}(\mathbf{x}), t)$ plus a small error term representing the probability that the adversary detects the programmed locations of the random oracle.

21 The Micali transformation

We describe how to construct a succinct *non-interactive* argument, starting from any probabilistically checkable proof (PCP). This is known as the *Micali transformation* [Mic00], which builds on the Kilian transformation (the succinct interactive argument discussed in Chapter 20).

21.1 Construction

The construction can be informally described in two steps.

- First apply the Kilian transformation from Chapter 20. This transforms the given PCP into a succinct interactive argument with the structure of a SP (prover sends a commitment, then the verifier sends a challenge, and finally the prover sends a response).
- Then, using a separate random oracle, apply the Fiat–Shamir transformation for SPs from Chapter 10. This removes the interaction, resulting in a succinct non-interactive argument.

In more detail, the argument prover, after committing to the PCP string via a Merkle commitment, uses the random oracle to itself derive PCP randomness based on the Merkle commitment (rather than receiving the PCP randomness from the verifier). This enables the argument prover to send in a single message all the relevant information: the Merkle commitment, the PCP answers, and the Merkle opening proof authenticating them. The argument verifier re derives the PCP randomness, and checks that both PCP verifier and the Merkle commitment checking algorithm accept.

Similarly to the Fiat–Shamir transformation for SPs, there are delicate technical details to ensure adaptive security. The instance \mathbf{x} (whose membership in the language is being proved) is included in the query to the random oracle to derive PCP randomness. Moreover, this query also includes a random salt τ , sampled by the argument prover and sent to the argument verifier along with the other information. See Section 9.6 for an intuitive discussion that motivates these technical details in the context of the Fiat–Shamir transformation for SPs.

Below we describe the construction in detail.

Construction 21.1.1: Micali transformation

Let $\text{PCP} = (\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}})$ be a PCP with proof length l over alphabet Σ , query complexity q , and randomness complexity r . Let $\lambda \in \mathbb{N}$ be a security parameter and $s \in \mathbb{N}$ be a privacy parameter. Define $\text{MT} := \text{MT}[\lambda, \Sigma, l, s]$ to be the Merkle commitment scheme with the stated parameters.

We define $\text{NARG} := \text{Micali}[\text{PCP}, \lambda, s]$ to be the non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ constructed as follows. The argument prover \mathcal{P} receives as input an instance \mathbf{x} and witness \mathbf{w} , and the argument verifier \mathcal{V} receives as input the instance \mathbf{x} and an

argument string π . Both receive query access to random oracles $f = (f_{\text{MT}}, f_{\$}) \in \mathcal{U}((\lambda, r))$: (a) an oracle f_{MT} used for the Merkle commitment scheme MT , and (b) an oracle $f_{\$}$ used to derive PCP randomness; this implies that the oracle configuration cnf is defined as $\text{cnf}(\lambda, n) := (\lambda, r)$ (see Definition 7.1.1).

- $\mathcal{P}^f(\mathbf{x}, \mathbf{w})$:
 1. Compute the PCP string $\Pi := \mathbf{P}_{\text{PCP}}(\mathbf{x}, \mathbf{w}) \in \Sigma^l$.
 2. Commit to Π via the Merkle commitment scheme: $(\mathbf{rt}, \mathbf{td}) := \text{MT.Commit}^{f_{\text{MT}}}(\Pi)$.
 3. Sample a random salt $\tau \in \{0, 1\}^s$.
 4. Derive PCP randomness $\rho := f_{\$}(\mathbf{x}, \mathbf{rt}, \tau) \in \{0, 1\}^r$.
 5. Simulate the PCP verifier $\mathbf{V}_{\text{PCP}}^\Pi(\mathbf{x}, \rho)$, yielding a query set $Q \subseteq [l]$ with $|Q| \leq q$.
 6. Set $\mathbf{a} := \Pi[Q] \in \Sigma^Q$ to be the answers for the query set.
 7. Compute an opening proof for \mathbf{a} : $\mathbf{pf} := \text{MT.Open}^{f_{\text{MT}}}(\mathbf{td}, Q)$.
 8. Output $\pi := (\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}, \tau)$.
- $\mathcal{V}^f(\mathbf{x}, \pi)$:
 1. Parse the argument string π as a tuple $(\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}, \tau)$.
 2. Derive PCP randomness ρ as in Item 4 of the argument prover \mathcal{P} .
 3. Check that $\mathbf{V}_{\text{PCP}}^{[Q, \mathbf{a}]}(\mathbf{x}, \rho) = 1$. (The PCP verifier $\mathbf{V}_{\text{PCP}}(\mathbf{x}, \rho)$ accepts if each query $j \in Q$ is answered with $\mathbf{a}[j] \in \Sigma$. If any query falls outside the set Q then reject.)
 4. Check that $\text{MT.Check}^{f_{\text{MT}}}(\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}) = 1$. (The query answers $\mathbf{a} \in \Sigma^Q$ are authenticated with respect to the Merkle commitment \mathbf{rt} .)

Efficiency We discuss efficiency properties of the above construction.

- *Argument size.* The argument string π in the above construction contains the same information sent from the argument prover in the Kilian transformation (the Merkle commitment and the authenticated answers to the queries), plus a salt consisting of s bits. Similarly to Equation 20.1, this leads to an argument size with the same asymptotics:

$$\lambda + q \cdot (\log l + \log |\Sigma| + s + \lambda \cdot \log l) + s = O(q \cdot (\log |\Sigma| + s + \lambda \cdot \log l)). \quad (21.1)$$

The difference is that the argument prover provides the information in a single message, with randomness derived via the random oracle (as in the Fiat–Shamir transformation).

- *Prover complexity.* The complexity of the argument prover is typically dominated by the complexity of the underlying PCP prover, as in the Kilian transformation. Moreover, the number of queries to the random oracle is $q_{\mathcal{P}} = O(l)$:
 - $O(l)$ queries to the random oracle f_{MT} (to commit to the PCP string, as in the Kilian transformation); and
 - 1 query to the random oracle $f_{\$}$ (to derive PCP randomness).
- *Verifier complexity.* The complexity of the argument verifier is typically dominated by the complexity of the underlying PCP verifier, as in the Kilian transformation. Moreover, the number of queries to the random oracle is $q_{\mathcal{V}} = O(q \cdot \log l)$:

- $O(q \cdot \log l)$ queries to the random oracle f_{MT} (to check the Merkle opening proof for q answers, as in the Kilian transformation); and
- 1 query to the random oracle f_S (to derive PCP randomness, like the argument prover).

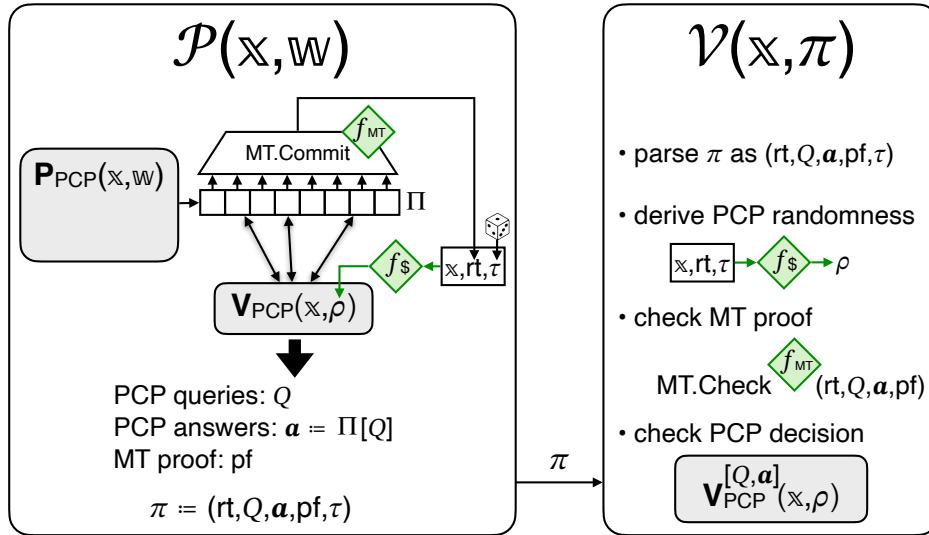


Figure 21.1: Diagram of the Micali transformation (**Micali** in Construction 21.1.1).

21.2 Soundness

We prove that Construction 21.1.1 is (adaptively) sound. The following theorem states that the (adaptive) soundness error of the non-interactive argument against a t -query argument prover is upper bounded by the soundness error of the underlying PCP *multiplied by $t + 1$* plus a small term. This latter term represents the error term incurred by “breaking” the Merkle commitment scheme used to commit to the PCP string.

Theorem 21.2.1

Let PCP be a PCP for a relation \mathcal{R} with soundness error ϵ_{PCP} and proof length l . For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $NARG := \mathbf{Micali}[PCP, \lambda, s]$ in Construction 21.1.1 is a non-interactive argument for \mathcal{R} with adaptive soundness error ϵ_{ARG} (see Definition 7.1.4) such that

$$\epsilon_{ARG}(\lambda, t, n) \leq (t + 1) \cdot \epsilon_{PCP}(n) + \kappa_{MT}(\lambda, t, l, t + 1, 1).$$

Above κ_{MT} is the MT multi-extraction error from Lemma 18.5.6, and $\kappa_{MT}(\lambda, t, l, t + 1, 1) \leq 2 \cdot \frac{t^2}{2^\lambda}$ if $6 \cdot 1 \cdot (\log l + 1) \leq t$.

The soundness error in Theorem 21.2.1 (soundness of the Micali transformation) differs from

the soundness error in Theorem 20.3.1 (soundness of the Kilian transformation) in two ways:

- the PCP soundness error $\epsilon_{\text{PCP}}(n)$ is multiplied by $(t + 1)$; and
- the additive error arising from “breaking” the Merkle commitment scheme is a multi-extraction error (one opening out of $t + 1$ commitments) rather than a single-extraction error.

In particular, for the same level of desired security, a PCP must be $(t + 1)$ times “sounder” to be used in the Micali transformation when compared to in the Kilian transformation.

The above differences come from the fact that in the Kilian transformation an argument prover has a single opportunity to attack the underlying PCP: after sending to the argument verifier a Merkle commitment \mathbf{rt} , the argument prover receives PCP randomness from the argument verifier, and has to open the requested locations in a way that is consistent with the Merkle commitment.

In contrast, in the Micali transformation, the argument prover can explore different choices of PCP randomness for different choices of Merkle commitments (in fact, even for the same Merkle commitment by changing the salt), because each PCP randomness is obtained as the answer to a certain query to the random oracle. Each such exploration costs a query, so a t -query argument prover can see at most t choices of PCP randomness; in fact, the argument prover can also output a Merkle commitment for which it did not query the random oracle (and hence for which it did not see the corresponding PCP randomness). This informally justifies the term $(t + 1) \cdot \epsilon_{\text{PCP}}(n)$ in the soundness error expression. The other term in the soundness error expression denotes the fact that in the security reduction we extract a PCP string from the Merkle commitment, out of up to $t + 1$ explored by the argument prover, that the argument prover outputs in the argument string. This is a multi-extraction error, unlike in the case of the Kilian transformation.

The above phenomenon is analogous to what happens in the Fiat–Shamir transformation for SPs (see Chapter 10), wherein the soundness error of the resulting non-interactive argument is $t + 1$ times the SP soundness error. No other error terms appear because no Merkle commitment schemes are used in that case.

Analyzing the Micali transformation as the “black-box” application of the Fiat–Shamir transformation to the Kilian transformation does not lead to an asymptotically tight result. (See Remark 21.2.5.) Instead, we analyze the Micali transformation directly: Theorem 21.2.1 directly follows from the lemma below. We separately bound the number $t_{\$}$ of queries to the (derived) oracle $f_{\$}$ and the number t_{MT} of queries to the (derived) oracle f_{MT} . Both are upper bounded by the number t of queries to the random oracle f .

Lemma 21.2.2. *There exists a PCP prover $\tilde{\mathbf{P}}_{\text{PCP}}$ such that for every $(t_{\text{MT}}, t_{\$})$ -query argument prover $\tilde{\mathcal{P}}$ the following holds for every instance size bound $n \in \mathbb{N}$:*

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| \begin{array}{l} f = (f_{\text{MT}}, f_{\$}) \leftarrow \mathcal{U}((\lambda, \mathbf{r})) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ \leq (t_{\$} + 1) \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \Pi) \leftarrow \tilde{\mathbf{P}}_{\text{PCP}}(\mathcal{P}) \\ \rho \leftarrow \{0, 1\}^r \end{array} \right] + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathbf{l}, t_{\$} + 1, 1).$$

We state a claim that reduces the argument prover to a PCP state-restoration prover, up to a multi-extraction error. Then we prove the lemma by using the fact that PCP state-restoration soundness can be related to the PCP’s (standard) soundness.

While the state-restoration game has a formal output, for the purpose of analysis we externalize certain quantities introduced during the execution of the game. We place these quantities under the arrow that leads to the game output. Specifically, in the claim below, we consider a query-answer trace tr_{MT} and a bit b_{MT} , which are set by the PCP state-restoration prover $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ that we construct.

Claim 21.2.3. *There exists a PCP state-restoration prover $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ such that, for every $(t_{\text{MT}}, t_{\$})$ -query argument prover $\tilde{\mathcal{P}}$, $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ makes at most $t_{\$}$ moves and the following two distributions are $\kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathsf{l}, t_{\$} + 1, 1)$ -close in statistical distance*

$$\left\{ \begin{array}{l|l} (\text{tr}_{\text{MT}}, \mathbf{x}, \Pi, b) & \left. \begin{array}{l} f = (f_{\text{MT}}, f_{\$}) \leftarrow \mathcal{U}((\lambda, \mathbf{r})) \\ (\mathbf{x}, \pi = (\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}, \tau)) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ (\Pi, \mathbf{td}) \leftarrow \text{MT.Extract}(\mathbf{rt}, \text{tr}_{\text{MT}}) \\ b = \mathcal{V}^f(\mathbf{x}, \pi) \end{array} \right\} \text{ and} \\ (\text{tr}_{\text{MT}}, \mathbf{x}, \Pi, b) & \left. \begin{array}{l} f = (f_{\text{MT}}, f_{\$}) \leftarrow \mathcal{U}((\lambda, \mathbf{r})) \\ (\mathbf{x}, \Pi, \sigma, \rho) \xleftarrow[\text{tr}_{\text{MT}}, b_{\text{MT}}]{} \text{Game}_{\text{PCP}}^{\text{sr}}(\lambda + s, f_{\$}, (\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}(\tilde{\mathcal{P}}))^{\mathbf{f}_{\text{MT}}}) \\ b := \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) \wedge b_{\text{MT}} \end{array} \right\}. \end{array} \right.$$

Moreover, the running time of $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ equals the running time of $\tilde{\mathcal{P}}$ plus $\mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathsf{l}, s, t_{\text{MT}}, t_{\$} + 1)$ where \mathbf{et}_{MT} is the time complexity of the MT extractor in Lemma 18.5.6.

Construction 21.2.4. Let MT.MultiExtract be the MT extractor for multiple commitments from Lemma 18.5.6 in Section 18.5.2. The PCP state-restoration prover $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}} := \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}(\tilde{\mathcal{P}})$ works as follows.

1. Lazily sample an oracle $\dot{g} \leftarrow \mathcal{U}(\mathbf{r})$ (to be used as auxiliary randomness).
2. Initialize an empty query-answer trace tr_{MT} .
3. Simulate $\tilde{\mathcal{P}}$ while answering its queries as described below.
4. When $\tilde{\mathcal{P}}$ performs a query to the oracle f_{MT} , answer according to f_{MT} and append the resulting query-answer pair to tr_{MT} .
5. When $\tilde{\mathcal{P}}$ performs the j -th query x_j to the oracle $f_{\$}$, do the following:
 - a) If x_j can be parsed as a tuple $(\mathbf{x}_j, \mathbf{rt}_j, \tau_j)$ where \mathbf{x}_j is an instance, \mathbf{rt}_j a Merkle commitment, and τ_j a salt string:
 - i. Let $\text{tr}_j \subseteq \text{tr}_{\text{MT}}$ be the query-answer pairs for f_{MT} between the previous iteration and the current iteration (or the beginning if this is the first iteration).
 - ii. Run the Merkle extractor: $(\Pi_j, \mathbf{td}_j) := \text{MT.MultiExtract}(\mathbf{rt}_j, \text{tr}_j)$.
 - iii. Set the salt string $\sigma_j := (\mathbf{rt}_j, \tau_j)$.
 - iv. Submit in Item 1a of the game the move $(\mathbf{x}_j, \Pi_j, \sigma_j)$.
 - v. Receive PCP randomness $\rho_j \in \{0, 1\}^r$ from the game.
 - b) Otherwise set $\rho_j := \dot{g}(x_j)$.
 - c) Return ρ_j to $\tilde{\mathcal{P}}$ as the answer to its query.
6. Let (\mathbf{x}, π) be the output of $\tilde{\mathcal{P}}$ when it halts, and parse π as $(\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}, \tau)$.
7. Let $\text{tr}' \subseteq \text{tr}_{\text{MT}}$ be the query-answer pairs for f_{MT} between the last iteration and $\tilde{\mathcal{P}}$'s halting.
8. Run the Merkle extractor to obtain a PCP string: $(\Pi, \mathbf{td}) := \text{MT.MultiExtract}(\mathbf{rt}, \text{tr}')$.
9. Set the salt string $\sigma := (\mathbf{rt}, \tau)$.
10. Set $b_{\text{MT}} := \text{MT.Check}^{f_{\text{MT}}}(\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf})$. (This bit is only used in the analysis.)
11. Output $(\mathbf{x}, \Pi, \sigma)$.

The prover $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ potentially makes a move (in Item 5(a)iv) for every query of $\tilde{\mathcal{P}}$ to the oracle $f_{\$}$ (no other moves are performed). Since there are at most $t_{\$}$ queries to $f_{\$}$, we get that $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ performs at most $t_{\$}$ moves. In terms of running time, $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ simulates $\tilde{\mathcal{P}}$ (which takes the running time of $\tilde{\mathcal{P}}$) and additionally performs at most $t_{\$} + 1$ Merkle commitment extractions (at most once per query of $\tilde{\mathcal{P}}$ to $f_{\$}$ plus an additional one for the output of $\tilde{\mathcal{P}}$). The total time for all extractions is given by the expression $\mathbf{et}_{\text{MT}}(\lambda, \Sigma, l, s, t_{\text{MT}}, t_{\$} + 1)$ where \mathbf{et}_{MT} is the (total) time complexity of $\text{MT}.\text{MultiExtract}$ in Lemma 18.5.6. Hence the running time of $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ is as in the claim.

Proof of Claim 21.2.3. We define an event E over the probability space of sampling a random oracle $f = (f_{\text{MT}}, f_{\$}) \leftarrow \mathcal{U}((\lambda, r))$, running $(\mathbf{x}, \pi = (\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}, \tau)) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f$ (as in the top distribution of the claim statement), and then running $\text{Game}_{\text{PCP}}^{\text{sr}}(\lambda + s, f_{\$}, (\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}(\tilde{\mathcal{P}}))^f_{\text{MT}})$ (as in the bottom distribution of the claim statement). The event E is as follows:

1. $\mathcal{V}^f(\mathbf{x}, \pi) = 1$ and $\Pi[Q] \neq \mathbf{a}$ (the argument verifier accepts but the MT extractor fails in extracting a corresponding PCP string in Item 8); OR
2. there exists $j, j' \in [t_{\$}]$ such that $\mathbf{rt}_j = \mathbf{rt}_{j'}$ but $\Pi_j \neq \Pi_{j'}$ (the MT extractor outputs two different strings on the same Merkle commitment but with different query-answer traces).

It suffices to upper bound the probability that E holds and showing that the following two distributions (which additionally include the randomness ρ) are identical:

$$\begin{aligned} & \left\{ \begin{array}{l} (tr_{\text{MT}}, \rho, \mathbf{x}, \Pi, b) \\ \text{conditioned on} \\ \overline{E} \end{array} \middle| \begin{array}{l} f = (f_{\text{MT}}, f_{\$}) \leftarrow \mathcal{U}((\lambda, r)) \\ (\mathbf{x}, \pi = (\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}, \tau)) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ (\Pi, \mathbf{td}) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}, tr_{\text{MT}}) \\ b := \mathcal{V}^f(\mathbf{x}, \pi) \\ \rho := f_{\$}(\mathbf{x}, \mathbf{rt}, \tau) \end{array} \right\} \\ & \equiv \\ & \left\{ \begin{array}{l} (tr_{\text{MT}}, \rho, \mathbf{x}, \Pi, b) \\ \text{conditioned on} \\ \overline{E} \end{array} \middle| \begin{array}{l} f = (f_{\text{MT}}, f_{\$}) \leftarrow \mathcal{U}((\lambda, r)) \\ (\mathbf{x}, \Pi, \sigma, \rho) \xleftarrow[\mathbf{tr}_{\text{MT}}, b_{\text{MT}}]{} \text{Game}_{\text{PCP}}^{\text{sr}}(\lambda + s, f_{\$}, (\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}(\tilde{\mathcal{P}}))^f_{\text{MT}}) \\ b := \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) \wedge b_{\text{MT}} \end{array} \right\}. \end{aligned}$$

Below we refer to the two experiments as the left-side experiment (the one to the left of the equality) and the right-side experiment (the one to the right of the equality). Both experiments involve the execution (or emulation) of $\tilde{\mathcal{P}}$.

First we argue that the distribution of *answers* to oracle queries are identical in both experiments.

- *Queries to the oracle f_{MT} .* In the left-side experiment $\tilde{\mathcal{P}}$ directly queries f_{MT} , while in the right-side experiment $\tilde{\mathcal{P}}$ queries f_{MT} through $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ (which emulates $\tilde{\mathcal{P}}$ and has query access to f_{MT}). Both experiments are conditioned on the same event \overline{E} . We conclude that the distribution of the query-answer trace tr_{MT} is identical in both experiments.
- *Queries to the oracle $f_{\$}$.* In the left-side experiment, $\tilde{\mathcal{P}}$ directly queries $f_{\$}$, receiving corresponding answers, conditioned on the event \overline{E} .

In the right-side experiment, $f_{\$}$ is given as the auxiliary randomness \mathbf{rnd} in the PCP state-restoration game, and queries of $\tilde{\mathcal{P}}$ to $f_{\$}$ are translated to moves of $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ or forwarded

to the (lazily sampled) oracle \dot{g} . Specifically, if a query x_j to $f_{\$}$ has the form $(\mathbf{x}_j, \mathbf{rt}_j, \tau_j)$ then it receives the answer $\text{rnd}(\mathbf{x}_j, \Pi_j, \sigma_j) = f_{\$}(\mathbf{x}_j, \Pi_j, \sigma_j)$ for $\sigma_j = (\mathbf{rt}_j, \tau_j)$; otherwise, it receives the answer $\dot{g}(x_j)$. All answers are conditioned on the event E .

Every unique query x_j is mapped to a unique PCP state-restoration move $(\mathbf{x}_j, \Pi_j, \sigma_j)$ or to a unique input to \dot{g} . Moreover, any duplicate query is mapped to the same move in the PCP state-restoration game or the same input to \dot{g} . The latter case is clear but the former case follows from the fact that E does not hold, as we now explain. Assume that rounds j and j' are duplicate queries that can be parsed as identical tuples $(\mathbf{x}_j, \mathbf{rt}_j, \tau_j)$ and $(\mathbf{x}_{j'}, \mathbf{rt}_{j'}, \tau_{j'})$. The mapped queries are identical except (potentially) the strings Π_j and $\Pi_{j'}$. However both are extracted from the same Merkle commitment $\mathbf{rt}_j = \mathbf{rt}_{j'}$, so it must be that $\Pi_j = \Pi_{j'}$: if $\Pi_j \neq \Pi_{j'}$ then this would contradict the assumption that E does not hold. Note that while queries to f_{MT} are the same in both experiments, queries to $f_{\$}$ (in the left-side experiment) differ from moves in the PCP state-restoration game (in the right-side experiment).

Next, given that $\tilde{\mathcal{P}}$ sees the same distribution of answers, we argue that all elements in $(\mathbf{tr}_{\text{MT}}, \rho, \mathbf{x}, \Pi, b)$ have the same distribution in both experiments.

- *Distribution of \mathbf{x}, Π .* Since $\tilde{\mathcal{P}}$ receives answers that have the same distribution in both experiments (as argued above), its output (\mathbf{x}, π) has the same distribution in both experiments as well. Recall that $\pi = (\mathbf{rt}, Q, \mathbf{a}, \text{pf}, \tau)$ and that $(\Pi, \text{td}) \leftarrow \text{MT.Extract}(\mathbf{rt}, \mathbf{tr}_{\text{MT}})$. Since \mathbf{tr}_{MT} is identical in both experiments, we conclude that Π is identical as well.
- *Distribution of ρ .* For the same reasons as the answer of the queries, ρ is identically distributed in both experiments. If the query-answer pair $((\mathbf{x}, \mathbf{rt}, \tau), \rho)$ was already performed, then since the answers to $f_{\$}$ are uniformly distributed we have that ρ is uniformly distributed in both sides. Otherwise, the query-answer pair $((\mathbf{x}, \mathbf{rt}, \tau), \rho)$ can be viewed as an additional query, which by the same argument is uniformly distributed.
- *Distribution of the bit b .* The bit b is a function of (\mathbf{x}, ρ, π) in the left-side experiment and is a function of $(\mathbf{x}, \rho, \Pi, b_{\text{MT}})$ in the right-side experiment. We show that $b = 1$ on the left-side experiment implies that $b = 1$ on the right-side experiment, and the same holds for $b = 0$.

If $\mathcal{V}^f(\mathbf{x}, \pi) = 1$ ($b = 1$ in the left-side experiment) then we know that: $\mathbf{V}_{\text{PCP}}^{[Q, \mathbf{a}]}(\mathbf{x}, \rho) = 1$ and $\text{MT.Check}^{f_{\text{MT}}}(\mathbf{rt}, Q, \mathbf{a}, \text{pf}) = 1$. Since the event E does not hold, this implies that $\Pi[Q] = \mathbf{a}$, which in turn implies that $\mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1$. Since $\text{MT.Check}^{f_{\text{MT}}}(\mathbf{rt}, Q, \mathbf{a}, \text{pf}) = 1$ we have that $b_{\text{MT}} = 1$ in the right-side experiment. Overall, we get that $b = 1$ in the right-side experiment.

If $\mathcal{V}^f(\mathbf{x}, \pi) = 0$ ($b = 0$ in the left-side experiment), then either: (a) $\mathbf{V}_{\text{PCP}}^{[Q, \mathbf{a}]}(\mathbf{x}, \rho) = 0$; or (b) $\text{MT.Check}^{f_{\text{MT}}}(\mathbf{rt}, Q, \mathbf{a}, \text{pf}) = 0$. If $\text{MT.Check}^{f_{\text{MT}}}(\mathbf{rt}, Q, \mathbf{a}, \text{pf}) = 0$ then $b_{\text{MT}} = 0$. Otherwise $\text{MT.Check}^{f_{\text{MT}}}(\mathbf{rt}, Q, \mathbf{a}, \text{pf}) = 1$ and $\mathbf{V}_{\text{PCP}}^{[Q, \mathbf{a}]}(\mathbf{x}, \rho) = 0$, which implies that $\Pi[Q] = \mathbf{a}$ (as else we would have $\text{MT.Check}^{f_{\text{MT}}}(\mathbf{rt}, Q, \mathbf{a}, \text{pf}) = 0$), which means that $\mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 0$. In both cases $b = 0$ in the right-side experiment.

Finally, we are left to upper bound the probability of the event E . This is straightforward via Lemma 18.5.6, which considers *extraction from multiple commitments*. Indeed, we invoke the MT extractor multiple times, each time on a (possibly) different root \mathbf{rt}_j with trace \mathbf{tr}_j

(recall that the MT extractor is a stateful algorithm). According to the lemma, the probability that there exists an execution for which the MT extractor does not succeed (but for which the adversary outputs a valid proof) is upper bounded by $\kappa_{\text{MT}}(\lambda, t_{\text{MT}}, l, t_s + 1, 1)$. Here the function κ_{MT} gets five parameters: λ is the security parameter, t_{MT} is the query bound, l is the length of the PCP string Π , $t_s + 1$ is the number of output Merkle commitments, and 1 is the number of opened Merkle commitments (the argument prover opens one of the output commitments). \square

Proof of Lemma 21.2.2. The probability statement on the left-side of the inequality is a function of the instance \mathbf{x} and the bit $b = \mathcal{V}^f(\mathbf{x}, \pi)$, generated from a random experiment. Using Claim 21.2.3, we can switch the experiment with the argument prover for the corresponding experiment with the state-restoration prover, the same instance, and the bit $b = \mathbf{V}_{\text{PCP}}^\Pi(\mathbf{x}, \rho) \wedge b_{\text{MT}}$.

Hence we get that the probability in the lemma statement can be upper bounded as follows:

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| \begin{array}{l} f = (f_{\text{MT}}, f_s) \leftarrow \mathcal{U}((\lambda, r)) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ & \leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^\Pi(\mathbf{x}, \rho) = 1 \\ \wedge b_{\text{MT}} = 1 \end{array} \middle| \begin{array}{l} f = (f_{\text{MT}}, f_s) \leftarrow \mathcal{U}((\lambda, r)) \\ (\mathbf{x}, \Pi, \sigma, \rho) \xleftarrow[b_{\text{MT}}]{} \mathbf{Game}_{\text{PCP}}^{\text{sr}}(\lambda + s, f_s, (\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}})^{f_{\text{MT}}}) \end{array} \right] \\ & \quad + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, l, t_s + 1, 1) \quad (\text{by Claim 21.2.3}) \\ & \leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^\Pi(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} f = (f_{\text{MT}}, f_s) \leftarrow \mathcal{U}((\lambda, r)) \\ (\mathbf{x}, \Pi, \sigma, \rho) \leftarrow \mathbf{Game}_{\text{PCP}}^{\text{sr}}(\lambda + s, f_s, (\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}})^{f_{\text{MT}}}) \end{array} \right] \\ & \quad + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, l, t_s + 1, 1) \quad (\text{by dropping the condition } b_{\text{MT}} = 1) . \end{aligned}$$

Above $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}} := \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}(\tilde{\mathcal{P}})$ is the PCP state-restoration prover obtained from $\tilde{\mathcal{P}}$ in the claim. The argument prover $\tilde{\mathcal{P}}$ makes at most t_s queries to f_s , so $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ makes at most t_s moves, and so (by Definition 19.2.2) the above probability is upper bounded by the following expression:

$$\epsilon_{\text{PCP}}^{\text{sr}}(\lambda + s, t_s, n) + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, l, t_s + 1, 1) .$$

By Theorem 19.2.3 (which bounds PCP state-restoration soundness error in terms of soundness error), the above probability is upper bounded by the following expression:

$$(t_s + 1) \cdot \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{PCP}}^\Pi(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} (\mathbf{x}, \Pi) \leftarrow \tilde{\mathbf{P}}_{\text{PCP}} \\ \rho \leftarrow \{0, 1\}^r \end{array} \right] + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, l, t_s + 1, 1) .$$

Above, $\tilde{\mathbf{P}}_{\text{PCP}} := \tilde{\mathbf{P}}_{\text{PCP}}(s, t, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}})$ is the PCP prover obtained from Theorem 19.2.3. \square

Remark 21.2.5 (alternative soundness analysis). In Section 21.1 we explain how the Micali transformation can be viewed as the Fiat–Shamir transformation for SPs applied to the (3-message public-coin) interactive argument resulting from the Kilian transformation (applied to the PCP).

So why does the above soundness analysis not follow this structure? (We do not simply combine the soundness guarantees of the Fiat–Shamir transformation and of the Kilian transformation.)

One issue is that we would need to extend the soundness analysis of the Fiat–Shamir transformation to work for a relaxation of SPs that are interactive arguments. The Kilian transformation outputs a protocol with the structure of an SP but with weaker soundness: it is an interactive argument rather than an interactive proof. That said, this extension could be carried out, which would lead to a modular analysis of the Micali transformation, resolving this particular issue.

There is another, more important, issue though. A modular analysis based on the two transformations would incur an undesirable overhead in soundness error, as we now explain. The Kilian transformation has a soundness error that is the sum of the PCP soundness error and a Merkle commitment single-extraction error. The Fiat–Shamir transformation for SPs (and relaxations thereof) has a soundness error that is $t_{\$} + 1$ times the soundness error of the SP. Overall this would establish a soundness error for the Micali transformation that is

$$(t_{\$} + 1) \cdot (\epsilon_{\text{PCP}}(n) + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, l)) .$$

However, the upper bound that we establish for the Micali transformation is better:

$$(t_{\$} + 1) \cdot \epsilon_{\text{PCP}}(n) + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, l, t_{\$} + 1, 1) .$$

Indeed, $(t_{\$} + 1)$ times the single-extraction error $\kappa_{\text{MT}}(\lambda, t_{\text{MT}}, l)$ is much bigger than the multi-extraction error $\kappa_{\text{MT}}(\lambda, t_{\text{MT}}, l, t_{\$} + 1, 1)$. Roughly, the former is $O(\frac{t_{\$} t_{\text{MT}}^2}{2^\lambda})$ while the latter is $O(\frac{t_{\text{MT}}^2}{2^\lambda})$.

Remark 21.2.6 (optimization via domain separation). The expression in Theorem 21.2.1 is essentially tight. Nevertheless, a minor modification to the Micali transformation yields modest improvements in security. Briefly, the idea is to use multiple oracles within the Merkle commitment (e.g., via domain separation).

22 Additional security definitions

We prove that Construction 21.1.1 satisfies additional security definitions.

22.1 Knowledge soundness

In Construction 21.1.1, if the PCP satisfies knowledge soundness then the resulting non-interactive argument satisfies adaptive knowledge soundness.

Theorem 22.1.1

Let PCP be a PCP for a relation \mathcal{R} with knowledge soundness error κ_{PCP} (see Definition 19.1.3). For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{NARG} := \text{Micali}[\text{PCP}, \lambda, s]$ in Construction 21.1.1 is a non-interactive argument for \mathcal{R} with knowledge soundness error κ_{ARG} (see Definition 7.1.5) such that

$$\kappa_{\text{ARG}}(\lambda, t, n) \leq (t+1) \cdot \kappa_{\text{PCP}}(n) + \kappa_{\text{MT}}(\lambda, t, l, t+1, 1).$$

Above κ_{MT} is the MT multi-extraction error from Lemma 18.5.6, and $\kappa_{\text{MT}}(\lambda, t, l, t+1, 1) \leq 2 \cdot \frac{t^2}{2^\lambda}$ if $6 \cdot 1 \cdot (\log l + 1) \leq t$.

The expression above is the same as in Theorem 21.2.1 (soundness of the Micali transformation), except that soundness error is replaced by knowledge soundness error. Intuitively, this is because the soundness analysis *already* involves the extraction of a PCP string (via the MT extractor), which the PCP extractor can use to recover a witness for the instance. The multiplicative loss $t+1$ persists, as a malicious argument prover can attack the PCP up to $t+1$ times.

The proof of the theorem follows steps similar to those used in the soundness analysis for Theorem 21.2.1. First, we (re)use Claim 21.2.3 to reduce an adversary against the non-interactive argument to an adversary performing a PCP state-restoration attack. Then, we use Theorem 19.2.6 to upper bound the probability that the adversary succeeds in a PCP state-restoration attack against knowledge soundness in terms of the PCP knowledge soundness error.

Proof of Theorem 22.1.1. Let \mathbf{E}_{PCP} be the knowledge extractor for PCP, and let $\mathbf{E}_{\text{PCP}}^{\text{sr}}$ be the PCP state-restoration extractor obtained from \mathbf{E}_{PCP} via Theorem 19.2.6.

Fix any t -query malicious argument prover $\tilde{\mathcal{P}}$ and instance size bound $n \in \mathbb{N}$. Let tr be the query-answer trace of $\tilde{\mathcal{P}}$; let tr_{MT} and tr_s be the query-answer traces for the oracles f_{MT} and f_s , respectively. Let $t_{\text{MT}} := |\text{tr}_{\text{MT}}|$ and $t_s := |\text{tr}_s|$, and observe that $t_{\text{MT}} + t_s = t$.

The extractor \mathcal{E} , given input $(\mathbf{x}, \pi, \text{tr})$, works as follows.

$\mathcal{E}(\mathbf{x}, \pi, \text{tr})$:

1. Parse π as a tuple $(\text{rt}, Q, \mathbf{a}, \text{pf}, \tau)$.
2. Derive tr_{MT} from tr .
3. Compute $(\Pi, \text{td}) \leftarrow \text{MT.Extract}(\text{rt}, \text{tr}_{\text{MT}})$.
4. Output $\mathbf{w} := \mathbf{E}_{\text{PCP}}^{\text{sr}}(\mathbf{x}, \Pi)$.

By Theorem 19.2.6 (which bounds PCP state-restoration knowledge soundness error in terms of PCP knowledge soundness error) and Claim 21.2.3 (and the fact that $t_{\text{MT}}, t_{\$} \leq t$) we conclude that:

$$\begin{aligned}
& \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| \begin{array}{l} f = (f_{\text{MT}}, f_{\$}) \leftarrow \mathcal{U}((\lambda, \mathbf{r})) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ \mathbf{w} \leftarrow \mathcal{E}(\mathbf{x}, \pi, \text{tr}) \end{array} \right] \\
&= \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| \begin{array}{l} f = (f_{\text{MT}}, f_{\$}) \leftarrow \mathcal{U}((\lambda, \mathbf{r})) \\ (\mathbf{x}, \pi) \xleftarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ \pi := (\text{rt}, Q, \mathbf{a}, \text{pf}, \tau) \\ (\Pi, \text{td}) \leftarrow \text{MT.Extract}(\text{rt}, \text{tr}_{\text{MT}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{PCP}}^{\text{sr}}(\mathbf{x}, \Pi) \end{array} \right] \quad (\text{by definition of } \mathcal{E}) \\
&= \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \\ \wedge b_{\text{MT}} = 1 \end{array} \middle| \begin{array}{l} f = (f_{\text{MT}}, f_{\$}) \leftarrow \mathcal{U}((\lambda, \mathbf{r})) \\ (\mathbf{x}, \Pi, \sigma, \rho) \xleftarrow{\text{tr}^{\text{sr}}, b_{\text{MT}}} \text{Game}_{\text{PCP}}^{\text{sr}}(\lambda + s, f_{\$}, (\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}(\tilde{\mathcal{P}}))^f_{\text{MT}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{PCP}}^{\text{sr}}(\mathbf{x}, \Pi) \\ + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathbf{l}, t_{\$} + 1, 1) \end{array} \right] \\
&\leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{PCP}}^{\Pi}(\mathbf{x}, \rho) = 1 \end{array} \middle| \begin{array}{l} f = (f_{\text{MT}}, f_{\$}) \leftarrow \mathcal{U}((\lambda, \mathbf{r})) \\ (\mathbf{x}, \Pi, \sigma, \rho) \xleftarrow{\text{tr}^{\text{sr}}} \text{Game}_{\text{PCP}}^{\text{sr}}(\lambda + s, f_{\$}, (\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}(\tilde{\mathcal{P}}))^f_{\text{MT}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{PCP}}^{\text{sr}}(\mathbf{x}, \Pi) \\ + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathbf{l}, t_{\$} + 1, 1) \end{array} \right] \\
&\leq \kappa_{\text{PCP}}^{\text{sr}}(\lambda + s, t_{\$}, n) + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathbf{l}, t_{\$} + 1, 1) \quad (\text{by Definition 19.2.5}) \\
&\leq (t_{\$} + 1) \cdot \kappa_{\text{PCP}}(n) + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathbf{l}, t_{\$} + 1, 1) \quad (\text{by Theorem 19.2.6}) \\
&\leq (t + 1) \cdot \kappa_{\text{PCP}}(n) + \kappa_{\text{MT}}(\lambda, t, \mathbf{l}, t + 1, 1) \quad (\text{since } t_{\text{MT}}, t_{\$} \leq t).
\end{aligned}$$

□

22.2 Zero knowledge

In Construction 21.1.1, if the PCP satisfies honest-verifier zero knowledge (and the privacy parameter s of the construction is large enough) then the resulting non-interactive argument satisfies adaptive zero knowledge.

Theorem 22.2.1

Let PCP be a PCP for a relation \mathcal{R} with proof length l , query complexity q , and honest-verifier zero-knowledge error z_{PCP} (see Definition 19.1.5). For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{NARG} := \text{Micali}[\text{PCP}, \lambda, s]$ in Construction 21.1.1 is a non-interactive argument for \mathcal{R} with adaptive zero-knowledge error z_{ARG} (see Definition 7.1.8) such that

$$z_{\text{ARG}}(\lambda, t, n) \leq z_{\text{PCP}}(n) + z_{\text{MT}}(\lambda, l(n), s, q(n), t) + \frac{t}{2^s},$$

where z_{MT} is the hiding error of the Merkle commitment scheme from Lemma 18.6.3.

The above expression has an intuitive explanation. Any given argument string $\pi = (\text{rt}, Q, \mathbf{a}, \text{pf}, \tau)$ contains PCP answers and a Merkle opening proof; the former incurs a statistical error z_{PCP} determined by the zero-knowledge property of the PCP, and the latter incurs a statistical error z_{MT} determined by the privacy property of the Merkle commitment scheme. The PCP merely needs to be zero knowledge against the honest verifier, because the PCP answers are produced based on PCP randomness obtained as answer from the random oracle, which is uniformly random.

The expression in the lemma also includes another term that arises for technical reasons due to rare events when the simulator programs random locations of the random oracle.

We formalize the above intuition by constructing the simulator and then proving the lemma by showing that its simulation error is as claimed.

Construction 22.2.2. The simulator is an algorithm $\mathcal{S}^f(\mathbf{x})$ that works as follows. Below we denote by \mathbf{S}_{PCP} the honest-verifier zero-knowledge simulator of the PCP (see Definition 19.1.5).

- $\mathcal{S}^f(\mathbf{x})$:

1. Sample a simulated view of the PCP verifier: $(\mathbf{x}, \rho, Q, \mathbf{a}) \leftarrow \mathbf{S}_{\text{PCP}}(\mathbf{x})$.
2. Run the MT simulator with f_{MT} as the oracle: $(\text{rt}, \text{pf}) \leftarrow \text{MT.Simulate}^{f_{\text{MT}}}(Q, \mathbf{a})$.
3. Sample a random salt $\tau \in \{0, 1\}^s$.
4. Set the argument string $\pi := (\text{rt}, Q, \mathbf{a}, \text{pf}, \tau)$.
5. Set the query-answer list $\mu_{\text{MT}} := \emptyset$. (The oracle f_{MT} is not programmed.)
6. Set the query-answer list $\mu_{\$} := \{((\mathbf{x}, \text{rt}, \tau), \rho)\}$, to be used to program oracle $f_{\$}$.
7. Set $\mu := (\mu_{\text{MT}}, \mu_{\$})$.
8. Output (π, μ) .

Note that \mathcal{S} programs the oracle $f_{\$}$ at one point and does not program the oracle f_{MT} (and, conversely, \mathcal{S} queries f_{MT} but does not query $f_{\$}$).

Proof. Fix a query bound $t \in \mathbb{N}$, t -query admissible adversary \mathcal{A} , and instance bound $n \in \mathbb{N}$. We wish to upper bound the statistical distance between the output of \mathcal{A} in the real world and in the simulated world. For that, we introduce two hybrid simulators (that take the witness as input):

- $\mathcal{S}_{\text{H2}}^f(\mathbf{x}, \mathbf{w})$ acts as $\mathcal{S}^f(\mathbf{x})$ except that the view in Item 1 is sampled as a real view $(\mathbf{x}, \rho, Q, \mathbf{a}) \leftarrow \text{View}_{\text{PCP}}(\mathbf{P}_{\text{PCP}}, \mathbf{V}_{\text{PCP}}, \mathbf{x}, \mathbf{w})$ of the PCP.

- $\mathcal{S}_{H1}^f(x, w)$ acts as $\mathcal{S}_{H2}^f(x, w)$ except that it samples the Merkle commitment using $MT.\text{Commit}$ and opening proof using $MT.\text{Open}$ (instead of via $MT.\text{Simulate}$).

We list the real-world, first-hybrid-world, second-hybrid-world, and simulated-world distributions, making explicit the operations underlying the relevant algorithms.

1. The real-world distribution:

$$\mathcal{D}_{\text{real}} := \left\{ \text{out} \mid \begin{array}{l} f \leftarrow \mathcal{U}((\lambda, r)) \\ (x, w, \text{aux}) \leftarrow \mathcal{A}^f \\ \pi \leftarrow \mathcal{P}^f(x, w) \\ \text{out} \leftarrow \mathcal{A}^f(\text{aux}, \pi) \end{array} \right\} \equiv \left\{ \text{out} \mid \begin{array}{l} f = (f_{\text{MT}}, f_{\$}) \leftarrow \mathcal{U}((\lambda, r)) \\ (x, w, \text{aux}) \leftarrow \mathcal{A}^f \\ \Pi \leftarrow \mathbf{P}_{\text{PCP}}(x, w) \\ (rt, td) \leftarrow MT.\text{Commit}^{f_{\text{MT}}}(\Pi) \\ \tau \leftarrow \{0, 1\}^s \\ \rho := f_{\$}(x, rt, \tau) \\ Q := \text{queries of } \mathbf{V}_{\text{PCP}}^{\Pi}(x, \rho) \\ \mathbf{a} := \Pi[Q] \\ pf := MT.\text{Open}^{f_{\text{MT}}}(td, Q) \\ \pi := (rt, Q, \mathbf{a}, pf, \tau) \\ \text{out} \leftarrow \mathcal{A}^f(\text{aux}, \pi) \end{array} \right\} .$$

2. The first-hybrid-world distribution:

$$\mathcal{D}_{H1} := \left\{ \text{out} \mid \begin{array}{l} f \leftarrow \mathcal{U}((\lambda, r)) \\ (x, w, \text{aux}) \leftarrow \mathcal{A}^f \\ (\pi, \mu) \leftarrow \mathcal{S}_{H1}^f(x, w) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\text{aux}, \pi) \end{array} \right\} \equiv \left\{ \text{out} \mid \begin{array}{l} f = (f_{\text{MT}}, f_{\$}) \leftarrow \mathcal{U}((\lambda, r)) \\ (x, w, \text{aux}) \leftarrow \mathcal{A}^f \\ \Pi \leftarrow \mathbf{P}_{\text{PCP}}(x, w) \\ (rt, td) \leftarrow MT.\text{Commit}^{f_{\text{MT}}}(\Pi) \\ \tau \leftarrow \{0, 1\}^s \\ \rho \leftarrow \{0, 1\}^r \\ Q := \text{queries of } \mathbf{V}_{\text{PCP}}^{\Pi}(x, \rho) \\ \mathbf{a} := \Pi[Q] \\ pf := MT.\text{Open}^{f_{\text{MT}}}(td, Q) \\ \mu := (\mu_{\text{MT}}, \mu_{\$}) : \\ \quad \bullet \mu_{\text{MT}} := \emptyset \\ \quad \bullet \mu_{\$} := \{((x, rt, \tau), \rho)\} \\ \pi := (rt, Q, \mathbf{a}, pf, \tau) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\text{aux}, \pi) \end{array} \right\} .$$

3. The second-hybrid-world distribution:

$$\mathcal{D}_{H2} := \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}((\lambda, r)) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ (\pi, \mu) \leftarrow \mathcal{S}_{H2}^f(\mathbf{x}, \mathbf{w}) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right. \right\} \equiv \left\{ \text{out} \left| \begin{array}{l} f = (f_{MT}, f_{\$}) \leftarrow \mathcal{U}((\lambda, r)) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ \Pi \leftarrow \mathbf{P}_{PCP}(\mathbf{x}, \mathbf{w}) \\ \tau \leftarrow \{0, 1\}^s \\ \rho \leftarrow \{0, 1\}^r \\ Q := \text{queries of } \mathbf{V}_{PCP}^\Pi(\mathbf{x}, \rho) \\ \mathbf{a} := \Pi[Q] \\ (\mathbf{rt}, \mathbf{pf}) \leftarrow MT.\text{Simulate}^{f_{MT}}(Q, \mathbf{a}) \\ \mu := (\mu_{MT}, \mu_{\$}) : \\ \quad \bullet \mu_{MT} := \emptyset \\ \quad \bullet \mu_{\$} := \{((\mathbf{x}, \mathbf{rt}, \tau), \rho)\} \\ \pi := (\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}, \tau) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right. \right\}.$$

4. The simulated-world distribution:

$$\mathcal{D}_{sim} := \left\{ \text{out} \left| \begin{array}{l} f \leftarrow \mathcal{U}((\lambda, r)) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ (\pi, \mu) \leftarrow \mathcal{S}^f(\mathbf{x}) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right. \right\} \equiv \left\{ \text{out} \left| \begin{array}{l} f = (f_{MT}, f_{\$}) \leftarrow \mathcal{U}((\lambda, r)) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ (\mathbf{x}, \rho, Q, \mathbf{a}) \leftarrow \mathbf{S}_{PCP}(\mathbf{x}) \\ (\mathbf{rt}, \mathbf{pf}) \leftarrow MT.\text{Simulate}^{f_{MT}}(Q, \mathbf{a}) \\ \tau \leftarrow \{0, 1\}^s \\ \mu := (\mu_{MT}, \mu_{\$}) : \\ \quad \bullet \mu_{MT} := \emptyset \\ \quad \bullet \mu_{\$} := \{((\mathbf{x}, \mathbf{rt}, \tau), \rho)\} \\ \pi := (\mathbf{rt}, Q, \mathbf{a}, \mathbf{pf}, \tau) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right. \right\}.$$

Next we analyze the statistical difference between these distributions.

- *Real versus first hybrid.* We analyze the statistical distance between \mathcal{D}_{real} and \mathcal{D}_{H1} .

The two distributions only differ in that the hybrid simulator \mathcal{S}_{H1} programs the oracle $f_{\$}$ by setting the answer to the query $(\mathbf{x}, \mathbf{rt}, \tau)$ to be a freshly sampled PCP randomness ρ (independent of f), whereas the argument prover \mathcal{P} does not program the oracle (it sets ρ to be the answer of $f_{\$}$ to the query $(\mathbf{x}, \mathbf{rt}, \tau)$).

The distribution of $((\mathbf{x}, \mathbf{rt}, \tau), \rho)$ is the same in both cases. However, \mathcal{A} has access to $f = (f_{MT}, f_{\$})$ before $f_{\$}$ is programmed. Thus, \mathcal{A} can distinguish between the two distributions *only* if \mathcal{A} queries $f_{\$}$ at $(\mathbf{x}, \mathbf{rt}, \tau)$ before $f_{\$}$ is programmed (and also after $f_{\$}$ is programmed).

The programmed query $(\mathbf{x}, \mathbf{rt}, \tau)$ is such that $\tau \in \{0, 1\}^s$ is sampled independently and uniformly at random. The probability that \mathcal{A} queries $(\mathbf{x}, \mathbf{rt}, \tau)$ before τ is sampled is at most $\frac{t}{2^s}$ (since \mathcal{A} makes at most t queries to f). In particular, this is an upper bound on the statistical distance between \mathcal{D}_{real} and \mathcal{D}_{H1} .

- *First hybrid versus second hybrid.* We analyze the statistical distance between \mathcal{D}_{H1} and \mathcal{D}_{H2} .

The two distributions only differ in that \mathcal{S}_{H1} samples the Merkle commitment using $MT.\text{Commit}$ and opening proof using $MT.\text{Open}$, and \mathcal{S}_{H2} samples a simulated Merkle commitment and opening proof using $MT.\text{Simulate}$.

The adversary \mathcal{A} is admissible, which implies that $|\mathbf{x}| \leq n$. By the hiding property of Merkle commitment schemes (Lemma 18.6.3), the statistical distance between \mathcal{D}_{H1} and \mathcal{D}_{H2} is upper bounded by

$$\max \left\{ z_{MT}(\lambda, l(\mathbf{x}), s, q(\mathbf{x}), t) \mid \mathbf{x}, \mathbf{w} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \text{ and } (\mathbf{x}, \mathbf{w}) \in \mathcal{R} \right\}.$$

Then, if the error bound z_{MT} is a monotonic function in the proof length and query complexity, we can upper bound the above expression by the following one:

$$z_{MT}(\lambda, l(n), s, q(n), t).$$

(As discussed in Remark 18.6.10, the error bound z_{MT} that we provide is monotonic.)

- *Second hybrid versus simulated.* We analyze the statistical distance between \mathcal{D}_{H2} and \mathcal{D}_{sim} . The two distributions only differ in that \mathcal{S}_{H2} samples a real PCP view and \mathcal{S} samples a simulated PCP view. By the zero-knowledge property of the PCP, the statistical distance between the PCP views for $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ is at most $z_{PCP}(\mathbf{x})$. Since \mathcal{A} outputs $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ with $|\mathbf{x}| \leq n$ (as \mathcal{A} is admissible), the statistical distance between \mathcal{D}_{H2} and \mathcal{D}_{sim} is upper bounded by $z_{PCP}(n)$.

Adding these error terms yields the statistical difference claimed in the lemma. \square

Part VI

SNARGs Based on IOPs

Overview

We show how to construct succinct arguments from a more general kind of probabilistic proof, known as interactive oracle proofs (IOPs). These probabilistic proofs simultaneously combine features of interactive proofs (multiple rounds) and probabilistically checkable proofs (few queries to a long proof string). IOPs are of significant practical interest because highly-efficient constructions are known, which lead to corresponding highly-efficient succinct arguments.

In this part we introduce interactive oracle proofs, and then describe how to construct succinct interactive arguments and then succinct non-interactive arguments. This involves combining many ideas introduced in earlier parts of this book.

23 Interactive oracle proofs	233
23.1 Definition	233
23.2 State restoration	237
24 Warmup: succinct interactive arguments from IOPs	239
24.1 Construction	239
24.2 Non-adaptive soundness	241
24.3 Adaptive soundness	244
25 The BCS transformation	248
25.1 Construction	248
25.2 Soundness	251
26 Additional security definitions	261
26.1 Knowledge soundness	261
26.2 Zero knowledge	268

23 Interactive oracle proofs

We introduce notations and definitions for *interactive oracle proofs* (IOPs), which are multi-round generalizations of PCPs.

In subsequent chapters we study how to construct non-interactive succinct arguments from IOPs. This includes the *iBCS transformation* in Chapter 24 and then, building on it, the *BCS transformation* in Chapters 25 and 26.

23.1 Definition

An interactive oracle proof (IOP) is a tuple of algorithms

$$\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$$

that works as follows. The IOP prover \mathbf{P}_{IOP} receives as input an instance \mathbf{x} and a witness w , and the IOP verifier \mathbf{V}_{IOP} receives as input the instance \mathbf{x} . They interact over some number k of rounds, where in each round $i \in [k]$ the IOP prover \mathbf{P}_{IOP} sends a proof string Π_i and then the IOP verifier \mathbf{V}_{IOP} sends a message β_i . The IOP verifier may query any of the received proof strings at any location. After the interaction, the IOP verifier \mathbf{V}_{IOP} outputs a bit denoting whether to accept or reject, computed based on the instance \mathbf{x} , the IOP verifier's own randomness, and the answers to any queries to the proof strings (these form the entire view of the IOP verifier). Both algorithms may be probabilistic. See Figure 23.1 for a diagram of an IOP.

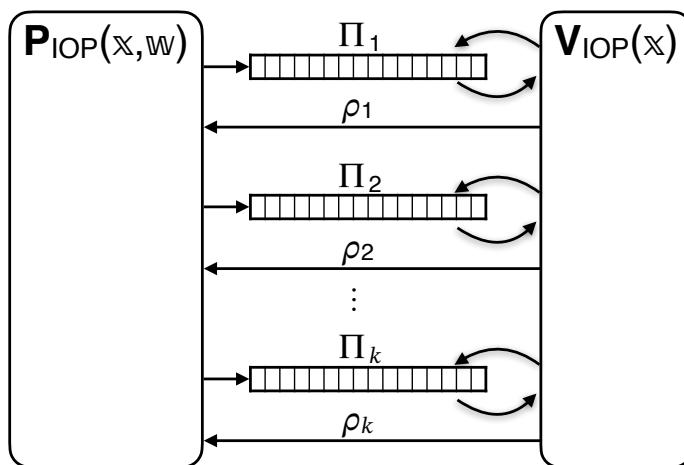


Figure 23.1: Diagram of an IOP.

The tuple $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ is an IOP for a relation \mathcal{R} with *(perfect) completeness* and *soundness error* ϵ_{IOP} if it satisfies the two properties stated below.

Definition 23.1.1. $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ for a relation \mathcal{R} has **perfect completeness** if for every $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$,

$$\Pr[\langle \mathbf{P}_{\text{IOP}}(\mathbf{x}, \mathbf{w}), \mathbf{V}_{\text{IOP}}(\mathbf{x}) \rangle_{\text{IOP}} = 1] = 1.$$

Definition 23.1.2. $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ for a relation \mathcal{R} has **soundness error** ϵ_{IOP} if for every $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ and malicious IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$,

$$\Pr[\langle \tilde{\mathbf{P}}_{\text{IOP}}, \mathbf{V}_{\text{IOP}}(\mathbf{x}) \rangle_{\text{IOP}} = 1] \leq \epsilon_{\text{IOP}}(\mathbf{x}).$$

We additionally define $\epsilon_{\text{IOP}}(n) := \max \{ \epsilon_{\text{IOP}}(\mathbf{x}) \mid \mathbf{x} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \text{ and } \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \}$.

Above, given two interactive algorithms A and B , we use the notation $\langle A, B \rangle_{\text{IOP}}$ to indicate the random variable that equals the output of B after interacting with A , and where the probability is taken over any randomness used by A or B .

Public-coin IOPs We only consider IOPs that are *public-coin*, which is a property of the IOP verifier \mathbf{V}_{IOP} (and is analogous to the one for the case of IPs in Definition 13.1.3).

Definition 23.1.3. $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ is **public-coin** if every message ρ_i sent by the IOP verifier \mathbf{V}_{IOP} is a random binary string of some prescribed size r_i (that is statistically independent of everything else); moreover, the IOP verifier \mathbf{V}_{IOP} has no other randomness. In this case, the decision bit of the IOP verifier \mathbf{V}_{IOP} is a function only of the instance \mathbf{x} , the IOP verifier randomness $\rho = (\rho_i)_{i \in [k]}$, and answers to queries to the received IOP strings $(\Pi_i)_{i \in [k]}$. We denote this bit by

$$\mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}).$$

In a public-coin IOP all queries by the IOP verifier can, without loss of generality, be postponed until after the interaction with the IOP prover. This is because the IOP verifier's messages to the IOP prover do not depend on the answer to any query. Therefore, a public-coin IOP can be viewed as having two phases: (i) the *interaction phase*, where the IOP prover and IOP verifier exchange messages, and the IOP verifier does not make any queries; and then (ii) the *query phase*, where the IOP verifier queries the received proof strings and then outputs a decision bit, based on the instance \mathbf{x} , the IOP verifier randomness $\rho = (\rho_i)_{i \in [k]}$, and answers to its queries.

Efficiency measures We are interested in several efficiency measures of a IOP.

- *Round complexity*, denoted k , is the number of back-and-forth interactions between the prover and verifier.
- *Alphabet*, denoted Σ , is the alphabet used to write proof strings. (In general, each proof string may be over a different alphabet, or multiple alphabets, but for simplicity we focus on one alphabet for all proof strings.)
- *Proof length*, denoted l , is the number of symbols across all proof strings. The total number of bits across all proof strings is thus $l \cdot \log |\Sigma|$. We denote by l_i the number of symbols in the proof string in round $i \in [k]$. Hence $l = \sum_{i \in [k]} l_i$.
- *Randomness complexity*, denoted r , is the number of random bits used by the verifier across all rounds. We denote by r_i the randomness complexity in round $i \in [k]$. Hence $r = \sum_{i \in [k]} r_i$.

- *Query complexity*, denoted q , is the number of locations in all proof strings read by the verifier. (Hence each query returns a symbol of $\log |\Sigma|$ bits.) We denote by q_i the query complexity to the proof string in round $i \in [k]$. Hence $q = \sum_{i \in [k]} q_i$.
- *Prover time and verifier time*, denoted pt and vt , are the time complexities of the prover (to output the IOP strings) and verifier (to output its decision bit).

The above efficiency measures may be functions of the instance \mathbf{x} (and possibly other parameters associated to the construction of an IOP).

Non-oracle messages In some IOP constructions it is useful to allow the IOP prover to send non-oracle messages: in each round $i \in [k]$ of an IOP, the IOP prover may send, in addition to the proof string Π_i , a non-oracle message α_i that the IOP verifier reads in its entirety. Thus, in the case of a public-coin IOP, the decision of the IOP verifier would then be computed as

$$\mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}).$$

One could view the non-oracle message α_i to be a segment of the IOP string Π_i that the IOP verifier always queries, but this view incurs minor overheads in constructions of succinct arguments based on IOPs. These overheads can be avoided via simple modifications to the construction that directly support non-oracle messages; we discuss these in Remarks 24.1.2 and 25.1.2.

Knowledge soundness The soundness notion (Definition 23.1.2) can be strengthened to knowledge soundness notions that are analogous to those in the case of IPs in Section 13.1. Informally, whenever an IOP prover convinces the IOP verifier, then an efficient extractor finds a witness (up to some error), given suitable information about the IOP prover.

Fix any instance \mathbf{x} and IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$. The IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ and IOP verifier \mathbf{V}_{IOP} (on input \mathbf{x}) interact; denote by $((\Pi_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ the transcript of this interaction and by b the decision bit of the IOP verifier \mathbf{V}_{IOP} . The knowledge extractor \mathbf{E}_{IOP} is tasked to find a witness \mathbf{w} when given as input the instance \mathbf{x} , the interaction transcript $((\Pi_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$, and black-box access to the IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$. This means that the knowledge extractor \mathbf{E}_{IOP} is *rewinding*: it can re-run the IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ multiple times, possibly with correlated inputs. The knowledge soundness error is an upper bound on the probability that $b = 1$ ($\tilde{\mathbf{P}}_{\text{IOP}}$ convinces \mathbf{V}_{IOP}) and $(\mathbf{x}, \mathbf{w}) \notin \mathcal{R}$ (\mathbf{E}_{IOP} does not find a witness). In general, the knowledge soundness error may be a function of the failure probability of $\tilde{\mathbf{P}}_{\text{IOP}}$, which we define below.

Definition 23.1.4. Let $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ be an IOP. A deterministic IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ has failure probability $\delta_{\tilde{\mathbf{P}}_{\text{IOP}}}$ if for every instance \mathbf{x} :

$$\Pr[\langle \tilde{\mathbf{P}}_{\text{IOP}}, \mathbf{V}_{\text{IOP}}(\mathbf{x}) \rangle_{\text{IOP}} = 0] \leq \delta_{\tilde{\mathbf{P}}_{\text{IOP}}}(\mathbf{x}).$$

Definition 23.1.5. $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ for a relation \mathcal{R} has **rewinding knowledge soundness error** κ_{IOP} with extraction time et_{IOP} if there exists a probabilistic algorithm \mathbf{E}_{IOP} (the extractor) such that for every instance \mathbf{x} and deterministic IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ with running time $\tau_{\tilde{\mathbf{P}}_{\text{IOP}}}$ the following holds:

$$\Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} b \xleftarrow{((\Pi_i)_{i \in [k]}, (\rho_i)_{i \in [k]})} \langle \tilde{\mathbf{P}}_{\text{IOP}}, \mathbf{V}_{\text{IOP}}(\mathbf{x}) \rangle_{\text{IOP}} \\ \mathbf{w} \xleftarrow{\mathbf{E}_{\text{IOP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IOP}})} \end{array} \right] \leq \kappa_{\text{IOP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{IOP}}}(\mathbf{x})).$$

Moreover, $\mathbf{E}_{\text{IOP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IOP}})$ runs in expected time $\mathbf{et}_{\text{IOP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{IOP}}}(\mathbf{x}), \tau_{\tilde{\mathbf{P}}_{\text{IOP}}}(\mathbf{x}))$. We additionally define

$$\begin{aligned}\kappa_{\text{IOP}}(n, \delta_{\tilde{\mathbf{P}}_{\text{IOP}}}) &:= \max \left\{ \kappa_{\text{IOP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{IOP}}}(\mathbf{x})) \mid \mathbf{x} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \right\}, \quad \text{and} \\ \mathbf{et}_{\text{IOP}}(n, \delta_{\tilde{\mathbf{P}}_{\text{IOP}}}, \tau_{\tilde{\mathbf{P}}_{\text{IOP}}}) &:= \max \left\{ \mathbf{et}_{\text{IOP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{IOP}}}(\mathbf{x}), \tau_{\tilde{\mathbf{P}}_{\text{IOP}}}(\mathbf{x})) \mid \mathbf{x} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \right\}.\end{aligned}$$

If $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ has knowledge soundness error κ_{IOP} (with any extraction time \mathbf{et}_{IOP}) then it has soundness error ϵ_{IOP} such that $\epsilon_{\text{IOP}}(\mathbf{x}) \leq \min_{\delta \in [0, 1]} \kappa_{\text{IOP}}(\mathbf{x}, \delta)$: if $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ then the extractor cannot output a witness \mathbf{w} such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ (as none exists), in which case the event bounded by the knowledge soundness condition equals the event bounded by the soundness condition.

A notable special case of Definition 23.1.5 is *straightline extraction*: \mathbf{E}_{IOP} is a polynomial-time deterministic algorithm that receives as input the instance \mathbf{x} and IOP strings $(\Pi_i)_{i \in [k]}$ (but not the IOP verifier messages $(\rho_i)_{i \in [k]}$ or access to $\tilde{\mathbf{P}}_{\text{IOP}}$).

Definition 23.1.6. $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ for a relation \mathcal{R} has **straightline knowledge soundness error** κ_{IOP} if there exists a polynomial-time deterministic algorithm \mathbf{E}_{IOP} (the extractor) such that for every deterministic IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ and instance \mathbf{x} :

$$\Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} b \xleftarrow{((\Pi_i)_{i \in [k]}, (\rho_i)_{i \in [k]})} \langle \tilde{\mathbf{P}}_{\text{IOP}}, \mathbf{V}_{\text{IOP}}(\mathbf{x}) \rangle_{\text{IOP}} \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IOP}}(\mathbf{x}, (\Pi_i)_{i \in [k]}) \end{array} \right] \leq \kappa_{\text{IOP}}(\mathbf{x}).$$

Zero knowledge A notion of zero knowledge for IOPs against honest IOP verifiers suffices for the applications that we study. Informally, zero knowledge states that the view of the verifier can be efficiently simulated by a probabilistic algorithm that only has access to the instance. In the case of a IOP the view consists of the instance, the randomness, and the query answers from the given IOP string. Honest-verifier zero knowledge then requires that the simulated view is statistically close to a view in a real execution, provided that the proved statement is true.

Definition 23.1.7. The IOP verifier's **view** in $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ on the instance-witness pair (\mathbf{x}, \mathbf{w}) , denoted $\mathbf{View}_{\text{IOP}}(\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}}, \mathbf{x}, \mathbf{w})$, is the random variable $(\mathbf{x}, \rho, (Q_i)_{i \in [k]}, (\mathbf{a}_i)_{i \in [k]})$ where:

- $\rho \in \{0, 1\}^r$ is a random choice of randomness for the IOP verifier \mathbf{V}_{IOP} ;
- $(Q_i \subseteq [l_i])_{i \in [k]}$ and $(\mathbf{a}_i \in \Sigma^{Q_i})_{i \in [k]}$ are the queries and answers made across the k IOP strings in an interaction between $\mathbf{P}_{\text{IOP}}(\mathbf{x}, \mathbf{w})$ and $\mathbf{V}_{\text{IOP}}(\mathbf{x}, \rho)$.

The IOP prover $\mathbf{P}_{\text{IOP}}(\mathbf{x}, \mathbf{w})$ may use private randomness (not part of the IOP verifier's view). If the IOP is public-coin then the view shows each round's randomness: $(\mathbf{x}, (\rho_i)_{i \in [k]}, (Q_i)_{i \in [k]}, (\mathbf{a}_i)_{i \in [k]})$.

Definition 23.1.8. $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ for a relation \mathcal{R} has **honest-verifier zero-knowledge error** z_{IOP} if there exists a polynomial-time probabilistic algorithm \mathbf{S}_{IOP} such that for every instance-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ the following random variables are $z_{\text{IOP}}(\mathbf{x})$ -close in statistical distance:

$$\mathbf{View}_{\text{IOP}}(\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}}, \mathbf{x}, \mathbf{w}) \text{ and } \mathbf{S}_{\text{IOP}}(\mathbf{x}).$$

We additionally define $z_{\text{IOP}}(n) := \max \left\{ z_{\text{IOP}}(\mathbf{x}) \mid \mathbf{x}, \mathbf{w} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \text{ and } (\mathbf{x}, \mathbf{w}) \in \mathcal{R} \right\}$.

Verification on local views We sometimes run an IOP verifier with query access to *partial* IOP strings defined on a subset of locations, and it is useful to introduce notation for this setting.

Definition 23.1.9. Let $(Q_i \subseteq [l_i])_{i \in [k]}$ be query sets, $(\mathbf{a}_i \in \Sigma^{Q_i})_{i \in [k]}$ query answers, \mathbf{x} an instance, and $\rho = (\rho_i)_{i \in [k]} \in \{0, 1\}^r$ a choice of IOP randomness (split according to rounds). We denote by

$$\mathbf{V}_{\text{IOP}}^{[Q_i, \mathbf{a}_i]_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]})$$

the decision bit of the IOP verifier \mathbf{V}_{IOP} , given instance \mathbf{x} and IOP randomness $\rho = (\rho_i)_{i \in [k]}$, when each query $j \in Q_i$ is answered with $\mathbf{a}_i[j] \in \Sigma$. (If \mathbf{V}_{IOP} queries outside the sets $(Q_i)_{i \in [k]}$ then $\mathbf{V}_{\text{IOP}}^{[Q_i, \mathbf{a}_i]_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 0$.)

23.2 State restoration

We state definitions for state-restoration soundness for public-coin IOPs. These definitions equal the definitions for public-coin IPs in Section 13.2.1 except for straightforward syntactical changes:

- IP prover messages α_i are replaced by IOP strings Π_i ; and
- the IP acceptance condition $\mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 1$ is replaced by the IOP acceptance condition $\mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1$.

While the differences are minor, we state the definitions for the case of IOPs for the sake of completeness. We refer the reader to the discussion for IPs for motivation and intuition.

Note that the discussion in Section 13.2.2, which compares the notions of standard soundness and state-restoration soundness for public-coin IPs, directly carries over to the case public-coin IOPs. In particular, the security of a non-interactive argument obtained from a given IOP via the BCS transformation is characterized by the state-restoration soundness of the IOP, as we discuss in Chapter 25. Therefore, to ensure security of the BCS transformation, one must assume that the given IOP has small state-restoration soundness error.

As for IPs, small state-restoration soundness error for an IOP is implied by natural strong notions of soundness, such as special soundness (see Chapter 30) and round-by-round soundness (see Chapter 31).

Definition 23.2.1. The **IOP state-restoration game** for $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ with salt size $s \in \mathbb{N}$, functions $\text{rnd} = (\text{rnd}_i)_{i \in [k]} \in \mathcal{U}((\mathbf{r}_i)_{i \in [k]})$, and IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ is defined below.

$\text{Game}_{\text{IOP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}})$:

1. Repeat the following until $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ decides to exit the loop.
 - $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ outputs $(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i))$, where \mathbf{x} is an instance, (Π_1, \dots, Π_i) are IOP strings, and $(\sigma_1, \dots, \sigma_i)$ are salt strings in $\{0, 1\}^s$.
 - Set $\rho_i := \text{rnd}_i(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i))$.
 - Send ρ_i to $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$.
2. $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ outputs $(\mathbf{x}, (\Pi_1, \dots, \Pi_k), (\sigma_1, \dots, \sigma_k))$, where \mathbf{x} is an instance, (Π_1, \dots, Π_k) are IOP strings, and $(\sigma_1, \dots, \sigma_k)$ are salt strings in $\{0, 1\}^s$.
3. For every $i \in [k]$, set $\rho_i := \text{rnd}_i(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i))$.
4. Output $(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$.

We denote by tr^{sr} the list of move-response pairs of the form $((\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i)), \rho_i)$ performed in the loop. We show tr^{sr} in an execution of the IOP state-restoration game using the following notation:

$$(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \text{Game}_{\text{IOP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}).$$

We say that $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ is **t -move** if $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ exits the loop after at most t iterations.

Definition 23.2.2. IOP = $(\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ has **state-restoration soundness error** $\epsilon_{\text{IOP}}^{\text{sr}}$ if for every salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move malicious IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$, and instance size bound $n \in \mathbb{N}$:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \end{array} \middle| \begin{array}{l} \text{rnd} = (\text{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \leftarrow \\ \text{Game}_{\text{IOP}}^{\text{sr}}(s, \text{rnd}, \mathbf{P}_{\text{IOP}}^{\text{sr}}) \end{array} \right] \leq \epsilon_{\text{IOP}}^{\text{sr}}(s, t, n).$$

Definition 23.2.3. IOP = $(\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ has **straightline state-restoration knowledge soundness error** $\kappa_{\text{IOP}}^{\text{sr}}$ if there exists a polynomial-time deterministic algorithm $\mathbf{E}_{\text{IOP}}^{\text{sr}}$ (the extractor) such that for every salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move deterministic IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$, and instance size bound n :

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \end{array} \middle| \begin{array}{l} \text{rnd} = (\text{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \\ \text{Game}_{\text{IOP}}^{\text{sr}}(s, \text{rnd}, \mathbf{P}_{\text{IOP}}^{\text{sr}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IOP}}^{\text{sr}}(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, \text{tr}^{\text{sr}}) \end{array} \right] \leq \kappa_{\text{IOP}}^{\text{sr}}(s, t, n).$$

Definition 23.2.4. Let IOP = $(\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ be an IOP. A deterministic IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ has **failure probability** $\delta_{\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}}$ if for every salt size $s \in \mathbb{N}$ and instance size bound n :

$$\Pr \left[\begin{array}{l} |\mathbf{x}| > n \\ \vee \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 0 \end{array} \middle| \begin{array}{l} \text{rnd} = (\text{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \leftarrow \\ \text{Game}_{\text{IOP}}^{\text{sr}}(s, \text{rnd}, \mathbf{P}_{\text{IOP}}^{\text{sr}}) \end{array} \right] \leq \delta_{\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}}(s, n).$$

Definition 23.2.5. IOP = $(\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ has **rewinding state-restoration knowledge soundness error** $\kappa_{\text{IOP}}^{\text{sr}}$ with **extraction time** $\mathbf{et}_{\text{IOP}}^{\text{sr}}$ if there exists a probabilistic algorithm $\mathbf{E}_{\text{IOP}}^{\text{sr}}$ (the extractor) such that for every salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move deterministic IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ with failure probability $\delta_{\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}}$ and running time $\tau_{\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}}$, and instance size bound n :

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \end{array} \middle| \begin{array}{l} \text{rnd} = (\text{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \\ \text{Game}_{\text{IOP}}^{\text{sr}}(s, \text{rnd}, \mathbf{P}_{\text{IOP}}^{\text{sr}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IOP}}^{\text{sr}}(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \text{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}) \end{array} \right] \leq \kappa_{\text{IOP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}}(s, n)).$$

Moreover, $\mathbf{E}_{\text{IOP}}^{\text{sr}}$ runs in expected time $\mathbf{et}_{\text{IOP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}}(s, n), \tau_{\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}}(s, n))$ (over the given inputs and internal randomness).

24 Warmup: succinct interactive arguments from IOPs

We describe how to construct a succinct *interactive* argument, starting from any public-coin interactive oracle proof (IOP); later, in Chapter 25, we describe how to additionally achieve a succinct *non-interactive* argument. The non-interactive argument in Chapter 25 is known as the Ben-Sasson–Chiesa–Schnorr (BCS) transformation, and the interactive argument in this chapter is a simplification of it known as the *interactive BCS (iBCS) transformation*. Throughout, we assume familiarity with the Merkle commitment scheme as introduced in Chapter 18.

24.1 Construction

The IOP model envisages a setting where the IOP verifier interacts with the IOP prover and has query access to the IOP prover’s messages. Similarly to the Kilian transformation for PCPs (see Chapter 20), the succinct interactive argument that we describe here relies on the Merkle commitment scheme to “implement” this query access interface. While this approach works for a more general class of IOPs, here we describe and analyze the case for public-coin IOPs.

The interactive phase of the public-coin IOP is mapped to a corresponding phase of the interactive argument: for each round $i \in [k]$, the argument prover uses the IOP prover to generate the i -th proof string Π_i , uses the Merkle commitment scheme to generate a short commitment rt_i to Π_i , and sends rt_i to the argument verifier; then the argument verifier sends the IOP verifier’s (random) i -th message ρ_i to the argument prover.

Next, the query phase of the public-coin IOP is mapped to a final message: the argument prover, having now received all IOP verifier random messages, simulates the IOP verifier in order to determine its queries across all IOP strings, and then sends a message that includes the queried locations, the query answers, and Merkle opening proofs that authenticate the answers for those locations (relative to the Merkle commitments sent earlier).

Finally, the argument verifier checks that the IOP verifier accepts (given the instance, its randomness, and answers to the queries) and that all Merkle opening proofs are valid.

Below we describe the construction in detail.

Construction 24.1.1: iBCS transformation

Let $IOP = (\mathbf{P}_{IOP}, \mathbf{V}_{IOP})$ be a public-coin IOP with round complexity k , proof length l over alphabet Σ , query complexity q , and randomness complexity r . Let $\lambda \in \mathbb{N}$ be a security parameter and $s \in \mathbb{N}$ be a privacy parameter. Define $MT := MT[\lambda, \Sigma, l, s]$ to be the Merkle commitment scheme with the stated parameters. (The output size of the

random oracle for the Merkle commitment scheme equals the security parameter λ .) We define $\mathbf{IARG} := \mathbf{iBCS}[\mathbf{IOP}, \lambda, s]$ to be the public-coin interactive argument $\mathbf{IARG} = (\mathcal{P}, \mathcal{V})$ constructed as follows. The argument prover \mathcal{P} receives as input an instance \mathbf{x} and witness \mathbf{w} , and the argument verifier \mathcal{V} receives as input the instance \mathbf{x} . Both receive query access to a random oracle $f \in \mathcal{U}(\lambda)$ and they interact as below.

1. For $i = 1, \dots, k$:

- *Prover message.* The argument prover \mathcal{P} computes its i -th message as follows.

- Compute the i -th proof string (and auxiliary state) of the IOP prover:

$$(\Pi_i \in \Sigma^{l_i}, \mathbf{aux}_i) := \begin{cases} \mathbf{P}_{\mathbf{IOP}}(\mathbf{x}, \mathbf{w}) & \text{if } i = 1 \\ \mathbf{P}_{\mathbf{IOP}}(\mathbf{aux}_{i-1}, \rho_{i-1}) & \text{if } i > 1 \end{cases}.$$

- Commit to Π_i via the Merkle commitment scheme: $(\mathbf{rt}_i, \mathbf{td}_i) := \mathbf{MT}.\mathbf{Commit}^f(\Pi_i)$.
- Send the i -th Merkle commitment $\mathbf{rt}_i \in \{0, 1\}^\lambda$.

- *Verifier message.* The argument verifier \mathcal{V} sends the i -th random message $\rho_i \in \{0, 1\}^{r_i}$ for the IOP verifier $\mathbf{V}_{\mathbf{IOP}}$.

2. *Answer queries.* The argument prover \mathcal{P} does the following.

- Simulate the IOP verifier $\mathbf{V}_{\mathbf{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]})$, yielding query sets $(Q_i \subseteq [l_i])_{i \in [k]}$.
- For every $i \in [k]$, set $\mathbf{a}_i := \Pi_i[Q_i] \in \Sigma^{Q_i}$ to be the answers for the i -th query set.
- For every $i \in [k]$, compute an opening proof for \mathbf{a}_i : $\mathbf{pf}_i := \mathbf{MT}.\mathbf{Open}^f(\mathbf{td}_i, Q_i)$.
- Send $((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]}$.

3. *Verification.* The argument verifier \mathcal{V} checks the following.

- Check that $\mathbf{V}_{\mathbf{IOP}}^{(Q_i, \mathbf{a}_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1$. (The IOP verifier $\mathbf{V}_{\mathbf{IOP}}(\mathbf{x}, (\rho_i)_{i \in [k]})$ accepts if each query $j \in Q_i$ is answered with $\mathbf{a}_i[j] \in \Sigma$. If any query falls outside the sets $(Q_i)_{i \in [k]}$ then reject.)
- Check that $\wedge_{i \in [k]} \mathbf{MT}.\mathbf{Check}^f(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) = 1$. (The query answers are authenticated with respect to the Merkle commitments $(\mathbf{rt}_i)_{i \in [k]}$.)

Efficiency We discuss efficiency properties of the above construction. Recall that $l = \sum_{i \in [k]} l_i$ and $q = \sum_{i \in [k]} q_i$, where l_i and q_i are the proof length and the query complexity for the i -th proof string in the IOP, respectively.

- *Communication complexity.* The argument prover sends to the argument verifier:
 - k Merkle commitments, which amounts to $k \cdot \lambda$ bits;
 - for each $i \in [k]$, q_i query-answer pairs in $[l_i] \times \Sigma$, which amounts to $\sum_{i \in [k]} q_i \cdot (\log l_i + \log |\Sigma|) \leq q \cdot (\log l + \log |\Sigma|)$ bits; and
 - for each $i \in [k]$, a Merkle opening proof for q_i queries in $[l_i]$, which amounts to

$$\sum_{i \in [k]} q_i \cdot (s + \lambda \cdot \log l_i) \leq q \cdot (s + \lambda \cdot \log l) \text{ bits.}^1$$

Hence the number of bits sent by the argument prover is at most:

$$k \cdot \lambda + q \cdot (\log l + \log |\Sigma| + s + \lambda \cdot \log l) = O(k \cdot \lambda + q \cdot (\log |\Sigma| + s + \lambda \cdot \log l)). \quad (24.1)$$

In the other direction, the argument verifier sends to the argument prover $r = \sum_{i \in [k]} r_i$ random bits. If zero knowledge is not a goal, then the privacy parameter s equals 0.

Simply sending the (uncommitted) IOP strings $(\Pi_i)_{i \in [k]}$ (across an interaction) would have cost $\sum_{i \in [k]} l_i \cdot \log |\Sigma| = l \cdot \log |\Sigma|$ bits. Therefore, using the Merkle commitment scheme to succinctly commit and then locally open the queried locations reduces the communication complexity to $q \cdot (\log l + \log |\Sigma|)$ plus data used to commit and authenticate. (For most parameter settings, the precise expression mentioned above is significantly smaller than the number of bits in the IOP string.)

- *Prover complexity.* The complexity of the argument prover is typically dominated by the complexity of the underlying IOP prover. Moreover, the number of queries to the random oracle is $q_P = \sum_{i \in [k]} O(l_i) = O(l)$; this is because the argument prover commits to each proof string $\Pi_i \in \Sigma^{l_i}$ in $O(l_i)$ queries (via MT.Commit), and $l = \sum_{i \in [k]} l_i$ (this is the definition of l).
- *Verifier complexity.* The complexity of the argument verifier is typically dominated by the complexity of the underlying IOP verifier. Moreover, the number of queries to the random oracle is $q_V = \sum_{i \in [k]} O(q_i \cdot \log l_i) = O(q \cdot \log l)$; this is because the argument verifier checks each opening proof pf_i in $O(q_i \cdot \log l_i)$ queries (via MT.Check), $q = \sum_{i \in [k]} q_i$ (this is the definition of q), and $l_i \leq l$.

Remark 24.1.2 (non-oracle messages in Construction 24.1.1). We discuss the modifications needed to additionally support non-oracle messages $(\alpha_i)_{i \in [k]}$ from the IOP prover.

- In Item 1c, the argument prover \mathcal{P} can additionally send a non-oracle message α_i (if any is output by \mathbf{P}_{IOP}), in which case the i -th argument prover message is (\mathbf{r}_i, α_i) .
- In Item 2a of the argument prover \mathcal{P} , the IOP verifier is run as $\mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$.
- In Item 3a of the argument verifier \mathcal{V} , the explicit input to the IOP verifier is

$$(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}).$$

24.2 Non-adaptive soundness

We prove that Construction 24.1.1 is *non-adaptively* sound (Definition 4.2.2): for any given instance that is not in the language, the probability that a bounded-query adversary convinces the argument verifier to accept is small. Theorem 24.2.1 below provides an upper bound on this error, which is the soundness error of the underlying IOP plus a small term. This is analogous to the case of PCPs, in Theorem 20.2.1 (in Section 20.2). Later in Section 24.3 we prove that Construction 24.1.1 is also *adaptively* sound.

¹See Section 29.2 for more on the size of Merkle opening proofs.

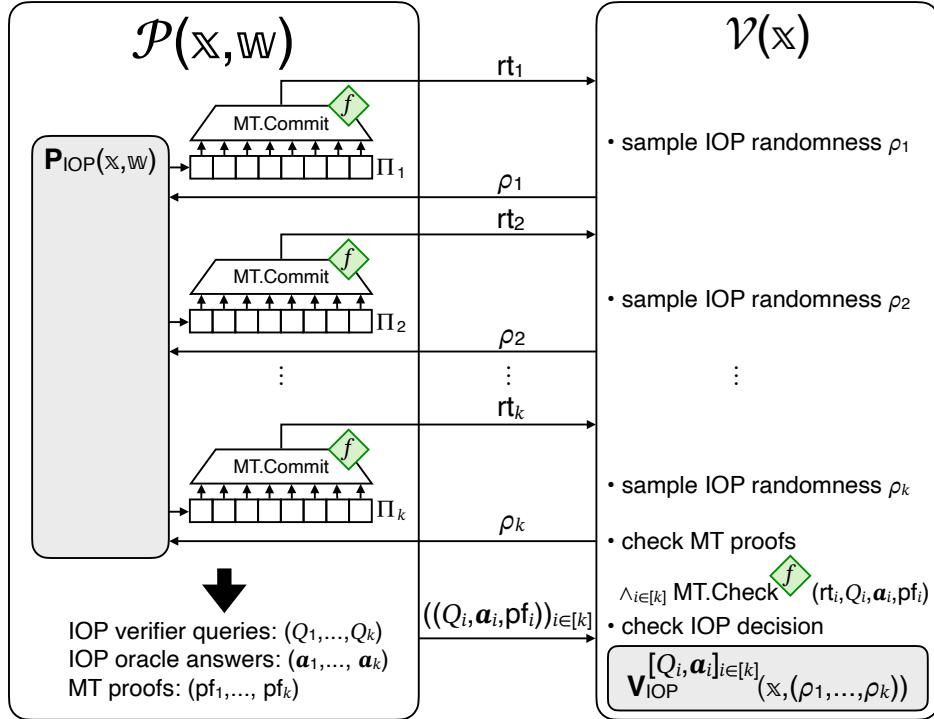


Figure 24.1: Diagram of the iBCS transformation (iBCS in Construction 24.1.1).

Theorem 24.2.1

Let IOP be a public-coin IOP for a relation \mathcal{R} with soundness error ϵ_{IOP} , round complexity k , and proof length l . For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{IARG} := \text{iBCS}[\text{IOP}, \lambda, s]$ in Construction 24.1.1 is an interactive argument for \mathcal{R} with non-adaptive soundness error ϵ_{ARG} (see Definition 4.2.2) such that

$$\epsilon_{\text{ARG}}(\lambda, t, \mathbf{x}) \leq \epsilon_{\text{IOP}}(\mathbf{x}) + \kappa_{\text{MT}}(\lambda, t, l, k, k).$$

Above κ_{MT} is the MT multi-extraction error from Lemma 18.5.6, and $\kappa_{\text{MT}}(\lambda, t, l, k, k) \leq 2 \cdot \frac{t^2}{2^\lambda}$ if $6 \cdot k \cdot (\log l + 1) \leq t$.

We prove the theorem via a security reduction of the following form. For every fixed instance \mathbf{x} , any malicious argument prover $\tilde{\mathcal{P}}$ that convinces the argument verifier $\mathcal{V}(\mathbf{x})$ with probability δ can be transformed into a corresponding malicious IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ that convinces the IOP verifier $\mathbf{V}_{\text{IOP}}(\mathbf{x})$ with probability at least δ minus a small term. Theorem 24.2.1 follows by noting that if $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$ then any IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ convinces $\mathbf{V}_{\text{IOP}}(\mathbf{x})$ with probability at most $\epsilon_{\text{IOP}}(\mathbf{x})$.

Below we state the security reduction and then prove it.

Lemma 24.2.2. *There exists an IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ such that for every t -query argument prover*

$\tilde{\mathcal{P}}$ the following holds for every instance \mathbf{x} :

$$\Pr \left[\langle \tilde{\mathcal{P}}^f, \mathcal{V}^f(\mathbf{x}) \rangle_{\text{ARG}} = 1 \mid f \leftarrow \mathcal{U}(\lambda) \right] \leq \Pr \left[\langle \tilde{\mathbf{P}}_{\text{IOP}}(\mathcal{P}), \mathbf{V}_{\text{IOP}}(\mathbf{x}) \rangle_{\text{IOP}} = 1 \right] + \kappa_{\text{MT}}(\lambda, t, l, k, k).$$

Moreover, the running time of $\tilde{\mathbf{P}}_{\text{IOP}}$ equals the running time of $\tilde{\mathcal{P}}$ plus $\text{et}_{\text{MT}}(\lambda, \Sigma, l, s, t, k)$ where et_{MT} is the time complexity of the MT extractor in Lemma 18.5.6.

Proof. Let $\text{MT}.\text{MultiExtract}$ be the MT extractor for multiple commitments from Lemma 18.5.6 in Section 18.5.2. The IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ works as follows:

1. Start running the argument prover $\tilde{\mathcal{P}}$, simulating its random oracle f .
2. For $i = 1, \dots, k$:
 - a) When $\tilde{\mathcal{P}}$ outputs the i -th Merkle commitment \mathbf{rt}_i , let \mathbf{tr}_i be the query-answer trace of this partial execution since outputting \mathbf{rt}_{i-1} (or the beginning if $i = 1$).
 - b) Run the MT extractor to obtain a proof string: $(\Pi_i, \mathbf{td}_i) \leftarrow \text{MT}.\text{MultiExtract}(\mathbf{rt}_i, \mathbf{tr}_i)$. ($\text{MT}.\text{MultiExtract}$ also outputs an opening trapdoor \mathbf{td}_i , but we do not use that here.)
 - c) Send the IOP string Π_i to the IOP verifier.
 - d) Receive the IOP random message $\rho_i \in \{0, 1\}^{r_i}$, and forward it to $\tilde{\mathcal{P}}$.

The IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ uses the argument prover $\tilde{\mathcal{P}}$ as a black box. The running time of $\tilde{\mathbf{P}}_{\text{IOP}}$ is the running time of (a partial execution of) $\tilde{\mathcal{P}}$ plus $\text{et}_{\text{MT}}(\lambda, \Sigma, l, s, t, k)$; indeed, the MT extractor $\text{MT}.\text{MultiExtract}$ is invoked against an adversary that makes at most t queries to the random oracle and outputs k Merkle commitments across its execution.

We now argue that the IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ satisfies the claimed property.

By construction, the argument verifier \mathcal{V} accepts if and only if the IOP verifier \mathbf{V}_{IOP} accepts and each invocation of the Merkle commitment checking algorithm $\text{MT}.\text{Check}$ accepts:

$$\Pr \left[\langle \tilde{\mathcal{P}}^f, \mathcal{V}^f(\mathbf{x}) \rangle_{\text{ARG}} = 1 \mid f \leftarrow \mathcal{U}(\lambda) \right] = \Pr \left[\begin{array}{l} \mathbf{V}_{\text{IOP}}^{[Q_i, \mathbf{a}_i]_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \\ \wedge \wedge_{i \in [k]} \text{MT}.\text{Check}^f(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) = 1 \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \text{For } i = 1, \dots, k: \\ \quad \cdot (\mathbf{rt}_i, \mathbf{aux}_i) \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}_{i-1}, \rho_{i-1}) \\ \quad \cdot \rho_i \leftarrow \{0, 1\}^{r_i} \\ \quad ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]} \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}_k, \rho_k) \end{array} \right].$$

We split the above probability as a sum of two probabilities that separately consider the disjoint cases where all extractions succeed or at least one extraction fails:

$$\Pr \left[\begin{array}{l} \mathbf{V}_{\text{IOP}}^{[Q_i, \mathbf{a}_i]_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \\ \wedge \wedge_{i \in [k]} \text{MT}.\text{Check}^f(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) = 1 \\ \wedge \wedge_{i \in [k]} \Pi_i[Q_i] = \mathbf{a}_i \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \text{For } i = 1, \dots, k: \\ \quad \cdot (\mathbf{rt}_i, \mathbf{aux}_i) \xleftarrow{\mathbf{tr}_i} \tilde{\mathcal{P}}^f(\mathbf{aux}_{i-1}, \rho_{i-1}) \\ \quad \cdot (\Pi_i, \mathbf{td}_i) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}_i, \mathbf{tr}_i) \\ \quad \cdot \rho_i \leftarrow \{0, 1\}^{r_i} \\ \quad ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]} \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}_k, \rho_k) \end{array} \right] \quad (24.2)$$

$$+ \Pr \left[\begin{array}{l} \mathbf{V}_{\text{IOP}}^{[Q_i, \mathbf{a}_i]_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \\ \wedge \wedge_{i \in [k]} \text{MT}.\text{Check}^f(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) = 1 \\ \wedge \vee_{i \in [k]} \Pi_i[Q_i] \neq \mathbf{a}_i \end{array} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \text{For } i = 1, \dots, k: \\ \quad \cdot (\mathbf{rt}_i, \mathbf{aux}_i) \xleftarrow{\mathbf{tr}_i} \tilde{\mathcal{P}}^f(\mathbf{aux}_{i-1}, \rho_{i-1}) \\ \quad \cdot (\Pi_i, \mathbf{td}_i) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}_i, \mathbf{tr}_i) \\ \quad \cdot \rho_i \leftarrow \{0, 1\}^{r_i} \\ \quad ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]} \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}_k, \rho_k) \end{array} \right]. \quad (24.3)$$

We conclude the proof by upper bounding each of the two terms.

- The term in Equation 24.2 is upper bounded by the IOP error, as we now explain. If $\wedge_{i \in [k]} \Pi_i[Q_i] = \mathbf{a}_i$ (every extracted IOP string agrees with the relevant query answers) then $\mathbf{V}_{\text{iop}}^{[Q_i, \mathbf{a}_i]_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = \mathbf{V}_{\text{iop}}^{(\Pi)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]})$ (the IOP verifier returns the same output). Moreover, the probability does not decrease if we omit the conditions on $\text{MT}.\text{Check}$ (and thus can ignore the final output of $\tilde{\mathcal{P}}$). Therefore the term in Equation 24.2 can be upper bounded by the following expression:

$$\Pr \left[\mathbf{V}_{\text{iop}}^{(\Pi)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \mid \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \text{For } i = 1, \dots, k: \\ \quad \cdot (\mathbf{r}_i, \mathbf{aux}_i) \xleftarrow{\text{tr}_i} \tilde{\mathcal{P}}^f(\mathbf{aux}_{i-1}, \rho_{i-1}) \\ \quad \cdot (\Pi_i, \mathbf{td}_i) \leftarrow \text{MT}.\text{Extract}(\mathbf{r}_i, \text{tr}_i) \\ \quad \cdot \rho_i \leftarrow \{0, 1\}^{r_i} \end{array} \right].$$

The right-side experiment corresponds to an interaction between the IOP prover $\tilde{\mathbf{P}}_{\text{iop}}(\mathcal{P})$ and the IOP verifier (which sends ρ_i in round i). Hence the above probability equals the following probability:

$$\Pr \left[\langle \tilde{\mathbf{P}}_{\text{iop}}, \mathbf{V}_{\text{iop}}(\mathbf{x}, (\rho_i)_{i \in [k]}) \rangle_{\text{iop}} = 1 \mid (\rho_i)_{i \in [k]} \leftarrow \{0, 1\}^r \right].$$

- The term in Equation 24.3 is upper bounded via the MT extraction error (Lemma 18.5.6), as we now explain. The probability does not decrease if we omit the condition on \mathbf{V}_{iop} . Therefore the term in Equation 24.3 can be upper bounded by the following expression:

$$\Pr \left[\wedge_{i \in [k]} \text{MT}.\text{Check}^f(\mathbf{r}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) = 1 \wedge \vee_{i \in [k]} \Pi_i[Q_i] \neq \mathbf{a}_i \mid \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \text{For } i = 1, \dots, k: \\ \quad \cdot (\mathbf{r}_i, \mathbf{aux}_i) \xleftarrow{\text{tr}_i} \tilde{\mathcal{P}}^f(\mathbf{aux}_{i-1}, \rho_{i-1}) \\ \quad \cdot (\Pi_i, \mathbf{td}_i) \leftarrow \text{MT}.\text{Extract}(\mathbf{r}_i, \text{tr}_i) \\ \quad \cdot \rho_i \leftarrow \{0, 1\}^{r_i} \\ ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]} \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}_k, \rho_k) \end{array} \right].$$

In turn, the above probability is at most $\kappa_{\text{MT}}(\lambda, t, l, k, k)$ by Lemma 18.5.6. (Here we do not use the fact that Lemma 18.5.6 guarantees that each \mathbf{td}_i can be used to extract a corresponding Merkle opening proof.)

□

24.3 Adaptive soundness

We prove that Construction 24.1.1 is *adaptively* sound (Definition 4.2.3): the probability that a bounded-query adversary outputs an instance not in the language and convinces the argument verifier is small. Theorem 24.3.1 below provides an upper bound on this error (which happens to be the same as the non-adaptive case in Theorem 24.2.1).

Theorem 24.3.1

Let IOP be a public-coin IOP for a relation \mathcal{R} with soundness error ϵ_{IOP} , round complexity k , and proof length l . For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{IARG} := \text{iBCS}[\text{IOP}, \lambda, s]$ in Construction 24.1.1 is an interactive argument for \mathcal{R} with adaptive soundness error ϵ_{ARG} (see Definition 4.2.3) such that

$$\epsilon_{\text{ARG}}(\lambda, t, n) \leq \epsilon_{\text{IOP}}(n) + \kappa_{\text{MT}}(\lambda, t, l, k, k).$$

Above κ_{MT} is the MT multi-extraction error from Lemma 18.5.6, and $\kappa_{\text{MT}}(\lambda, t, l, k, k) \leq 2 \cdot \frac{t^2}{2^\lambda}$ if $6 \cdot k \cdot (\log l + 1) \leq t$.

We cannot use the security reduction in Lemma 24.2.2 to prove the adaptive case, because now the choice of instance by the malicious prover depends on the random oracle. Nevertheless, we can adapt the statement of the lemma and its proof in a straightforward way to accommodate instances chosen by the malicious prover. Theorem 24.3.1 follows by noting that, for every instance \mathbf{x} such that $|\mathbf{x}| \leq n$ and $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$, the IOP verifier accepts \mathbf{x} with probability at most $\epsilon_{\text{IOP}}(n)$.

Lemma 24.3.2. *There exists an IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ such that for every t -query argument prover $\tilde{\mathcal{P}}$ the following holds for every instance size bound $n \in \mathbb{N}$:*

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \langle \tilde{\mathcal{P}}^f(\mathbf{aux}), \mathcal{V}^f(\mathbf{x}) \rangle_{\text{ARG}} = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \mathbf{aux}) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ & \leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \langle \tilde{\mathbf{P}}_{\text{IOP}}(\mathbf{aux}, \mathcal{P}), \mathbf{V}_{\text{IOP}}(\mathbf{x}) \rangle_{\text{IOP}} = 1 \end{array} \middle| (\mathbf{x}, \mathbf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{IOP}}(\mathcal{P}) \right] + \kappa_{\text{MT}}(\lambda, t, l, k, k). \end{aligned}$$

Moreover, the running time of $\tilde{\mathbf{P}}_{\text{IOP}}$ equals the running time of $\tilde{\mathcal{P}}$ plus $\text{et}_{\text{MT}}(\lambda, \Sigma, l, s, t, k)$ where et_{MT} is the time complexity of the MT extractor in Lemma 18.5.6.

Proof. Below, since the argument prover $\tilde{\mathcal{P}}$ moves first, we can write that $\tilde{\mathcal{P}}$ outputs simultaneously the choice of instance \mathbf{x} , its first message \mathbf{rt}_1 for \mathcal{V} , and an auxiliary state \mathbf{aux}_1 for after receiving \mathcal{V} 's first message.

Let $\text{MT}.\text{MultiExtract}$ be the MT extractor for multiple commitments from Lemma 18.5.6 in Section 18.5.2. The IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ works as follows:

1. Start running the argument prover $\tilde{\mathcal{P}}$, simulating its random oracle f .
2. For $i = 1, \dots, k$:
 - a) If $i = 1$: When $\tilde{\mathcal{P}}$ outputs the instance \mathbf{x} and its first Merkle commitment \mathbf{rt}_1 , let \mathbf{tr}_1 be the query-answer trace of this partial execution since the beginning.
 - If $i > 1$: When $\tilde{\mathcal{P}}$ outputs the i -th Merkle commitment \mathbf{rt}_i , let \mathbf{tr}_i be the query-answer trace of this partial execution since outputting \mathbf{rt}_{i-1} .
 - b) Run the MT extractor to obtain a proof string: $(\Pi_i, \mathbf{td}_i) \leftarrow \text{MT}.\text{MultiExtract}(\mathbf{rt}_i, \mathbf{tr}_i)$. ($\text{MT}.\text{MultiExtract}$ also outputs an opening trapdoor \mathbf{td}_i , but we do not use that here.)

- c) Send the IOP string Π_i to the IOP verifier.
- d) Receive the IOP random message $\rho_i \in \{0, 1\}^{r_i}$, and forward it to $\tilde{\mathcal{P}}$.

The IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ uses the argument prover $\tilde{\mathcal{P}}$ as a black box. The running time of $\tilde{\mathbf{P}}_{\text{IOP}}$ is the running time of (a partial execution of) $\tilde{\mathcal{P}}$ plus $\mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathbf{l}, s, t, k)$; indeed, the MT extractor $\text{MT}.\text{MultiExtract}$ is invoked against an adversary that makes at most t queries to the random oracle and outputs k Merkle commitments across its execution.

We argue that the IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ satisfies the claimed property.

By construction, the argument verifier \mathcal{V} accepts if and only if the IOP verifier \mathbf{V}_{IOP} accepts and each invocation of the Merkle commitment checking algorithm $\text{MT}.\text{Check}$ accepts:

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \langle \tilde{\mathcal{P}}^f(\mathbf{aux}), \mathcal{V}^f(\mathbf{x}) \rangle_{\text{ARG}} = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \mathbf{aux}) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ &= \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{IOP}}^{[Q_i, \mathbf{a}_i]_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \\ \wedge \wedge_{i \in [k]} \text{MT}.\text{Check}^f(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \mathbf{rt}_1, \mathbf{aux}_1) \leftarrow \tilde{\mathcal{P}}^f \\ \rho_1 \leftarrow \{0, 1\}^{r_1} \\ \text{For } i = 2, \dots, k: \\ \quad \cdot (\mathbf{rt}_i, \mathbf{aux}_i) \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}_{i-1}, \rho_{i-1}) \\ \quad \cdot \rho_i \leftarrow \{0, 1\}^{r_i} \\ \quad ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]} \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}_k, \rho_k) \end{array} \right]. \end{aligned}$$

We split the above probability as a sum of two probabilities that separately consider the disjoint cases where all extractions succeed or at least one extraction fails:

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{IOP}}^{[Q_i, \mathbf{a}_i]_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \\ \wedge \wedge_{i \in [k]} \text{MT}.\text{Check}^f(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) = 1 \\ \wedge \wedge_{i \in [k]} \Pi_i[Q_i] = \mathbf{a}_i \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \mathbf{rt}_1, \mathbf{aux}_1) \xleftarrow{\mathbf{tr}_1} \tilde{\mathcal{P}}^f \\ (\Pi_1, \mathbf{td}_1) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}_1, \mathbf{tr}_1) \\ \rho_1 \leftarrow \{0, 1\}^{r_1} \\ \text{For } i = 2, \dots, k: \\ \quad \cdot (\mathbf{rt}_i, \mathbf{aux}_i) \xleftarrow{\mathbf{tr}_i} \tilde{\mathcal{P}}^f(\mathbf{aux}_{i-1}, \rho_{i-1}) \\ \quad \cdot (\Pi_i, \mathbf{td}_i) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}_i, \mathbf{tr}_i) \\ \quad \cdot \rho_i \leftarrow \{0, 1\}^{r_i} \\ \quad ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]} \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}_k, \rho_k) \end{array} \right] \tag{24.4} \\ & + \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{IOP}}^{[Q_i, \mathbf{a}_i]_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \\ \wedge \wedge_{i \in [k]} \text{MT}.\text{Check}^f(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) = 1 \\ \wedge \vee_{i \in [k]} \Pi_i[Q_i] \neq \mathbf{a}_i \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \mathbf{rt}_1, \mathbf{aux}_1) \xleftarrow{\mathbf{tr}_1} \tilde{\mathcal{P}}^f \\ (\Pi_1, \mathbf{td}_1) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}_1, \mathbf{tr}_1) \\ \rho_1 \leftarrow \{0, 1\}^{r_1} \\ \text{For } i = 2, \dots, k: \\ \quad \cdot (\mathbf{rt}_i, \mathbf{aux}_i) \xleftarrow{\mathbf{tr}_i} \tilde{\mathcal{P}}^f(\mathbf{aux}_{i-1}, \rho_{i-1}) \\ \quad \cdot (\Pi_i, \mathbf{td}_i) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}_i, \mathbf{tr}_i) \\ \quad \cdot \rho_i \leftarrow \{0, 1\}^{r_i} \\ \quad ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]} \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}_k, \rho_k) \end{array} \right] \tag{24.5} \end{aligned}$$

We conclude the proof by upper bounding each of the two terms.

- The term in Equation 24.4 is upper bounded by the IOP error, as we now explain. If $\wedge_{i \in [k]} \Pi_i[Q_i] = \mathbf{a}_i$ (every extracted IOP string agrees with the relevant query answers) then $\mathbf{V}_{\text{IOP}}^{[Q_i, \mathbf{a}_i]_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = \mathbf{V}_{\text{IOP}}^{(\Pi)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]})$ (the IOP verifier returns the same output). Moreover, the probability does not decrease if we omit the conditions on $\text{MT}.\text{Check}$ (and thus can ignore the final output of $\tilde{\mathcal{P}}$). Therefore the term in Equation 24.4 can be upper bounded by the following expression:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{IOP}}^{(\Pi)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \mathbf{rt}_1, \mathbf{aux}_1) \xleftarrow{\text{tr}_1} \tilde{\mathcal{P}}^f \\ (\Pi_1, \mathbf{td}_1) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}_1, \mathbf{tr}_1) \\ \rho_1 \leftarrow \{0, 1\}^{r_1} \\ \text{For } i = 2, \dots, k: \\ \quad \cdot (\mathbf{rt}_i, \mathbf{aux}_i) \xleftarrow{\text{tr}_i} \tilde{\mathcal{P}}^f(\mathbf{aux}_{i-1}, \rho_{i-1}) \\ \quad \cdot (\Pi_i, \mathbf{td}_i) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}_i, \mathbf{tr}_i) \\ \quad \cdot \rho_i \leftarrow \{0, 1\}^{r_i} \end{array} \right].$$

The right-side experiment corresponds to an interaction between the IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}(\mathcal{P})$ and the IOP verifier (which sends ρ_i in round i). Hence the above probability equals the following probability:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \langle \tilde{\mathbf{P}}_{\text{IOP}}(\mathbf{aux}, \mathcal{P}), \mathbf{V}_{\text{IOP}}(\mathbf{x}) \rangle_{\text{IOP}} = 1 \end{array} \middle| (\mathbf{x}, \mathbf{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{IOP}}(\mathcal{P}) \right].$$

- The term in Equation 24.5 is upper bounded via the MT extraction error (Lemma 18.5.6), as we now explain. The probability does not decrease if we omit the condition on \mathbf{V}_{IOP} . Therefore the term in Equation 24.5 can be upper bounded by the following expression:

$$\Pr \left[\begin{array}{l} \wedge_{i \in [k]} \text{MT}.\text{Check}^f(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) = 1 \\ \wedge \vee_{i \in [k]} \Pi_i[Q_i] \neq \mathbf{a}_i \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \mathbf{rt}_1, \mathbf{aux}_1) \xleftarrow{\text{tr}_1} \tilde{\mathcal{P}}^f \\ (\Pi_1, \mathbf{td}_1) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}_1, \mathbf{tr}_1) \\ \rho_1 \leftarrow \{0, 1\}^{r_1} \\ \text{For } i = 2, \dots, k: \\ \quad \cdot (\mathbf{rt}_i, \mathbf{aux}_i) \xleftarrow{\text{tr}_i} \tilde{\mathcal{P}}^f(\mathbf{aux}_{i-1}, \rho_{i-1}) \\ \quad \cdot (\Pi_i, \mathbf{td}_i) \leftarrow \text{MT}.\text{Extract}(\mathbf{rt}_i, \mathbf{tr}_i) \\ \quad \cdot \rho_i \leftarrow \{0, 1\}^{r_i} \\ \quad ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]} \leftarrow \tilde{\mathcal{P}}^f(\mathbf{aux}_k, \rho_k) \end{array} \right].$$

In turn, the above probability is at most $\kappa_{\text{MT}}(\lambda, t, l, k, k)$ by Lemma 18.5.6. (Here we do not use the fact that Lemma 18.5.6 guarantees that each \mathbf{td}_i can be used to extract a corresponding Merkle opening proof.)

□

25 The BCS transformation

We describe how to construct a succinct *non-interactive* argument, starting from any public-coin interactive oracle proof (IOP). This is known as the *Ben-Sasson–Chiesa–Spooner (BCS) transformation* [BCS16], and builds on the succinct interactive argument discussed in Chapter 24.

25.1 Construction

The construction can be informally described in two steps.

- First apply the iBCS transformation from Chapter 24. This transforms the given (public-coin) IOP into a succinct (public-coin) interactive argument.
- Then, using a separate random oracle, apply the Fiat–Shamir transformation for (public-coin) IPs from Chapter 15. This removes the interaction, resulting in a succinct non-interactive argument. (Alternatively, one could apply the “plain” variant of the Fiat–Shamir transformation from Chapter 14; see Remark 25.1.3.)

In other words, the prover, after committing to each IOP string via a Merkle commitment, can simply use the random oracle to itself derive IOP randomness based on the Merkle commitment (rather than receiving the IOP randomness from the verifier).

Below we describe the construction in detail.

Construction 25.1.1: Ben-Sasson–Chiesa–Spooner (BCS) transformation

Let $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ be a public-coin IOP with round complexity k , proof length l over alphabet Σ , query complexity q , and randomness complexity r . Let $\lambda \in \mathbb{N}$ be a security parameter and $s \in \mathbb{N}$ be a privacy parameter. Suppose that $\min_{i \in [k]} r_i \geq \lambda$. Define $\text{MT} := \text{MT}[\lambda, \Sigma, l, s]$ to be the Merkle commitment scheme with the stated parameters.

We define $\text{NARG} := \text{BCS}[\text{IOP}, \lambda, s]$ to be the non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ constructed as follows. The argument prover \mathcal{P} receives as input an instance \mathbf{x} and witness \mathbf{w} , and the argument verifier \mathcal{V} receives as input the instance \mathbf{x} and an argument string π . Both receive query access to $k + 1$ random oracles $(f_{\text{MT}}, (f_i)_{i \in [k]}) \in \mathcal{U}((\lambda, (r_i)_{i \in [k]}))$: (a) an oracle f_{MT} used for the Merkle commitment scheme MT , and (b) for every $i \in [k]$, an oracle f_i used to derive IOP randomness for round i (and extend the hash chain); this implies that the oracle configuration cnf is defined as $\text{cnf}(\lambda, n) := (\lambda, (r_i)_{i \in [k]})$ (see Definition 7.1.1).

- $\mathcal{P}^f(\mathbf{x}, \mathbf{w})$:
 1. For $i = 1, \dots, k$:

a) Compute the i -th proof string (and auxiliary state) of the IOP prover:

$$(\Pi_i \in \Sigma^{l_i}, \text{aux}_i) := \begin{cases} \mathbf{P}_{\text{IOP}}(\mathbf{x}, \mathbf{w}) & \text{if } i = 1 \\ \mathbf{P}_{\text{IOP}}(\text{aux}_{i-1}, \rho_{i-1}) & \text{if } i > 1 \end{cases}.$$

b) Commit to Π_i via the Merkle commitment scheme:

$$(\mathbf{rt}_i, \mathbf{td}_i) := \text{MT}.\text{Commit}^{f_{\text{MT}}}(\Pi_i).$$

c) Sample a random salt $\tau_i \in \{0, 1\}^s$.

d) Derive IOP randomness

$$\rho_i := \begin{cases} f_1(\mathbf{x}, \mathbf{rt}_1, \tau_1) \in \{0, 1\}^{r_1} & \text{if } i = 1 \\ f_i(\rho_{i-1}, \mathbf{rt}_i, \tau_i) \in \{0, 1\}^{r_i} & \text{if } i > 1 \end{cases}.$$

2. Simulate the IOP verifier $\mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]})$, yielding query sets $(Q_i \subseteq [l_i])_{i \in [k]}$.
3. For every $i \in [k]$, set $\mathbf{a}_i := \Pi_i[Q_i] \in \Sigma^{Q_i}$ to be the answers for the i -th query set.
4. For every $i \in [k]$, compute an opening proof for \mathbf{a}_i : $\mathbf{pf}_i := \text{MT}.\text{Open}^{f_{\text{MT}}}(\mathbf{td}_i, Q_i)$.
5. Output the argument string $\pi := ((\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i, \tau_i))_{i \in [k]}$.

- $\mathcal{V}^f(\mathbf{x}, \pi)$:

1. Parse the argument string π as a tuple $((\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i, \tau_i))_{i \in [k]}$.
2. For $i = 1, \dots, k$:
 - a) derive IOP verifier randomness ρ_i as in Item 1d of the argument prover \mathcal{P} .
 3. Check that $\mathbf{V}_{\text{IOP}}^{[Q_i, \mathbf{a}_i]_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1$. (The IOP verifier $\mathbf{V}_{\text{IOP}}(\mathbf{x}, (\rho_i)_{i \in [k]})$ accepts if each query $j \in Q_i$ is answered with $\mathbf{a}_i[j] \in \Sigma$. If any query falls outside the sets $(Q_i)_{i \in [k]}$ then reject.)
 4. Check that $\wedge_{i \in [k]} \text{MT}.\text{Check}^{f_{\text{MT}}}(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) = 1$. (The query answers are authenticated with respect to the Merkle commitments $(\mathbf{rt}_i)_{i \in [k]}$.)

Efficiency We discuss efficiency properties of the above construction.

- *Argument size.* The argument string π in the above construction contains the same information sent from the argument prover in the iBCS transformation (the Merkle commitments, and the authenticated answers to the queries), plus k salts. Via a similar step as in Equation 24.1, this leads to an argument size that is

$$O(k \cdot (\lambda + s) + q \cdot (\log |\Sigma| + s + \lambda \cdot \log l)). \quad (25.1)$$

The difference is that the argument prover provides the information in a single message, with randomness derived via the random oracle (as in the Fiat–Shamir transformation).

- *Prover complexity.* The complexity of the argument prover is typically dominated by the complexity of the underlying IOP prover, as in the iBCS transformation. Moreover, the number of queries to the random oracle is $\mathbf{q}_P = O(k + l)$: (a) $O(l)$ queries to the random oracle f_{MT} (to commit to the IOP strings, as in the iBCS transformation); and (b) k queries to the random oracle f_s (to derive IOP randomness).
- *Verifier complexity.* The complexity of the argument verifier is typically dominated by the complexity of the underlying IOP verifier, as in the iBCS transformation. Moreover, the number of queries to the random oracle is $\mathbf{q}_V = O(k + q \cdot \log l)$: (a) $O(q \cdot \log l)$ queries to the random oracle f_{MT} (to check the Merkle opening proofs for q answers, as in the iBCS transformation); and (b) k queries to the random oracle f_s (to derive IOP randomness).

Remark 25.1.2 (non-oracle messages in Construction 25.1.1). We discuss the modifications needed to additionally support non-oracle messages $(\alpha_i)_{i \in [k]}$ from the IOP prover.

- In Item 1d of the argument prover \mathcal{P} , the IOP randomness is derived as

$$\rho_i := \begin{cases} f_1(\mathbf{x}, \mathbf{rt}_1, \alpha_1, \tau_1) \in \{0, 1\}^{r_1} & \text{if } i = 1 \\ f_i(\rho_{i-1}, \mathbf{rt}_i, \alpha_i, \tau_i) \in \{0, 1\}^{r_i} & \text{if } i > 1 \end{cases}.$$

That is, the non-oracle message α_i is included in the query to the random oracle.

- In Item 2 of the argument prover \mathcal{P} , the IOP verifier is run as

$$\mathbf{V}_{IOP}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}).$$

- In Item 5 of the argument prover \mathcal{P} , the argument string π is augmented to include all the non-oracle messages:

$$((\mathbf{rt}_i, Q_i, \mathbf{a}_i, \alpha_i, \mathbf{pf}_i, \tau_i))_{i \in [k]}.$$

- In Item 3 of the argument verifier \mathcal{V} , the explicit input to the IOP verifier is

$$(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}).$$

Remark 25.1.3. Construction 25.1.1 is the result of applying the faster variant of the Fiat–Shamir transformation (Construction 15.1.1) to the interactive argument in Construction 24.1.1. One can, instead, apply the plain variant of the Fiat–Shamir transformation (Construction 14.1.1), resulting in a simpler (and less optimized) construction. In this case, we do not need the condition $\min_{i \in [k]} r_i \geq \lambda$. Moreover, in Item 1d the argument prover derives the IOP randomness as follows

$$\rho_i := \begin{cases} f_1(\mathbf{x}, \mathbf{rt}_1, \tau_1) \in \{0, 1\}^{r_1} & \text{if } i = 1 \\ f_i(\mathbf{x}, (\mathbf{rt}_1, \dots, \mathbf{rt}_i), (\tau_1, \dots, \tau_i)) \in \{0, 1\}^{r_i} & \text{if } i > 1 \end{cases}.$$

Similarly to the differing security bounds between the plain variant and faster variant of the Fiat–Shamir transformation, the above simpler variant leads to slightly smaller security bounds: the error term $\frac{t^2}{2^\lambda}$, which represents the possibility of breaking the hash chain in the faster variant, does not appear in the security bounds.

Remark 25.1.4 (more randomness than time). We explain how the BCS transformation can be modified to support IOPs where the IOP verifier has oracle access to its own randomness, which in particular enables the IOP prover to receive much more randomness than allowed by the IOP verifier’s running time.

Consider a generalization of the definition of a public-coin IOP whereby the IOP verifier’s decision is computed by querying the randomness sent by the IOP verifier over the interaction. In other words, the decision is computed as

$$\mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}, (\rho_i)_{i \in [k]}}(\mathbf{x}) \quad \text{rather than} \quad \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}).$$

We can modify Construction 25.1.1 to support this more general notion of public-coin IOPs. Namely, in Item 1d the argument prover \mathcal{P} derives the j -th entry $\rho_{i,j}$ of the random string ρ_i as

$$\rho_{i,j} := \begin{cases} f_1(\mathbf{x}, \mathbf{rt}_1, \tau_1, j) & \text{if } i = 1 \\ f_i(\rho_{i-1}, \mathbf{rt}_i, \tau_i, j) & \text{if } i > 1 \end{cases}.$$

Subsequently, the argument verifier \mathcal{V} uses the random oracle f_i to recompute only the entries of ρ_i that the IOP verifier queries.

This ensures that the running time of the argument verifier \mathcal{V} *only* depends on the number of queries to the randomness, *but not on the total randomness complexity*. Indeed, the argument verifier only needs to invoke the random oracle to answer randomness queries of the underlying IOP verifier, and in particular, does not have to materialize the unqueried randomness. This is because the random oracle, which serves as a shared randomness resource, enables the argument prover to suitably materialize all the relevant randomness without impacting the argument verifier.

We did not discuss such a generalization for the Fiat–Shamir transformation or the Micali transformation, but in both cases it is straightforward to support an IP verifier or PCP verifier that makes oracle queries to its own randomness.

25.2 Soundness

We prove that Construction 25.1.1 is adaptively sound. The following theorem states that the adaptive soundness error of the non-interactive argument is upper bounded by the IOP state-restoration soundness error of the underlying IOP plus a small error term. The error term has two parts: one part represents the probability of “breaking” the Merkle commitment scheme used to commit to IOP strings; and the other part represents the probability of “breaking” the hash-chain of IOP randomness strings. The fact that the upper bound depends on the IOP state-restoration soundness error, rather than the IOP (standard) soundness error, is expected, due to the use of the Fiat–Shamir transformation.

Theorem 25.2.1

Let IOP be a public-coin IOP for a relation \mathcal{R} with state-restoration soundness error $\epsilon_{\text{IOP}}^{\text{sr}}$ (see Definition 23.2.2), round complexity k , and proof length l . For every security

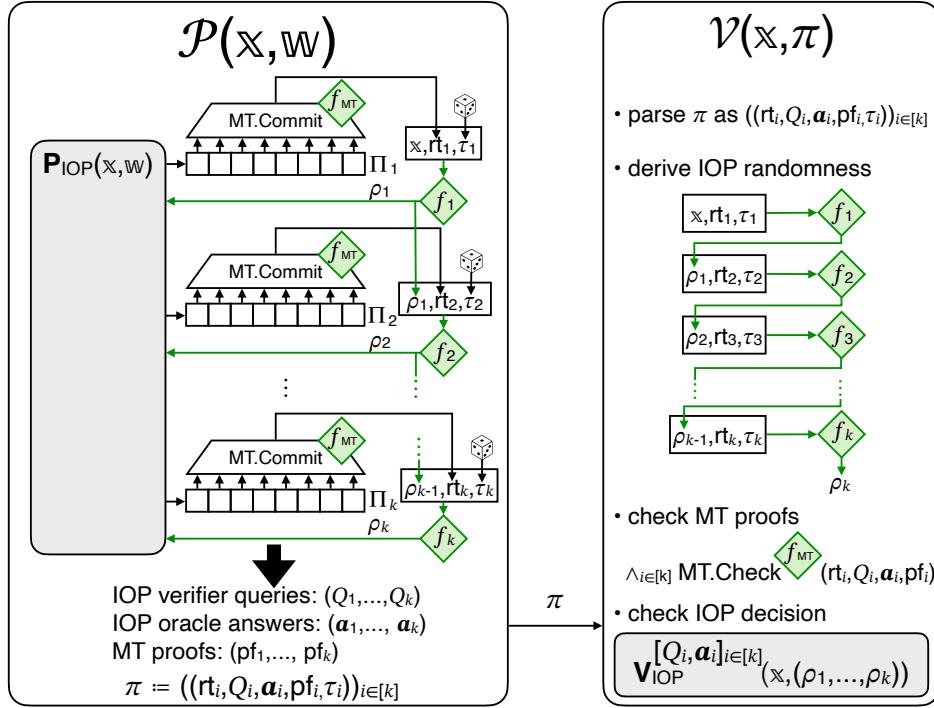


Figure 25.1: Diagram of the Ben-Sasson–Chiesa–Spooner (BCS) transformation (BCS in Construction 25.1.1).

parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{NARG} := \text{BCS}[\text{IOP}, \lambda, s]$ in Construction 25.1.1 is a non-interactive argument for \mathcal{R} with adaptive soundness error ϵ_{ARG} (see Definition 7.1.4) such that

$$\epsilon_{\text{ARG}}(\lambda, t, n) \leq \epsilon_{\text{IOP}}^{\text{sr}}(\lambda + s, t, n) + \kappa_{\text{MT}}(\lambda, t, l, (t+1) \cdot k, k) + \frac{t^2}{2^\lambda}.$$

Above κ_{MT} is the MT multi-extraction error from Lemma 18.5.6. If $6 \cdot k \cdot (\log l + 1) \leq t$, then $\epsilon_{\text{ARG}}(\lambda, t, n) \leq \epsilon_{\text{IOP}}^{\text{sr}}(\lambda + s, t, n) + 3 \cdot \frac{t^2}{2^\lambda}$.

We discuss two natural approaches to prove the theorem. In Section 25.2.1 we outline one approach, and in Section 25.2.2 we work out the other approach.

25.2.1 Direct approach

Theorem 25.2.1 can be proved by reducing an argument prover $\tilde{\mathcal{P}}$ to a corresponding IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$, relying on the security properties of Merkle commitment schemes and hash chains (each of which introduces a small error). The lemma below captures this reduction and directly implies the theorem using the fact that $t_{\$}, t_{\text{MT}} \leq t$.

Lemma 25.2.2. *There exists an IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ such that, for every $(t_{\text{MT}}, t_{\$})$ -*

query argument prover $\tilde{\mathcal{P}}$, $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}})$ makes at most t_s moves and the following holds:

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \mid \begin{array}{l} f = (f_{\text{MT}}, (f_i)_{i \in [k]}) \leftarrow \mathcal{U}((\lambda, (\mathbf{r}_i)_{i \in [k]})) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ & \leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \end{array} \mid \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \leftarrow \text{Game}_{\text{IOP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}})).f_{\text{MT}}) \end{array} \right] \\ & \quad + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, l, (t_s + 1) \cdot k, k) + \frac{t_s^2}{2^\lambda}. \end{aligned}$$

Moreover, the running time of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ is bounded by

$$t_{\mathcal{P}} + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, l, s, t_{\text{MT}}, (t_s + 1) \cdot k) + O(r_{\max} \cdot k \cdot t_s \cdot \log t_s),$$

where $t_{\mathcal{P}}$ is the running time of $\tilde{\mathcal{P}}$ and \mathbf{et}_{MT} is the time complexity of the MT extractor in Lemma 18.5.6.

We do not prove the lemma (we work out the other approach in Section 25.2.2 instead). Nevertheless below we provide the construction of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ from $\tilde{\mathcal{P}}$, as it aids in understanding the security reduction. We recommend comparing the construction of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ below to the construction of $\tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}$ for the Micali transformation in Construction 21.2.4.

Construction 25.2.3. Let $\text{MT}.\text{MultiExtract}$ be the MT extractor for multiple commitments from Lemma 18.5.6 in Section 18.5.2; below we use the notation for $\text{MT}.\text{MultiExtract}$ introduced in Remark 18.5.8. Let $\text{FS}.\text{Backtrack}$ be the procedure in Definition 15.2.2 (used in the analysis of the Fiat–Shamir transformation). The IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}} := \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}})$ works as follows.

1. For every $i \in [k]$, lazily sample an oracle $\dot{g}_i \leftarrow \mathcal{U}(\mathbf{r}_i)$ (to be used as auxiliary randomness).
2. Initialize an empty query-answer trace tr_{MT} .
3. Simulate $\tilde{\mathcal{P}}$ while answering its queries as described below.
4. When $\tilde{\mathcal{P}}$ performs a query to the oracle f_{MT} , answer according to f_{MT} and append the resulting query-answer pair to tr_{MT} .
5. When $\tilde{\mathcal{P}}$ performs a query x to the oracle f_i (for $i \in [k]$), do the following:
 - a) Let $\text{tr} \subseteq \text{tr}_{\text{MT}}$ be the query-answer pairs for f_{MT} between the previous iteration and the current iteration (or the beginning, if this is the first iteration).
 - b) If $i = 1$ and x looks like a query $(\mathbf{x}, \mathbf{rt}_1, \tau_1)$:
 - i. Run the Merkle extractor: $(\Pi_1, \text{td}_1) := \text{MT}.\text{MultiExtract}(\mathbf{rt}_1, \text{tr})$.
 - ii. Set the salt string $\sigma_1 := (\mathbf{rt}_1, \tau_1)$.
 - iii. Output the move $(\mathbf{x}, \Pi_1, \sigma_1)$ for the IOP state-restoration game.
 - iv. Receive IOP randomness $\rho_1 \in \{0, 1\}^{r_1}$ from the game.
 - c) If $i \in \{2, \dots, k\}$ and x looks like a query $(\rho_{i-1}, \mathbf{rt}_i, \tau_i)$:
 - i. Let tr_{fast} be the query-answer trace of $\tilde{\mathcal{P}}$ to $(f_i)_{i \in [k]}$ (so far).
 - ii. Compute $(\mathbf{x}, (\mathbf{rt}_1, \dots, \mathbf{rt}_{i-1}), (\tau_1, \dots, \tau_{i-1})) := \text{FS}.\text{Backtrack}(i - 1, \rho_{i-1}, \text{tr}_{\text{fast}})$.
 - iii. If $\text{FS}.\text{Backtrack}$ aborted then set $\rho_i := \dot{g}_i(x)$ and skip to Item 5e.
 - iv. Compute $((\Pi_1, \text{td}_1), \dots, (\Pi_i, \text{td}_i)) := \text{MT}.\text{MultiExtract}((\mathbf{rt}_1, \dots, \mathbf{rt}_i), \text{tr})$.

- v. For every $j \in [i]$, set the salt string $\sigma_j := (\mathbf{rt}_j, \tau_j)$.
- vi. Output the move $(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i))$ for the IOP state-restoration game.
- vii. Receive IOP randomness $\rho_i \in \{0, 1\}^{r_i}$ from the game.
- d) Otherwise set $\rho_i := \dot{g}_i(x)$.
- e) Return ρ_i to $\tilde{\mathcal{P}}$ as the answer to its query.
- 6. Let (\mathbf{x}, π) be the output of $\tilde{\mathcal{P}}$ when it halts, and parse π as $((\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i, \tau_i))_{i \in [k]}$.
- 7. Let $\mathbf{tr}' \subseteq \mathbf{tr}_{\text{MT}}$ be the query-answer pairs for f_{MT} between the last iteration and $\tilde{\mathcal{P}}$'s halting.
- 8. Run the Merkle extractor: $((\Pi_i, \mathbf{td}_i))_{i \in [k]} := \text{MT}. \text{MultiExtract}((\mathbf{rt}_i)_{i \in [k]}, \mathbf{tr}')$.
- 9. For every $i \in [k]$, set the salt string $\sigma_i := (\mathbf{rt}_i, \tau_i)$.
- 10. Set $b_{\text{MT}} := \wedge_{i \in [k]} \text{MT}. \text{Check}^{f_{\text{MT}}}(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) = 1$. (This bit is only used in the analysis.)
- 11. Output $(\mathbf{x}, (\Pi_1, \dots, \Pi_k), (\sigma_1, \dots, \sigma_k))$.

The prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ potentially makes a move for every query of $\tilde{\mathcal{P}}$ to the oracles $(f_i)_{i \in [k]}$ (no other moves are performed). Since $\tilde{\mathcal{P}}$ makes at most $t_{\$}$ queries to $(f_i)_{i \in [k]}$, $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ performs at most $t_{\$}$ moves. Next, we discuss the running time of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$. The simulation of $\tilde{\mathcal{P}}$ takes time $\mathbf{t}_{\mathcal{P}}$, the running time of $\tilde{\mathcal{P}}$. Moreover, $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ performs at most $t_{\$}$ calls to $\text{FS}. \text{Backtrack}$; $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ can ensure that the query-answer trace $\mathbf{tr}_{\text{fast}}$ given as input to $\text{FS}. \text{Backtrack}$ is already sorted (lexicographically by answer and then by query), which takes time $r_{\max} \cdot t_{\$} \cdot \log t_{\$}$, and then each call to $\text{FS}. \text{Backtrack}$ takes time $k \cdot r_{\max} \cdot \log t_{\$}$ (see Definition 15.2.2). Finally, $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ performs at most $(t_{\$} + 1) \cdot k$ calls and Merkle commitment extractions ($\tilde{\mathcal{P}}$ makes at most $t_{\$}$ queries to $(f_i)_{i \in [k]}$ each leading to up to k Merkle commitment extractions and a final output leading to k Merkle commitment extractions). The total time for all extractions is given by the expression $\mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathbf{l}, s, t_{\text{MT}}, (t_{\$} + 1) \cdot k)$ where \mathbf{et}_{MT} is the (total) time complexity of $\text{MT}. \text{MultiExtract}$ in Lemma 18.5.6.

Overall, running time of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ equals the running time of $\tilde{\mathcal{P}}$ plus $\mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathbf{l}, s, t_{\text{MT}}, (t_{\$} + 1) \cdot k) + O(r_{\max} \cdot k \cdot t_{\$} \cdot \log t_{\$})$.

The key claim to proving Lemma 25.2.2 is the following claim.

Claim 25.2.4. *For every $(t_{\text{MT}}, t_{\$})$ -query argument prover $\tilde{\mathcal{P}}$, $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}})$ in Construction 25.2.3 makes at most $t_{\$}$ moves and the statistical distance of the following two distribution is upper bounded by $\kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathbf{l}, (t_{\$} + 1) \cdot k, k) + \frac{t_{\$}^2}{2^\lambda}$:*

$$\left\{ \begin{array}{l} (\mathbf{tr}_{\text{MT}}, (\rho_i)_{i \in [k]}, \mathbf{x}, (\Pi_i)_{i \in [k]}, b) \\ \left| \begin{array}{l} f = (f_{\text{MT}}, (f_i)_{i \in [k]}) \leftarrow \mathcal{U}((\lambda, (r_i)_{i \in [k]})) \\ (\mathbf{x}, \pi = ((\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i, \tau_i))_{i \in [k]}) \xleftarrow{\mathbf{tr}} \tilde{\mathcal{P}}^f \\ ((\Pi_i, \mathbf{td}_i))_{i \in [k]} := \text{MT}. \text{MultiExtract}((\mathbf{rt}_i)_{i \in [k]}, \mathbf{tr}_{\text{MT}}) \\ b \xleftarrow{\mathbf{tr}_v} \mathcal{V}^f(\mathbf{x}, \pi) \\ \text{parse } \mathbf{tr}_v \text{ as } ((x_i, \rho_i))_{i \in [k]} \end{array} \right. \end{array} \right\} \text{ and} \\ \left\{ \begin{array}{l} (\mathbf{tr}_{\text{MT}}, (\rho_i)_{i \in [k]}, \mathbf{x}, (\Pi_i)_{i \in [k]}, b) \\ \left| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow[\mathbf{tr}_{\text{MT}}, b_{\text{MT}}]{} \mathbf{Game}_{\text{IOP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}))^{f_{\text{MT}}})) \\ b := \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) \wedge b_{\text{MT}} \end{array} \right. \end{array} \right\} .$$

25.2.2 Modular approach

We prove Theorem 25.2.1 via a modular approach, as we explain. The non-interactive argument $(\mathcal{P}, \mathcal{V}) = \mathbf{BCS}[\text{IOP}, \lambda, s]$ in Construction 25.1.1 is the result of applying (the faster variant of) the Fiat–Shamir transformation $\mathbf{FS}_{\text{IP}}^*[\text{IARG}, \lambda, s]$ in Construction 15.1.1 to the interactive argument $\text{IARG} = (\mathcal{P}_i, \mathcal{V}_i) = \mathbf{iBCS}[\text{IOP}, \lambda, s]$ in Construction 24.1.1. In other words,

$$\mathbf{BCS}[\text{IOP}, \lambda, s] = \mathbf{FS}_{\text{IP}}^*[\mathbf{iBCS}[\text{IOP}, \lambda, s], \lambda, s]. \quad (25.2)$$

Note that the Fiat–Shamir transformation is applied to an interactive *argument* IARG in the ROM rather than an interactive *proof* IP. In particular, we use the fact that the security reduction of the Fiat–Shamir transformation works even if the given interactive protocol has “its own” oracles.

In the interactive argument IARG , the argument prover \mathcal{P}_i sends $k + 1$ messages, where the first k messages are Merkle commitments $(\alpha_i)_{i \in [k]} = (\mathbf{rt}_i)_{i \in [k]}$ and the last message is a tuple $\alpha_{k+1} = ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]}$; the argument verifier \mathcal{V}_i sends k random messages $(\rho_i)_{i \in [k]}$. In particular, there is no random verifier message in round $k + 1$, which has a minor impact on some notation below.

In light of Equation 25.2, we will explain how Lemma 16.3.3 tells us that the adaptive soundness error of the non-interactive argument $(\mathcal{P}, \mathcal{V}) = \mathbf{BCS}[\text{IOP}, \lambda, s]$ in Construction 25.1.1 is upper bounded by the IP state-restoration soundness error of the interactive argument $\text{IARG} = (\mathcal{P}_i, \mathcal{V}_i) = \mathbf{iBCS}[\text{IOP}, \lambda, s]$ in Construction 24.1.1. We are thus left to upper bound the IP state-restoration soundness error of the interactive argument $\text{IARG} = (\mathcal{P}_i, \mathcal{V}_i)$. We do this in the lemma below, and conclude the proof of the theorem at the end of the section.

Lemma 25.2.5. *There exists an IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ such that, for every t_{MT} -query argument prover $\tilde{\mathcal{P}}_i^{\text{sr}}$ that makes at most $t_{\$}$ moves in an IP state-restoration game, $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_i^{\text{sr}})$ makes at most $t_{\$}$ moves and the following holds for every instance size bound $n \in \mathbb{N}$:*

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}_i^{f_{\text{MT}}}(\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 1 \end{array} \right] \left| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \leftarrow \\ \text{Game}_{\text{IP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathcal{P}}_i^{\text{sr}})^{f_{\text{MT}}}) \end{array} \right. \right. \\ \leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \end{array} \right] \left| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \leftarrow \\ \text{Game}_{\text{IOP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_i^{\text{sr}}))^{f_{\text{MT}}}) \end{array} \right. \right. \\ + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathsf{l}, (t_{\$} + 1) \cdot k, k).$$

Moreover, the running time of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ equals the running time of $\tilde{\mathcal{P}}_i^{\text{sr}}$ plus $\mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathsf{l}, s, t_{\text{MT}}, (t_{\$} + 1) \cdot k)$ where \mathbf{et}_{MT} is the time complexity of the MT extractor in Lemma 18.5.6.

Construction 25.2.6. Let $\text{MT}.\text{MultiExtract}$ be the MT extractor for multiple commitments from Lemma 18.5.6 in Section 18.5.2; below we use the notation for $\text{MT}.\text{MultiExtract}$ introduced in Remark 18.5.8. The IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}} := \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_i^{\text{sr}})$ works as follows.

1. Initialize an empty query-answer trace \mathbf{tr}_{MT} .
2. Simulate $\tilde{\mathcal{P}}_i^{\text{sr}}$ while answering its queries as described below.

3. When $\tilde{\mathcal{P}}_i^{\text{sr}}$ performs a query to the oracle f_{MT} , answer according to f_{MT} and append the resulting query-answer pair to tr_{MT} .
4. When $\tilde{\mathcal{P}}_i^{\text{sr}}$ performs a move $(\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i))$ with $i \in [k]$ and $(\alpha_1, \dots, \alpha_i) = (\mathbf{r}_1, \dots, \mathbf{r}_i)$:
 - a) Let $\text{tr} \subseteq \text{tr}_{\text{MT}}$ be the query-answer pairs for f_{MT} between the previous iteration and the current iteration (or the beginning if this is the first iteration).
 - b) Compute $((\Pi_1, \text{td}_1), \dots, (\Pi_i, \text{td}_i)) := \text{MT}.\text{MultiExtract}((\mathbf{r}_1, \dots, \mathbf{r}_i), \text{tr})$.
 - c) For every $j \in [i]$, set the salt string $\sigma'_j := (\mathbf{r}_j, \sigma_j)$.
 - d) Output the move $(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma'_1, \dots, \sigma'_i))$ for the IOP state-restoration game.
 - e) Receive IOP randomness $\rho_i \in \{0, 1\}^{r_i}$ from the game.
 - f) Return ρ_i to $\tilde{\mathcal{P}}_i^{\text{sr}}$ as response of the game to its move.
5. Finally, $\tilde{\mathcal{P}}_i^{\text{sr}}$ halts and outputs $(\mathbf{x}, (\alpha_1, \dots, \alpha_k, \alpha_{k+1}), (\sigma_1, \dots, \sigma_k))$, with $(\alpha_1, \dots, \alpha_i) = (\mathbf{r}_1, \dots, \mathbf{r}_i)$ and $\alpha_{k+1} = ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]}$.
6. Let $\text{tr}' \subseteq \text{tr}_{\text{MT}}$ be the query-answer pairs for f_{MT} between the last iteration and $\tilde{\mathcal{P}}_i^{\text{sr}}$'s halting.
7. Run the Merkle extractor: $((\Pi_i, \text{td}_i))_{i \in [k]} := \text{MT}.\text{MultiExtract}((\mathbf{r}_i)_{i \in [k]}, \text{tr}')$.
8. For every $i \in [k]$, set the salt string $\sigma'_i := (\mathbf{r}_i, \sigma_i)$.
9. Set $b_{\text{MT}} := \wedge_{i \in [k]} \text{MT}.\text{Check}^{f_{\text{MT}}}(\mathbf{r}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i)$. (This decision bit is used in the analysis only.)
10. Output $(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]})$.

The prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ makes a move for every move of $\tilde{\mathcal{P}}_i^{\text{sr}}$. Since $\tilde{\mathcal{P}}_i^{\text{sr}}$ makes at most t_s moves, $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ makes at most t_s moves. In terms of running time, $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ simulates $\tilde{\mathcal{P}}_i^{\text{sr}}$ (which takes the running time of $\tilde{\mathcal{P}}_i^{\text{sr}}$) and additionally performs at most $(t_s + 1) \cdot k$ Merkle commitment extractions ($\tilde{\mathcal{P}}_i^{\text{sr}}$ makes at most t_s moves each containing up to k Merkle commitments and a final output containing k Merkle commitments). The total time for all extractions is given by the expression $\text{et}_{\text{MT}}(\lambda, \Sigma, l, s, t_{\text{MT}}, (t_s + 1) \cdot k)$ where et_{MT} is the (total) time complexity of $\text{MT}.\text{MultiExtract}$ in Lemma 18.5.6.

Construction 25.2.7. The function IP2IOPTrace receives as input a move-response trace tr^{sr} and a query-answer trace tr_{MT} , and outputs a move-response trace $\text{tr}_{\text{IOP}}^{\text{sr}}$, and works as follows.

$\text{IP2IOPTrace}(\text{tr}^{\text{sr}}, \text{tr}_{\text{MT}})$:

1. Initialize an empty move-response trace $\text{tr}_{\text{IOP}}^{\text{sr}}$.
2. For every move x in tr^{sr} :
 - a) Let $\rho_i \in \{0, 1\}^{r_i}$ be the response to the move x .
 - b) If $x = (\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i))$ with $i \in [k]$ and $(\alpha_1, \dots, \alpha_i) = (\mathbf{r}_1, \dots, \mathbf{r}_i)$:
 - i. Let $\text{tr} \subseteq \text{tr}_{\text{MT}}$ be the query-answer pairs for f_{MT} between the previous iteration and the current iteration (or the beginning if this is the first iteration).
 - ii. Compute $((\Pi_1, \text{td}_1), \dots, (\Pi_i, \text{td}_i)) := \text{MT}.\text{MultiExtract}((\mathbf{r}_1, \dots, \mathbf{r}_i), \text{tr})$.
 - iii. For every $j \in [i]$, set the salt string $\sigma'_j := (\mathbf{r}_j, \sigma_j)$.
 - iv. Add the move-response pair $((\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i)), \rho_i)$ to $\text{tr}_{\text{IOP}}^{\text{sr}}$.
3. Output $\text{tr}_{\text{IOP}}^{\text{sr}}$.

Assume that tr^{sr} has size at most t_s . The function IP2IOPTrace can be computed as follows. First, partition the query-answer trace tr_{MT} according to the move-response trace tr^{sr} . This can be done by performing a linear pass on tr_{MT} and tr^{sr} , in time $O(|\text{tr}_{\text{MT}}| + |\text{tr}^{\text{sr}}|) = O(t_{\text{MT}} + t_s)$. Then, the number of extractions is at most $t_s \cdot k$ and the total time for all extractions is

given by the expression $\mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathsf{l}, s, t_{\text{MT}}, t_{\$} \cdot k)$ where \mathbf{et}_{MT} is the (total) time complexity of $\text{MT}.\text{MultiExtract}$ in Lemma 18.5.6. Therefore, the total running time is at most

$$O(t_{\text{MT}} + t_{\$}) + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathsf{l}, s, t_{\text{MT}}, t_{\$} \cdot k).$$

Proof of Lemma 25.2.5. We argue that the IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$ in Construction 25.2.6 satisfies the probability statement in the lemma. It suffices to prove the following upper bound:

$$\Delta \left(\left\{ (\mathbf{x}, b, \mathbf{tr}_{\text{IOP}}^{\text{sr}}, \mathbf{tr}_{\text{MT}}, (\rho_i)_{i \in [k]}) \middle| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\mathbf{tr}_{\text{sr}}, \mathbf{tr}_{\text{MT}}} \\ \mathbf{Game}_{\text{IP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathbf{P}}_i^{\text{sr}})^{f_{\text{MT}}}) \\ b \leftarrow \mathcal{V}_i^{f_{\text{MT}}}(\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \\ \mathbf{tr}_{\text{IOP}}^{\text{sr}} := \text{IP2IOPTrace}(\mathbf{tr}_{\text{sr}}, \mathbf{tr}_{\text{MT}}) \end{array} \right\}, \left\{ (\mathbf{x}, b \wedge b_{\text{MT}}, \mathbf{tr}_{\text{IOP}}^{\text{sr}}, \mathbf{tr}_{\text{MT}}, (\rho_i)_{i \in [k]}) \middle| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\mathbf{tr}_{\text{IOP}}^{\text{sr}}, \mathbf{tr}_{\text{MT}}, b_{\text{MT}}} \\ \mathbf{Game}_{\text{IOP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathbf{P}}_i^{\text{sr}}))^{f_{\text{MT}}}) \\ b \leftarrow \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) \end{array} \right\} \right) \leq \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathsf{l}, (t_{\$} + 1) \cdot k, k). \quad (25.3)$$

Above, on the right-side, $\mathbf{tr}_{\text{IOP}}^{\text{sr}}$ is the move-response trace of the IOP state-restoration game, \mathbf{tr}_{MT} is the query-answer trace to the oracle f_{MT} , and b_{MT} is the internal variable defined in Item 9.

We begin by considering the left-side distribution. Recall that $(\alpha_i)_{i \in [k]} = (\mathbf{rt}_i)_{i \in [k]}$ and that $\alpha_{k+1} = ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]}$. Moreover, \mathcal{V}_i accepts if and only if the IOP verifier \mathbf{V}_{IOP} accepts and every invocation of the Merkle commitment checking algorithm $\text{MT}.\text{Check}$ accepts. Therefore, the left-side distribution equals to:

$$\left\{ (\mathbf{x}, b \wedge b_{\text{MT}}, \mathbf{tr}_{\text{IOP}}^{\text{sr}}, \mathbf{tr}_{\text{MT}}, (\rho_i)_{i \in [k]}) \middle| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\mathbf{tr}_{\text{sr}}, \mathbf{tr}_{\text{MT}}} \\ \mathbf{Game}_{\text{IP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathbf{P}}_i^{\text{sr}})^{f_{\text{MT}}}) \\ \text{parse } (\alpha_i)_{i \in [k]} \text{ as } (\mathbf{rt}_i)_{i \in [k]} \\ \text{parse } \alpha_{k+1} \text{ as } ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]} \\ b \leftarrow \mathbf{V}_{\text{IOP}}^{[Q_i, \mathbf{a}_i]_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) \\ b_{\text{MT}} \leftarrow \wedge_{i \in [k]} \text{MT}.\text{Check}^{f_{\text{MT}}}(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) \\ \mathbf{tr}_{\text{IOP}}^{\text{sr}} := \text{IP2IOPTrace}(\mathbf{tr}_{\text{sr}}, \mathbf{tr}_{\text{MT}}) \end{array} \right\}.$$

Define the event E as follows:

$$\left[\begin{array}{l} b_{\text{MT}} = 0 \\ \vee \wedge_{i \in [k]} \Pi_i[Q_i] = \mathbf{a}_i \end{array} \quad \left| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}_{\text{MT}}} \\ \text{Game}_{\text{IP}}^{\text{sr}}(\lambda + s, \text{rnd}, (\tilde{\mathcal{P}}_i^{\text{sr}})^{f_{\text{MT}}}) \\ \text{parse } (\alpha_i)_{i \in [k]} \text{ as } (\mathbf{rt}_i)_{i \in [k]} \\ \text{parse } \alpha_{k+1} \text{ as } ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]} \\ b \leftarrow \mathbf{V}_{\text{IOP}}^{[\mathbf{Q}_i, \mathbf{a}_i]_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) \\ b_{\text{MT}} \leftarrow \wedge_{i \in [k]} \text{MT.Check}^{f_{\text{MT}}}(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) \\ ((\Pi_i, \mathbf{td}_i))_{i \in [k]} := \text{MT.MultiExtract}((\mathbf{rt}_i)_{i \in [k]}, \text{tr}_{\text{MT}}) \end{array} \right. \right].$$

By Claim 1.2.10, it suffices to upper bound the probability of \overline{E} and to show that the two distributions in Equation 25.3 are identical when conditioned on E .

Conditioned on the event E : either $b_{\text{MT}} = 0$, in which case $b \wedge b_{\text{MT}} = 0$ in both sides; or $b_{\text{MT}} = 1$ and $\wedge_{i \in [k]} \Pi_i[Q_i] = \mathbf{a}_i$ (every extracted IOP string agrees with the relevant query answers), in which case $\mathbf{V}_{\text{IOP}}^{[\mathbf{Q}_i, \mathbf{a}_i]_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]})$ (the IOP verifier returns the same output), so b is the same on both sides. Therefore, the following two distributions are equivalent:

$$\left\{ \begin{array}{l} (\mathbf{x}, b \wedge b_{\text{MT}}, \text{tr}_{\text{IOP}}^{\text{sr}}, \text{tr}_{\text{MT}}, (\rho_i)_{i \in [k]}) \\ \text{conditioned on} \\ E \end{array} \quad \left| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}_{\text{sr}}, \text{tr}_{\text{MT}}} \\ \text{Game}_{\text{IP}}^{\text{sr}}(\lambda + s, \text{rnd}, (\tilde{\mathcal{P}}_i^{\text{sr}})^{f_{\text{MT}}}) \\ \text{parse } (\alpha_i)_{i \in [k]} \text{ as } (\mathbf{rt}_i)_{i \in [k]} \\ \text{parse } \alpha_{k+1} \text{ as } ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]} \\ b \leftarrow \mathbf{V}_{\text{IOP}}^{[\mathbf{Q}_i, \mathbf{a}_i]_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) \\ b_{\text{MT}} \leftarrow \wedge_{i \in [k]} \text{MT.Check}^{f_{\text{MT}}}(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) \\ \text{tr}_{\text{IOP}}^{\text{sr}} := \text{IP2IOPTrace}(\text{tr}_{\text{sr}}, \text{tr}_{\text{MT}}) \end{array} \right. \right\}$$

and

$$\left\{ \begin{array}{l} (\mathbf{x}, b \wedge b_{\text{MT}}, \text{tr}_{\text{IOP}}^{\text{sr}}, \text{tr}_{\text{MT}}, (\rho_i)_{i \in [k]}) \\ \text{conditioned on} \\ E \end{array} \quad \left| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}_{\text{sr}}, \text{tr}_{\text{MT}}} \\ \text{Game}_{\text{IP}}^{\text{sr}}(\lambda + s, \text{rnd}, (\tilde{\mathcal{P}}_i^{\text{sr}})^{f_{\text{MT}}}) \\ \text{parse } (\alpha_i)_{i \in [k]} \text{ as } (\mathbf{rt}_i)_{i \in [k]} \\ \text{parse } \alpha_{k+1} \text{ as } ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]} \\ ((\Pi_i, \mathbf{td}_i))_{i \in [k]} := \text{MT.MultiExtract}((\mathbf{rt}_i)_{i \in [k]}, \text{tr}_{\text{MT}}) \\ b \leftarrow \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) \\ b_{\text{MT}} \leftarrow \wedge_{i \in [k]} \text{MT.Check}^{f_{\text{MT}}}(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) \\ \text{tr}_{\text{IOP}}^{\text{sr}} := \text{IP2IOPTrace}(\text{tr}_{\text{sr}}, \text{tr}_{\text{MT}}) \end{array} \right. \right\}.$$

The tuple $(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ has the same distribution as the tuple in the output of the IOP state-restoration game $(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ with the IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_i^{\text{sr}})$, as can be seen by inspection of Construction 25.2.6. Moreover, by the construction,

the trace of the IOP state-restoration game of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_i^{\text{sr}})$ is the same as the trace $\text{tr}_{\text{IOP}}^{\text{sr}} := \text{IP2IOPTrace}(\text{tr}^{\text{sr}}, \text{tr}_{\text{MT}})$, as can be seen by inspection of Construction 25.2.7. Hence, the right-side distribution equals:

$$\left\{ \begin{array}{l} (\mathbf{x}, b \wedge b_{\text{MT}}, \text{tr}_{\text{IOP}}^{\text{sr}}, \text{tr}_{\text{MT}}, (\rho_i)_{i \in [k]}) \\ \text{conditioned on } E \end{array} \middle| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}_{\text{IOP}}^{\text{sr}}, \text{tr}_{\text{MT}}} \\ \text{Game}_{\text{IOP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_i^{\text{sr}}))^{f_{\text{MT}}}) \\ \text{parse } (\alpha_i)_{i \in [k]} \text{ as } (\mathbf{rt}_i)_{i \in [k]} \\ \text{parse } \alpha_{k+1} \text{ as } ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]} \\ b \leftarrow \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) \\ b_{\text{MT}} \leftarrow \wedge_{i \in [k]} \text{MT.Check}^{f_{\text{MT}}}(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) \end{array} \right\}.$$

The above is exactly the right-side distribution of Equation 25.3, conditioned on E . Thus, by Claim 1.2.10, we are left with bounding the probability of the event \overline{E} .

The probability of \overline{E} is upper bounded by the MT extraction error (Lemma 18.5.6), and is upper bounded by the following expression:

$$\Pr \left[\begin{array}{l} b_{\text{MT}} = 1 \\ \wedge \vee_{i \in [k]} \Pi_i[Q_i] \neq \mathbf{a}_i \end{array} \middle| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}_{\text{MT}}} \\ \text{Game}_{\text{IP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathcal{P}}_i^{\text{sr}})^{f_{\text{MT}}}) \\ \text{parse } (\alpha_i)_{i \in [k]} \text{ as } (\mathbf{rt}_i)_{i \in [k]} \\ \text{parse } \alpha_{k+1} \text{ as } ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]} \\ b_{\text{MT}} \leftarrow \wedge_{i \in [k]} \text{MT.Check}^{f_{\text{MT}}}(\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i) \\ ((\Pi_i, \mathbf{td}_i))_{i \in [k]} := \text{MT.MultiExtract}((\mathbf{rt}_i)_{i \in [k]}, \text{tr}_{\text{MT}}) \end{array} \right].$$

In turn, the above probability is at most $\kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathbf{l}, (t_s + 1) \cdot k, k)$ by Lemma 18.5.6. Here the function κ_{MT} gets five parameters: λ is the security parameter, t_{MT} is the query bound of $\tilde{\mathcal{P}}_i^{\text{sr}}$, \mathbf{l} is the IOP proof length (a bound on message length), $(t_s + 1) \cdot k$ is the number of output Merkle commitments ($\tilde{\mathcal{P}}_i^{\text{sr}}$ makes at most t_s moves each containing up to k Merkle commitments and a final output containing k Merkle commitments), and k is the number of opened Merkle commitments ($\tilde{\mathcal{P}}_i^{\text{sr}}$ produces k openings). \square

Proof of Theorem 25.2.1. Lemma 25.2.5 tells us that the interactive argument $\text{IARG} = (\mathcal{P}_i, \mathcal{V}_i) = \text{iBCS}[\text{IOP}, \lambda, s]$ in Construction 24.1.1 has IP state-restoration soundness error

$$\epsilon_{\text{IP}}^{\text{sr}}(s, t_s, n) \leq \epsilon_{\text{IOP}}^{\text{sr}}(\lambda + s, t_s, n) + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathbf{l}, (t_s + 1) \cdot k, k)$$

against adversaries that make at most t_{MT} queries to the random oracle f_{MT} .

Next, consider the non-interactive argument $\mathbf{FS}_{\text{IP}}^{\star}[\text{IARG}, \lambda, s] = \text{BCS}[\text{IOP}, \lambda, s]$ in Construction 25.1.1. Fix a malicious argument prover $\tilde{\mathcal{P}}$ for $\mathbf{FS}_{\text{IP}}^{\star}[\text{IARG}, \lambda, s]$ that makes t_s queries to oracles $(f_i)_{i \in [k]}$ and t_{MT} queries to oracle f_{MT} . We now explain how to apply Lemma 16.3.3 to obtain a reduction from the malicious argument prover $\tilde{\mathcal{P}}$ to an IP state-restoration prover $\tilde{\mathcal{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}})$ for the interactive argument $\text{IARG} = (\mathcal{P}_i, \mathcal{V}_i) = \text{iBCS}[\text{IOP}, \lambda, s]$ (in which the argument prover and argument verifier have access to the random oracle f_{MT}). Recall that the transformation $\mathbf{FS}_{\text{IP}}^{\star}$ introduces random oracles $(f_i)_{i \in [k]}$ to transform an IP into a non-interactive argument,

and Lemma 16.3.3 captures the security of this transformation by providing a reduction that transforms a malicious argument prover $\tilde{\mathcal{P}}$ against the non-interactive argument into an IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}})$ for the IP state-restoration game of the underling IP. However here we apply the transformation $\mathbf{FS}_{\text{IP}}^*$ to the interactive argument $\text{IARG} = (\mathcal{P}_i, \mathcal{V}_i) = \mathbf{iBCS}[\text{IOP}, \lambda, s]$, which uses a random oracle f_{MT} . Nevertheless, Lemma 16.3.3 still works in this case because the reduction makes only black-box use of the given non-interactive argument prover $\tilde{\mathcal{P}}$, and hence it does not matter if $\tilde{\mathcal{P}}$ has access to an additional oracle f_{MT} (separate from the oracles $(f_i)_{i \in [k]}$ introduced by $\mathbf{FS}_{\text{IP}}^*$). To make explicit this difference, we use the notation $\tilde{\mathcal{P}}_i^{\text{sr}}(\tilde{\mathcal{P}})$ to refer to $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}})$ where $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ is augmented to forward queries from $\tilde{\mathcal{P}}$ to f_{MT} and return the corresponding answers.

Therefore, by Lemma 16.3.3, we deduce the following:

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| \begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ f := (f_{\text{MT}}, (f_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \\ & \leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}_i^{f_{\text{MT}}}(\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 1 \end{array} \middle| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \leftarrow \\ \text{Game}_{\text{IP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathcal{P}}_i^{\text{sr}}(\tilde{\mathcal{P}}))^{f_{\text{MT}}}) \end{array} \right] \\ & \quad + \frac{t_{\$}^2}{2^\lambda}. \end{aligned}$$

This tells us that the non-interactive argument $\mathbf{FS}_{\text{IP}}^*[\text{IARG}, \lambda, s] = \mathbf{BCS}[\text{IOP}, \lambda, s]$ in Construction 25.1.1 has adaptive soundness error

$$\epsilon_{\text{ARG}}(\lambda, t_{\$}, n) \leq \epsilon_{\text{IP}}^{\text{sr}}(s, t_{\$}, n) + \frac{t_{\$}^2}{2^\lambda}$$

where $\epsilon_{\text{IP}}^{\text{sr}}$ is the IP state-restoration soundness error of IARG .

Using the fact that $t_{\$}, t_{\text{MT}} \leq t$, we conclude that Construction 25.1.1 has adaptive soundness error

$$\epsilon_{\text{ARG}}(\lambda, t, n) \leq \epsilon_{\text{IP}}^{\text{sr}}(s, t, n) + \frac{t^2}{2^\lambda} \leq \epsilon_{\text{IOP}}^{\text{sr}}(\lambda + s, t, n) + \kappa_{\text{MT}}(\lambda, t, l, (t+1) \cdot k, k) + \frac{t^2}{2^\lambda}.$$

Finally, by Lemma 18.5.6, if $6 \cdot k \cdot (\log l + 1) \leq t$ then $\kappa_{\text{MT}}(\lambda, t, l, (t+1) \cdot k, k) \leq 2 \cdot \frac{t^2}{2^\lambda}$, which lets us conclude that in this case $\epsilon_{\text{ARG}}(\lambda, t, n) \leq \epsilon_{\text{IOP}}^{\text{sr}}(\lambda + s, t, n) + 3 \cdot \frac{t^2}{2^\lambda}$. \square

26 Additional security definitions

We prove that Construction 25.1.1 satisfies additional security definitions.

26.1 Knowledge soundness

In Construction 25.1.1, if the IOP satisfies (state-restoration) knowledge soundness then the resulting non-interactive argument satisfies adaptive knowledge soundness.

Theorem 26.1.1

Let IOP be a public-coin IOP for a relation \mathcal{R} with rewinding state-restoration knowledge soundness error $\kappa_{\text{IOP}}^{\text{sr}}$ with extraction time $\mathbf{et}_{\text{IOP}}^{\text{sr}}$ (see Definition 23.2.5), round complexity k , proof length l , and maximum randomness complexity $r_{\max} := \max_{i \in [k]} r_i$. For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{NARG} := \mathbf{BCS}[\text{IOP}, \lambda, s]$ in Construction 25.1.1 is a non-interactive argument for \mathcal{R} with rewinding knowledge soundness error κ_{ARG} with extraction time \mathbf{et}_{ARG} (see Definition 7.1.7) such that

- $\kappa_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}}(\lambda, n)) \leq \kappa_{\text{IOP}}^{\text{sr}}(s, t, n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n)) + \kappa_{\text{MT}}(\lambda, t, l, (t+1) \cdot k, k) + \frac{t^2}{2^\lambda}$, and
- $\mathbf{et}_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}}(\lambda, n), \tau_{\tilde{\mathcal{P}}}(\lambda, n)) \leq \mathbf{et}_{\text{IOP}}^{\text{sr}}(s, t, n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n), \tau'_{\tilde{\mathcal{P}}}(\lambda, n)) + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, l, s, t, (t+1) \cdot k) + O(r_{\max} \cdot k \cdot t \cdot \log t)$.

Above, $\delta'_{\tilde{\mathcal{P}}}(\lambda, n) := \delta_{\tilde{\mathcal{P}}}(\lambda, n) + \kappa_{\text{MT}}(\lambda, t, l, (t+1) \cdot k, k) + \frac{t^2}{2^\lambda}$ and $\tau'_{\tilde{\mathcal{P}}}(\lambda, n) := \tau_{\tilde{\mathcal{P}}}(\lambda, n) + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, l, s, t, (t+1) \cdot k) + O(r_{\max} \cdot k \cdot t \cdot \log t)$.

Moreover, if the IOP state-restoration extractor is straightline (see Definition 23.2.3) then the NARG extractor is also straightline (see Definition 7.1.5). In this case:

- the (straightline) knowledge soundness error is

$$\kappa_{\text{ARG}}(\lambda, t, n) \leq \kappa_{\text{IOP}}^{\text{sr}}(\lambda + s, t, n) + \kappa_{\text{MT}}(\lambda, t, l, (t+1) \cdot k, k) + \frac{t^2}{2^\lambda};$$

- the (straightline) extraction time is

$$\mathbf{et}_{\text{ARG}}(\lambda, t, n) \leq \mathbf{et}_{\text{IOP}}^{\text{sr}}(s, t, n) + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, l, s, t, (t+1) \cdot k) + O(r_{\max} \cdot k \cdot t \cdot \log t).$$

Above κ_{MT} is the MT multi-extraction error from Lemma 18.5.6. If $6 \cdot l \cdot (\log l + 1) \leq t$ then $\kappa_{\text{MT}}(\lambda, t, l, (t+1) \cdot k, k) + \frac{t^2}{2^\lambda} \leq 3 \cdot \frac{t^2}{2^\lambda}$.

We discuss two natural approaches to prove the theorem. In Section 26.1.1 we outline one approach, and in Section 26.1.2 we work out the other approach.

26.1.1 Direct approach

Theorem 26.1.1 can be proved by reducing an argument prover $\tilde{\mathcal{P}}$ to a corresponding IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$, and relying on the IOP state-restoration extractor. The lemma below captures this reduction and directly implies the theorem using the fact that $t_{\$}, t_{\text{MT}} \leq t$.

Lemma 26.1.2. *There exists an extractor \mathcal{E} and an IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ such that, for every $(t_{\text{MT}}, t_{\$})$ -query argument prover $\tilde{\mathcal{P}}$, the following holds:*

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f = (f_{\text{MT}}, (f_i)_{i \in [k]}) \leftarrow \mathcal{U}((\lambda, (\mathbf{r}_i)_{i \in [k]})) \\ (\mathbf{x}, \pi) \xrightarrow{\text{tr}} \tilde{\mathcal{P}}^f \\ b \xleftarrow{\text{tr}_{\mathcal{V}}} \mathcal{V}^f(\mathbf{x}, \pi) \\ \mathbf{w} \leftarrow \mathcal{E}(\mathbf{x}, \pi, \text{tr}, \text{tr}_{\mathcal{V}}, \tilde{\mathcal{P}}) \end{array} \right] \\ & \leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}_{\text{IOP}}^{\text{sr}}} \\ \text{Game}_{\text{IOP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}))^f_{\text{MT}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IOP}}^{\text{sr}}(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \text{tr}_{\text{IOP}}^{\text{sr}}, (\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}))) \\ b \leftarrow \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) \end{array} \right] \\ & \quad + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathsf{l}, (t_{\$} + 1) \cdot k, k) + \frac{t_{\$}^2}{2^\lambda}. \end{aligned}$$

Moreover, the running time $\mathbf{et}_{\text{ARG}}(\lambda, (t_{\text{MT}}, t_{\$}), n, \delta_{\tilde{\mathcal{P}}}(\lambda, n), \tau_{\tilde{\mathcal{P}}}(\lambda, n))$ of \mathcal{E} is bounded by

$$\mathbf{et}_{\text{IOP}}^{\text{sr}}(s, t_{\$}, n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n), \tau'_{\tilde{\mathcal{P}}}(\lambda, n)) + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathsf{l}, s, t_{\text{MT}}, (t_{\$} + 1) \cdot k) + O(r_{\max} \cdot k \cdot t_{\$} \cdot \log t_{\$}).$$

Above, we have $\delta'_{\tilde{\mathcal{P}}}(\lambda, n) := \delta_{\tilde{\mathcal{P}}}(\lambda, n) + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathsf{l}, (t_{\$} + 1) \cdot k, k) + \frac{t_{\$}^2}{2^\lambda}$, $\tau'_{\tilde{\mathcal{P}}}(\lambda, n) := \tau_{\tilde{\mathcal{P}}}(\lambda, n) + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathsf{l}, s, t_{\text{MT}}, (t_{\$} + 1) \cdot k) + O(r_{\max} \cdot k \cdot t_{\$} \cdot \log t_{\$})$, and \mathbf{et}_{MT} is the time complexity of the MT extractor in Lemma 18.5.6.

We do not prove the lemma (we work out the other approach in Section 26.1.2 instead). Nevertheless below we provide the construction of \mathcal{E} from $\mathbf{E}_{\text{IOP}}^{\text{sr}}$, as it aids in understanding the security reduction.

Construction 26.1.3. Let $\mathbf{E}_{\text{IOP}}^{\text{sr}}$ be the IOP state-restoration knowledge extractor for \mathbf{V}_{IOP} . The extractor \mathcal{E} for \mathcal{V} receives as input an instance \mathbf{x} , argument string π , query-answer trace tr of the argument prover, query-answer trace $\text{tr}_{\mathcal{V}}$ of the argument verifier, and (black-box access to) the argument prover $\tilde{\mathcal{P}}$, and works as follows.

$\mathcal{E}(\mathbf{x}, \pi, \text{tr}, \text{tr}_{\mathcal{V}}, \tilde{\mathcal{P}})$:

1. Parse the argument string π as a tuple $((\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i, \tau_i))_{i \in [k]}$.
2. Initialize an empty list tr^{sr} of move-response pairs.
3. Consider query-answer pairs in the order they appear in tr . For each query x in tr to the oracle f_i (for $i \in [k]$), do the following:
 - a) Let $\rho_i \in \{0, 1\}^{r_i}$ be the answer to the query x in tr .
 - b) Let tr_{MT} be the query-answer pairs for f_{MT} between the previous query to one of $(f_i)_{i \in [k]}$ and the current query (or the beginning if this is the first such query).

- c) If $i = 1$ and x looks like a query $(\mathbf{x}, \mathbf{rt}_1, \tau_1)$:
 - i. Run the Merkle extractor to obtain an IOP string: $(\Pi_1, \mathbf{td}_1) := \text{MT}.\text{MultiExtract}(\mathbf{rt}_1, \mathbf{tr}_{\text{MT}})$.
 - ii. Set the salt string $\sigma_1 := (\mathbf{rt}_1, \tau_1)$.
 - iii. Add the move-response pair $((\mathbf{x}, \Pi_1, \sigma_1), \rho_i)$ to \mathbf{tr}^{sr} .
- d) If $i \in \{2, \dots, k\}$ and x looks like a query $(\rho_{i-1}, \mathbf{rt}_i, \tau_i)$:
 - i. Let $\mathbf{tr}_{\text{fast}}$ be the query-answer pairs in \mathbf{tr} to $(f_i)_{i \in [k]}$ so far.
 - ii. Compute $(\mathbf{x}, (\mathbf{rt}_1, \dots, \mathbf{rt}_{i-1}), (\tau_1, \dots, \tau_{i-1})) := \text{FS}.\text{Backtrack}(i-1, \rho_{i-1}, \mathbf{tr}_{\text{fast}})$ (on abort continue to next loop iteration).
 - iii. Compute $((\Pi_1, \mathbf{td}_1), \dots, (\Pi_i, \mathbf{td}_i)) := \text{MT}.\text{MultiExtract}((\mathbf{rt}_1, \dots, \mathbf{rt}_i), \mathbf{tr}_{\text{MT}})$.
 - iv. For every $j \in [i]$ set the salt string $\sigma_j := (\mathbf{rt}_j, \tau_j)$.
 - v. Add the move-response pair $((\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i)), \rho_i)$ to \mathbf{tr}^{sr} .
- 4. Let \mathbf{tr}_{MT} be the query-answer pairs for f_{MT} in \mathbf{tr} after the last query to one of $(f_i)_{i \in [k]}$.
- 5. Run the Merkle extractor: $((\Pi_i, \mathbf{td}_i))_{i \in [k]} := \text{MT}.\text{MultiExtract}((\mathbf{rt}_i)_{i \in [k]}, \mathbf{tr}_{\text{MT}})$.
- 6. For every $i \in [k]$, set the salt string $\sigma_i := (\mathbf{rt}_i, \tau_i)$.
- 7. Parse \mathbf{tr}_v as $((x_i, \rho_i))_{i \in [k]}$.
- 8. Let $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}} := \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}})$ be the IOP state-restoration prover in Construction 25.2.3.
- 9. Compute $\mathbf{w} := \mathbf{E}_{\text{IOP}}^{\text{sr}}(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \mathbf{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}})$.
- 10. Output \mathbf{w} .

The extractor \mathcal{E} iterates over the trace \mathbf{tr} of length $t_{\text{MT}} + t_{\$}$. For each query to one of $(f_i)_{i \in [k]}$, \mathcal{E} invokes $\text{FS}.\text{Backtrack}$. Ensuring that the input $\mathbf{tr}_{\text{fast}}$ to $\text{FS}.\text{Backtrack}$ is always sorted takes a total time of $O(t_{\$} \cdot r_{\max} \log t_{\$})$. In addition, each of the $t_{\$}$ calls to $\text{FS}.\text{Backtrack}$ takes time $O(r_{\max} \cdot k \cdot \log t_{\$})$.

Moreover, \mathcal{E} performs at most $(t_{\$} + 1) \cdot k$ Merkle commitment extractions ($\tilde{\mathcal{P}}$ makes at most $t_{\$}$ queries to $(f_i)_{i \in [k]}$ each leading to up to k Merkle commitment extractions and a final output leading to k Merkle commitment extractions). The total time for all extractions is $\text{et}_{\text{MT}}(\lambda, \Sigma, l, s, t_{\text{MT}}, (t_{\$} + 1) \cdot k)$, where et_{MT} is the (total) time complexity of $\text{MT}.\text{MultiExtract}$ in Lemma 18.5.6.

Finally, \mathcal{E} runs $\mathbf{E}_{\text{IOP}}^{\text{sr}}$ with the prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}})$. By Claim 25.2.4, the failure probability of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ is $\delta'_{\tilde{\mathcal{P}}}(\lambda, n) \leq \delta_{\tilde{\mathcal{P}}}(\lambda, n) + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, l, (t_{\$} + 1) \cdot k, k) + \frac{t_{\$}^2}{2^\lambda}$, and by Lemma 25.2.2 the running time of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}})$ is $\tau'_{\tilde{\mathcal{P}}}(\lambda, n) \leq \tau_{\tilde{\mathcal{P}}}(\lambda, n) + \text{et}_{\text{MT}}(\lambda, \Sigma, l, s, t_{\text{MT}}, (t_{\$} + 1) \cdot k) + O(r_{\max} \cdot k \cdot t_{\$} \cdot \log t_{\$})$. Hence the time to run $\mathbf{E}_{\text{IOP}}^{\text{sr}}$ is $\text{et}_{\text{IOP}}^{\text{sr}}(s, t_{\$}, n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n), \tau'_{\tilde{\mathcal{P}}}(\lambda, n))$.

Overall, the running time of \mathcal{E} , denoted by $\text{et}_{\text{ARG}}(\lambda, (t_{\text{MT}}, t_{\$}), n, \delta_{\tilde{\mathcal{P}}}(\lambda, n), \tau_{\tilde{\mathcal{P}}}(\lambda, n))$, is at most

$$\text{et}_{\text{IOP}}^{\text{sr}}(s, t_{\$}, n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n), \tau'_{\tilde{\mathcal{P}}}(\lambda, n)) + \text{et}_{\text{MT}}(\lambda, \Sigma, l, s, t_{\text{MT}}, (t_{\$} + 1) \cdot k) + O(r_{\max} \cdot k \cdot t_{\$} \cdot \log t_{\$}).$$

26.1.2 Modular approach

We prove Theorem 26.1.1 via a modular approach, similarly to the modular approach we took to prove Theorem 25.2.1. Recall from Equation 25.2 that the non-interactive argument $(\mathcal{P}, \mathcal{V}) = \mathbf{BCS}[\text{IOP}, \lambda, s]$ in Construction 25.1.1 is the result of applying (the faster variant of) the Fiat–Shamir transformation $\mathbf{FS}_{\text{IP}}^{\star}[\text{IARG}, \lambda, s]$ in Construction 15.1.1 to the interactive argument $\text{IARG} = (\mathcal{P}_i, \mathcal{V}_i) = \mathbf{iBCS}[\text{IOP}, \lambda, s]$ in Construction 24.1.1.

In light of Equation 25.2, we will explain how Lemma 16.3.3 tells us that the *knowledge* soundness error of the non-interactive argument $(\mathcal{P}, \mathcal{V}) = \mathbf{BCS}[\text{IOP}, \lambda, s]$ in Construction 25.1.1

is upper bounded by the IP state-restoration *knowledge* soundness error of the interactive argument $\text{IARG} = (\mathcal{P}_i, \mathcal{V}_i) = \text{iBCS}[\text{IOP}, \lambda, s]$ in Construction 24.1.1. We are thus left to upper bound the IP state-restoration knowledge soundness error of the interactive argument $\text{IARG} = (\mathcal{P}_i, \mathcal{V}_i)$. We do this in the lemma below, and conclude the proof of the theorem at the end of the section.

In the lemma below, we run the IP state-restoration game $\text{Game}_{\text{IP}}^{\text{sr}}$ with an *argument* prover $\tilde{\mathcal{P}}_i^{\text{sr}}$ that has query access to a random oracle f_{MT} . We use the notation $\xleftarrow{\text{tr}^{\text{sr}}, \text{tr}}$ to denote two traces: tr^{sr} is the move-response trace of $\tilde{\mathcal{P}}_i^{\text{sr}}$ in the IP state-restoration game; and tr is the query-answer trace of $\tilde{\mathcal{P}}_i^{\text{sr}}$ to the oracle f_{MT} . Moreover, we assume that the traces contain timing information, which allows us to order the queries by time of execution among both traces together (i.e., we can know if a given entry in tr occurred before or after a given entry in tr^{sr}).

Lemma 26.1.4. *Here $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ refers to the IOP state-restoration prover in Construction 25.2.6. There exists an IP state-restoration extractor $\mathbf{E}_{\text{IP}}^{\text{sr}}$ such that, for every t_{MT} -query argument prover $\tilde{\mathcal{P}}_i^{\text{sr}}$ that makes at most $t_{\$}$ moves in an IP state-restoration game, the following holds for every instance size bound $n \in \mathbb{N}$:*

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}, \text{tr}_{\text{MT}}} \\ \text{Game}_{\text{IP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathcal{P}}_i^{\text{sr}})^{f_{\text{MT}}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IP}}^{\text{sr}}(\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \text{tr}^{\text{sr}}, \text{tr}_{\text{MT}}, \tilde{\mathcal{P}}_i^{\text{sr}}) \\ b \leftarrow \mathcal{V}_i^{f_{\text{MT}}}(\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \end{array} \right] \\ \leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}_{\text{IOP}}^{\text{sr}}} \\ \text{Game}_{\text{IOP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_i^{\text{sr}}))^{f_{\text{MT}}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IOP}}^{\text{sr}}(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \text{tr}_{\text{IOP}}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_i^{\text{sr}})) \\ b \leftarrow \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) \end{array} \right] \\ + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathsf{l}, (t_{\$} + 1) \cdot k, k). \end{math>$$

Moreover, the running time of the extractor $\mathbf{E}_{\text{IP}}^{\text{sr}}$ is bounded by

$$\mathbf{et}_{\text{IOP}}^{\text{sr}}(s, t_{\$}, n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n), \tau'_{\tilde{\mathcal{P}}}(\lambda, n)) + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathsf{l}, s, t_{\text{MT}}, (t_{\$} + 1) \cdot k) + O(t_{\text{MT}} + t_{\$}).$$

Above:

- $\delta'_{\tilde{\mathcal{P}}}(\lambda, n) := \delta_{\tilde{\mathcal{P}}}(\lambda, n) + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathsf{l}, (t_{\$} + 1) \cdot k, k)$,
- $\tau'_{\tilde{\mathcal{P}}}(\lambda, n) := \tau_{\tilde{\mathcal{P}}}(\lambda, n) + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathsf{l}, s, t_{\text{MT}}, (t_{\$} + 1) \cdot k)$, and
- \mathbf{et}_{MT} is the time complexity of the MT extractor in Lemma 18.5.6.

Construction 26.1.5. Let $\mathbf{E}_{\text{IOP}}^{\text{sr}}$ be the IOP state-restoration knowledge extractor for \mathbf{V}_{IOP} . Let $\text{MT}.\text{MultiExtract}$ be the MT extractor for multiple commitments from Lemma 18.5.6 in Section 18.5.2; below we use the notation for $\text{MT}.\text{MultiExtract}$ introduced in Remark 18.5.8. The IP state-restoration extractor $\mathbf{E}_{\text{IP}}^{\text{sr}}$ receives as input the final output $(\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ of the IP state-restoration game, the move-response trace tr^{sr} of $\tilde{\mathcal{P}}_i^{\text{sr}}$, its query-answer trace tr_{MT} with the random oracle f_{MT} , and black-box access to $\tilde{\mathcal{P}}_i^{\text{sr}}$ itself, and works as follows.

$\mathbf{E}_{\text{IP}}^{\text{sr}}(\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \mathbf{tr}^{\text{sr}}, \mathbf{tr}_{\text{MT}}, \tilde{\mathcal{P}}_i^{\text{sr}})$:

1. Compute $\mathbf{tr}_{\text{IOP}}^{\text{sr}} \leftarrow \text{IP2IOPTrace}(\mathbf{tr}^{\text{sr}}, \mathbf{tr}_{\text{MT}})$ (see Construction 25.2.7).
2. Parse $(\alpha_i)_{i \in [k]}$ as $(\mathbf{rt}_i)_{i \in [k]}$ and α_{k+1} as $((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]}$.
3. Let $\mathbf{tr} \subseteq \mathbf{tr}_{\text{MT}}$ be the query-answer pairs for f_{MT} performed after the last move in \mathbf{tr}^{sr} .
4. Run the Merkle extractor to obtain IOP strings:

$$((\Pi_i, \mathbf{td}_i))_{i \in [k]} := \text{MT.MultiExtract}((\mathbf{rt}_i)_{i \in [k]}, \mathbf{tr}).$$
5. For every $i \in [k]$, set the salt string $\sigma'_i := (\mathbf{rt}_i, \sigma_i)$.
6. Let $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}} := \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_i^{\text{sr}})$ be the IOP state-restoration prover in Construction 25.2.6.
7. Compute $\mathbf{w} \leftarrow \mathbf{E}_{\text{IP}}^{\text{sr}}(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \mathbf{tr}_{\text{IOP}}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}).$
8. Output \mathbf{w} .

The trace has size at most $t_{\$}$ and thus the procedure IP2IOPTrace together with the final extraction runs in time $\mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathbf{l}, s, t_{\text{MT}}, (t_{\$} + 1) \cdot k) + O(t_{\text{MT}} + t_{\$})$, where \mathbf{et}_{MT} is the (total) time complexity of MT.MultiExtract in Lemma 18.5.6. Finally, $\mathbf{E}_{\text{IP}}^{\text{sr}}$ runs $\mathbf{E}_{\text{IOP}}^{\text{sr}}$ with the prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$. By Equation 25.3, the failure probability of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ is $\delta'_{\tilde{\mathcal{P}}}(\lambda, n) \leq \delta_{\tilde{\mathcal{P}}}(\lambda, n) + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathbf{l}, (t_{\$} + 1) \cdot k, k)$, and by Lemma 25.2.5 the running time of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ is $\tau'_{\tilde{\mathcal{P}}}(\lambda, n) \leq \tau_{\tilde{\mathcal{P}}}(\lambda, n) + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathbf{l}, s, t_{\text{MT}}, (t_{\$} + 1) \cdot k)$. Hence the time to run $\mathbf{E}_{\text{IP}}^{\text{sr}}$ is $\mathbf{et}_{\text{IOP}}^{\text{sr}}(s, t_{\$}, n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n), \tau'_{\tilde{\mathcal{P}}}(\lambda, n))$.

Overall, the running time of $\mathbf{E}_{\text{IP}}^{\text{sr}}$ is at most

$$\mathbf{et}_{\text{IOP}}^{\text{sr}}(s, t_{\$}, n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n), \tau'_{\tilde{\mathcal{P}}}(\lambda, n)) + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathbf{l}, s, t_{\text{MT}}, (t_{\$} + 1) \cdot k) + O(t_{\text{MT}} + t_{\$}).$$

Proof. We argue that the IP state-restoration extractor $\mathbf{E}_{\text{IP}}^{\text{sr}}$ satisfies the claimed property. By Equation 25.3 we get that:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \right] \left[\begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\mathbf{tr}^{\text{sr}}, \mathbf{tr}_{\text{MT}}} \\ \text{Game}_{\text{IP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathcal{P}}_i^{\text{sr}})^{f_{\text{MT}}}) \\ b \leftarrow \mathcal{V}_i^{f_{\text{MT}}}(\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IP}}^{\text{sr}}(\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \mathbf{tr}^{\text{sr}}, \mathbf{tr}_{\text{MT}}, \tilde{\mathcal{P}}_i^{\text{sr}}) \end{array} \right] \\ \leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \right] \left[\begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\mathbf{tr}_{\text{IOP}}^{\text{sr}}} \\ \text{Game}_{\text{IOP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_i^{\text{sr}}))^{f_{\text{MT}}}) \\ b \leftarrow \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IP}}^{\text{sr}}(\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \mathbf{tr}^{\text{sr}}, \mathbf{tr}_{\text{MT}}, \tilde{\mathcal{P}}_i^{\text{sr}}) \end{array} \right] \\ + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathbf{l}, (t_{\$} + 1) \cdot k, k). \quad (26.1) \end{math>$$

The only information that matters for the probability statement is $(\mathbf{x}, \mathbf{w}, (\Pi_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$. The subtuple $(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ has the same distribution as the information in the output of the IOP state-restoration game $(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ with the IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_i^{\text{sr}})$, as can be seen by inspection of Construction 25.2.6. We are left to argue about the witness \mathbf{w} , the output of $\mathbf{E}_{\text{IP}}^{\text{sr}}(\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \mathbf{tr}^{\text{sr}}, \mathbf{tr}_{\text{MT}}, \tilde{\mathcal{P}}_i^{\text{sr}})$. Internally, $\mathbf{E}_{\text{IP}}^{\text{sr}}$ computes $\mathbf{tr}_{\text{IOP}}^{\text{sr}} \leftarrow \text{IP2IOPTrace}(\mathbf{tr}^{\text{sr}}, \mathbf{tr}_{\text{MT}})$, which is the trace of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_i^{\text{sr}})$, and then returns

the output of $\mathbf{E}_{\text{IOP}}^{\text{sr}}(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \mathbf{tr}_{\text{IOP}}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}})$ where $(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ is the output of the IOP state-restoration game $(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ with the IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_{\text{i}}^{\text{sr}})$. Hence the probability in Equation 26.1 above equals the following probability:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\mathbf{tr}_{\text{IOP}}^{\text{sr}}} \\ \text{Game}_{\text{IOP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_{\text{i}}^{\text{sr}}))^{f_{\text{MT}}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IOP}}^{\text{sr}}(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \mathbf{tr}_{\text{IOP}}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_{\text{i}}^{\text{sr}})) \\ b \leftarrow \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) \end{array} \right].$$

□

Construction 26.1.6. Let $\mathbf{E}_{\text{IOP}}^{\text{sr}}$ be the IOP state-restoration knowledge extractor for \mathbf{V}_{IOP} , and let $\mathbf{E}_{\text{IP}}^{\text{sr}}$ be the IP state-restoration knowledge extractor for \mathcal{V}_{i} constructed from it in Construction 26.1.5. The extractor \mathcal{E} for \mathcal{V} receives as input an instance \mathbf{x} , argument string π , query-answer trace \mathbf{tr} of the argument prover, query-answer trace \mathbf{tr}_{v} of the argument verifier, and (black-box access to) the argument prover $\tilde{\mathcal{P}}$, and works as follows.

$\mathcal{E}(\mathbf{x}, \pi, \mathbf{tr}, \mathbf{tr}_{\text{v}}, \tilde{\mathcal{P}})$:

1. Let $\tilde{\mathcal{P}}_{\text{i}}^{\text{sr}} := \tilde{\mathcal{P}}_{\text{i}}^{\text{sr}}(\tilde{\mathcal{P}})$ be the IP state-restoration prover in Construction 16.3.1.
2. Let $\mathbf{E}_{\text{IP}}^{\text{sr}}$ be the IP state-restoration extractor in Construction 26.1.5 obtained from $\mathbf{E}_{\text{IOP}}^{\text{sr}}$.
3. Parse the argument string π as a tuple $((\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i, \tau_i))_{i \in [k]}$.
4. Set the prover messages $(\alpha_i)_{i \in [k]} := (\mathbf{rt}_i)_{i \in [k]}$ and $\alpha_{k+1} := ((Q_i, \mathbf{a}_i, \mathbf{pf}_i))_{i \in [k]}$.
5. For every $i \in [k]$, set the salt string $\sigma_i := \tau_i$.
6. Parse the argument verifier trace \mathbf{tr}_{v} as k query-answer pairs $((x_i, \rho_i))_{i \in [k]}$.
7. Let $\mathbf{tr}_{\$}$ be the query-answer trace consisting of queries to $(f_i)_{i \in [k]}$ in \mathbf{tr} .
8. Set the move-response trace $\mathbf{tr}^{\text{sr}} := \text{FastToSRTTrace}(\mathbf{tr}_{\$})$. (See Construction 16.3.2.)
9. Let \mathbf{tr}_{MT} be the query-answer trace consisting of queries to f_{MT} in \mathbf{tr} .
10. Compute $\mathbf{w} \leftarrow \mathbf{E}_{\text{IP}}^{\text{sr}}(\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \mathbf{tr}^{\text{sr}}, \mathbf{tr}_{\text{MT}}, \tilde{\mathcal{P}}_{\text{i}}^{\text{sr}})$.
11. Output \mathbf{w} .

Proof of Theorem 26.1.1. Lemma 26.1.4 tells us that the interactive argument $\text{IARG} = (\mathcal{P}_{\text{i}}, \mathcal{V}_{\text{i}}) = \text{iBCS}[\text{IOP}, \lambda, s]$ in Construction 24.1.1 has IP rewinding state-restoration knowledge soundness error

$$\kappa_{\text{IP}}^{\text{sr}}(s, t_{\$}, n, \delta_{\tilde{\mathcal{P}}_{\text{i}}^{\text{sr}}}(s, n)) \leq \kappa_{\text{IOP}}^{\text{sr}}(\lambda + s, t_{\$}, n, \delta_{\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_{\text{i}}^{\text{sr}})}(s, n)) + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathbf{l}, (t_{\$} + 1) \cdot k, k),$$

against adversaries $\tilde{\mathcal{P}}_{\text{i}}^{\text{sr}}$ that make at most t_{MT} queries to the random oracle f_{MT} . Moreover, by Equation 25.3 we have that $\delta_{\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}(\tilde{\mathcal{P}}_{\text{i}}^{\text{sr}})}(s, n) \leq \delta_{\tilde{\mathcal{P}}_{\text{i}}^{\text{sr}}}(s, n) + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathbf{l}, (t_{\$} + 1) \cdot k, k)$; indeed, the failure probability depends only on the instance \mathbf{x} and the verifier's decision bit b output by each prover, which are close in statistical distance by Equation 25.3.

Next, consider the non-interactive argument $\mathbf{FS}_{\text{IP}}^{\star}[\text{IARG}, \lambda, s] = \mathbf{BCS}[\text{IOP}, \lambda, s]$ in Construction 25.1.1. Fix a malicious argument prover $\tilde{\mathcal{P}}$ for $\mathbf{FS}_{\text{IP}}^{\star}[\text{IARG}, \lambda, s]$ that makes $t_{\$}$ queries to oracles $(f_i)_{i \in [k]}$ and t_{MT} queries to oracle f_{MT} . (In particular, $t_{\$} + t_{\text{MT}} \leq t$.) We now explain how

to apply Lemma 16.3.3 to obtain a reduction from the malicious argument prover $\tilde{\mathcal{P}}$ to an IP state-restoration prover $\tilde{\mathcal{P}}_i^{\text{sr}}(\tilde{\mathcal{P}})$ for the interactive argument $\text{IARG} = (\mathcal{P}_i, \mathcal{V}_i) = \text{iBCS}[\text{IOP}, \lambda, s]$ (in which the argument prover and argument verifier have access to the random oracle f_{MT}). Recall that the transformation FS_{IP}^* introduces random oracles $(f_i)_{i \in [k]}$ to transform an IP into a non-interactive argument, and Lemma 16.3.3 captures the security of this transformation by providing a reduction that transforms a malicious argument prover $\tilde{\mathcal{P}}$ against the non-interactive argument into an IP state-restoration prover $\tilde{\mathcal{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}})$ for the IP state-restoration game of the underling IP. However here we apply the transformation FS_{IP}^* to the interactive argument $\text{IARG} = (\mathcal{P}_i, \mathcal{V}_i) = \text{iBCS}[\text{IOP}, \lambda, s]$, which uses a random oracle f_{MT} . Nevertheless, Lemma 16.3.3 still works in this case because the reduction makes only black-box use of the given non-interactive argument prover $\tilde{\mathcal{P}}$, and hence it does not matter if $\tilde{\mathcal{P}}$ has access to an additional oracle f_{MT} (separate from the oracles $(f_i)_{i \in [k]}$ introduced by FS_{IP}^*). To make explicit this difference, we use the notation $\tilde{\mathcal{P}}_i^{\text{sr}}(\tilde{\mathcal{P}})$ to refer to $\tilde{\mathcal{P}}_{\text{IP}}^{\text{sr}}(\tilde{\mathcal{P}})$ where $\tilde{\mathcal{P}}_{\text{IP}}^{\text{sr}}$ is augmented to forward queries from $\tilde{\mathcal{P}}$ to f_{MT} and return the corresponding answers.

Therefore, by Lemma 16.3.3, we deduce the following:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \right] \left[\begin{array}{l} (f_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ f := (f_{\text{MT}}, (f_i)_{i \in [k]}) \\ (\mathbf{x}, \pi) \xrightarrow{\text{tr}} \tilde{\mathcal{P}} f \\ b \xleftarrow{\text{tr}_v} \mathcal{V}^f(\mathbf{x}, \pi) \\ \mathbf{w} \leftarrow \mathcal{E}(\mathbf{x}, \pi, \text{tr}, \text{tr}_v, \tilde{\mathcal{P}}) \end{array} \right] \\ \leq \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \right] \left[\begin{array}{l} f_{\text{MT}} \leftarrow \mathcal{U}(\lambda) \\ \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}, \text{tr}_{\text{MT}}} \\ \text{Game}_{\text{IP}}^{\text{sr}}(\lambda + s, \mathbf{rnd}, (\tilde{\mathcal{P}}_i^{\text{sr}}(\tilde{\mathcal{P}}))^{f_{\text{MT}}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IP}}^{\text{sr}}(\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \text{tr}^{\text{sr}}, \text{tr}_{\text{MT}}, \tilde{\mathcal{P}}_i^{\text{sr}}(\tilde{\mathcal{P}})) \\ b \leftarrow \mathcal{V}_i^{f_{\text{MT}}}(\mathbf{x}, (\alpha_i)_{i \in [k+1]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \end{array} \right] \\ + \frac{t_{\$}^2}{2^\lambda}. \end{math>$$

This tells us that the non-interactive argument $\text{FS}_{\text{IP}}^*[\text{IARG}, \lambda, s] = \text{BCS}[\text{IOP}, \lambda, s]$ in Construction 25.1.1 has rewinding knowledge soundness error

$$\begin{aligned} \kappa_{\text{ARG}}(\lambda, t_{\$}, n, \delta_{\tilde{\mathcal{P}}}) &\leq \kappa_{\text{IP}}^{\text{sr}} \left(\lambda + s, t_{\$}, n, \delta_{\tilde{\mathcal{P}}_i^{\text{sr}}(\tilde{\mathcal{P}})} \right) + \frac{t_{\$}^2}{2^\lambda} \\ &\leq \kappa_{\text{IP}}^{\text{sr}} \left(\lambda + s, t_{\$}, n, \delta_{\tilde{\mathcal{P}}} + \frac{t_{\$}^2}{2^\lambda} \right) + \frac{t_{\$}^2}{2^\lambda}, \end{aligned}$$

where $\kappa_{\text{IP}}^{\text{sr}}$ is the IP rewinding state-restoration knowledge soundness error of IARG .

We conclude that Construction 25.1.1 has adaptive knowledge soundness error

$$\begin{aligned} \kappa_{\text{ARG}}(\lambda, (t_{\$}, t_{\text{MT}}), n, \delta_{\tilde{\mathcal{P}}}) &\leq \kappa_{\text{IP}}^{\text{sr}} \left(\lambda + s, t_{\$}, n, \delta_{\tilde{\mathcal{P}}} + \frac{t_{\$}^2}{2^\lambda} \right) + \frac{t_{\$}^2}{2^\lambda} \\ &\leq \kappa_{\text{IOP}}^{\text{sr}} \left(\lambda + s, t_{\$}, n, \delta_{\tilde{\mathcal{P}}} + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathbf{l}, (t_{\$} + 1) \cdot k, k) + \frac{t_{\$}^2}{2^\lambda} \right) \end{aligned}$$

$$+ \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathsf{I}, (t_{\$} + 1) \cdot \mathsf{k}, \mathsf{k}) + \frac{t_{\$}^2}{2^\lambda}.$$

Finally, by Lemma 18.5.6, if $6 \cdot \mathsf{k} \cdot (\log \mathsf{I} + 1) \leq t_{\text{MT}}$ then $\kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathsf{I}, (t_{\$} + 1) \cdot \mathsf{k}, \mathsf{k}) \leq 2 \cdot \frac{t_{\text{MT}}^2}{2^\lambda}$, which lets us conclude that in this case $\kappa_{\text{ARG}}(\lambda, t, n, \delta_{\tilde{\mathcal{P}}}) \leq \kappa_{\text{IOP}}^{\text{sr}}\left(\lambda + s, t, n, \delta_{\tilde{\mathcal{P}}} + 3 \cdot \frac{t_{\$}^2}{2^\lambda}\right) + 3 \cdot \frac{t_{\$}^2}{2^\lambda}$.

We discuss the running time of \mathcal{E} in Construction 26.1.6, which is dominated by the time to run $\text{FastToSRTTrace}(\mathsf{tr}_{\$})$ and the time to run $\mathbf{E}_{\text{IP}}^{\text{sr}}$. The time to run $\text{FastToSRTTrace}(\mathsf{tr}_{\$})$ is $O(\mathsf{r}_{\max} \cdot \mathsf{k} \cdot t_{\$} \cdot \log t_{\$})$. The time to run $\mathbf{E}_{\text{IP}}^{\text{sr}}$ depends on the failure probability and running time of $\tilde{\mathcal{P}}_i^{\text{sr}}(\tilde{\mathcal{P}})$. By Lemma 16.3.3, the failure probability of $\tilde{\mathcal{P}}_i^{\text{sr}}(\tilde{\mathcal{P}})$ is at most the failure probability of $\tilde{\mathcal{P}}$ plus $\frac{t_{\$}^2}{2^\lambda}$. The running time of $\tilde{\mathcal{P}}_i^{\text{sr}}(\tilde{\mathcal{P}})$ equals the running time of $\tilde{\mathcal{P}}$ plus $O(\mathsf{r}_{\max} \cdot \mathsf{k} \cdot t_{\$} \cdot \log t_{\$})$. By Lemma 26.1.4, the running time of $\mathbf{E}_{\text{IP}}^{\text{sr}}$ is bounded by

$$\mathbf{et}_{\text{IOP}}^{\text{sr}}(s, t_{\$}, n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n), \tau'_{\tilde{\mathcal{P}}}(\lambda, n)) + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathsf{I}, s, t_{\text{MT}}, (t_{\$} + 1) \cdot \mathsf{k}) + O(t_{\text{MT}} + t_{\$}).$$

Above, $\tau'_{\tilde{\mathcal{P}}}(\lambda, n) \leq \tau_{\tilde{\mathcal{P}}}(\lambda, n) + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathsf{I}, s, t_{\text{MT}}, (t_{\$} + 1) \cdot \mathsf{k}) + O(\mathsf{r}_{\max} \cdot \mathsf{k} \cdot t_{\$} \cdot \log t_{\$})$ and $\delta'_{\tilde{\mathcal{P}}}(\lambda, n) \leq \delta_{\tilde{\mathcal{P}}}(\lambda, n) + \kappa_{\text{MT}}(\lambda, t_{\text{MT}}, \mathsf{I}, (t_{\$} + 1) \cdot \mathsf{k}, \mathsf{k}) + \frac{t_{\$}^2}{2^\lambda}$. Overall, the running time of \mathcal{E} is

$$\mathbf{et}_{\text{IOP}}^{\text{sr}}(s, t_{\$}, n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n), \tau'_{\tilde{\mathcal{P}}}(\lambda, n)) + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathsf{I}, s, t_{\text{MT}}, (t_{\$} + 1) \cdot \mathsf{k}) + O(\mathsf{r}_{\max} \cdot \mathsf{k} \cdot t_{\$} \cdot \log t_{\$} + t_{\text{MT}}).$$

Finally, recalling that $t_{\$} + t_{\text{MT}} \leq t$, we get the upper bound

$$\mathbf{et}_{\text{IOP}}^{\text{sr}}(s, t, n, \delta'_{\tilde{\mathcal{P}}}(\lambda, n), \tau'_{\tilde{\mathcal{P}}}(\lambda, n)) + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathsf{I}, s, t, (t + 1) \cdot \mathsf{k}) + O(\mathsf{r}_{\max} \cdot \mathsf{k} \cdot t \cdot \log t).$$

The straightline case The above analysis directly specializes to the straightline case. Suppose that $\mathbf{E}_{\text{IOP}}^{\text{sr}}$ is a straightline extractor (Definition 23.2.3): it is a deterministic algorithm that does not need access to the cheating prover $\tilde{\mathcal{P}}_{\text{IOP}}^{\text{sr}}$. Then $\mathbf{E}_{\text{IP}}^{\text{sr}}$ in Construction 26.1.5 (which is based on $\mathbf{E}_{\text{IOP}}^{\text{sr}}$) is a straightline extractor (Definition 13.2.3): it is a deterministic algorithm that does not need access to the cheating prover $\tilde{\mathcal{P}}_i^{\text{sr}}$. In turn, \mathcal{E} in Construction 26.1.6 (which is based on $\mathbf{E}_{\text{IP}}^{\text{sr}}$) is a straightline extractor (Definition 7.1.5): it is a deterministic algorithm that does not need access to the cheating prover $\tilde{\mathcal{P}}$. In this case the knowledge soundness error simplifies to the following expression (as it no longer depends on the failure probability of the prover):

$$\kappa_{\text{ARG}}(\lambda, t, n) \leq \kappa_{\text{IOP}}^{\text{sr}}(\lambda + s, t, n) + \kappa_{\text{MT}}(\lambda, t, \mathsf{I}, (t + 1) \cdot \mathsf{k}, \mathsf{k}) + \frac{t^2}{2^\lambda}.$$

Moreover, the (straightline) extraction time also simplifies to the following expression (as it no longer depends on the failure probability or running time of the prover):

$$\mathbf{et}_{\text{ARG}}(\lambda, t, n) \leq \mathbf{et}_{\text{IOP}}^{\text{sr}}(s, t, n) + \mathbf{et}_{\text{MT}}(\lambda, \Sigma, \mathsf{I}, s, t, (t + 1) \cdot \mathsf{k}) + O(\mathsf{r}_{\max} \cdot \mathsf{k} \cdot t \cdot \log t).$$

□

26.2 Zero knowledge

In Construction 25.1.1, if the public-coin IOP satisfies honest-verifier zero knowledge (and the privacy parameter s of the construction is large enough) then the resulting non-interactive argument satisfies adaptive zero knowledge.

Theorem 26.2.1

Let IOP be a public-coin IOP for a relation \mathcal{R} with round complexity k , proof length l , query complexity q , and honest-verifier zero-knowledge error z_{IOP} (see Definition 23.1.8). For every security parameter $\lambda \in \mathbb{N}$ and privacy parameter $s \in \mathbb{N}$, $\text{NARG} := \text{BCS}[\text{IOP}, \lambda, s]$ in Construction 25.1.1 is a non-interactive argument for \mathcal{R} with adaptive zero-knowledge error z_{ARG} (see Definition 7.1.8) such that

$$z_{\text{ARG}}(\lambda, t, n) \leq z_{\text{IOP}}(n) + z_{\text{MT}}(\lambda, l(n), s, q(n), t) + \frac{t}{2^s},$$

where z_{MT} is the hiding error of the Merkle commitment scheme from Lemma 18.6.3.

Construction 26.2.2. The simulator is an algorithm $\mathcal{S}^f(\mathbf{x})$ that works as follows. Below we denote by \mathbf{S}_{IOP} the honest-verifier zero-knowledge simulator of the IOP (see Definition 23.1.8).

- $\mathcal{S}^f(\mathbf{x})$:

1. Sample a simulated view of the IOP verifier: $(\mathbf{x}, (\rho_i)_{i \in [k]}, (Q_i)_{i \in [k]}, (\mathbf{a}_i)_{i \in [k]}) \leftarrow \mathbf{S}_{\text{IOP}}(\mathbf{x})$.
2. For $i = 1, \dots, k$:
 - a) Run the MT simulator with f_{MT} as the oracle: $(\mathbf{rt}_i, \mathbf{pf}_i) \leftarrow \text{MT.Simulate}^{f_{\text{MT}}}(Q_i, \mathbf{a}_i)$.
 - b) Sample a random salt $\tau_i \in \{0, 1\}^s$.
 - c) Set $x_i := \begin{cases} (\mathbf{x}, \mathbf{rt}_1, \tau_1) & \text{if } i = 1 \\ (\rho_{i-1}, \mathbf{rt}_i, \tau_i) & \text{if } i > 1 \end{cases}$.
 - d) Set the query-answer list $\mu_i := \{(x_i, \rho_i)\}$, to be used to program oracle f_i .
3. Set the argument string $\pi := ((\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i, \tau_i))_{i \in [k]}$.
4. Set the query-answer list $\mu_{\text{MT}} := \emptyset$. (The oracle f_{MT} is not programmed.)
5. Set $\mu := (\mu_{\text{MT}}, (\mu_i)_{i \in [k]})$.
6. Output (π, μ) .

Note that \mathcal{S} programs the oracles $(f_i)_{i \in [k]}$ at one point each and does not program the oracle f_{MT} (and, conversely, \mathcal{S} queries f_{MT} but does not query any of $(f_i)_{i \in [k]}$).

Proof. Fix a query bound $t \in \mathbb{N}$, t -query admissible adversary \mathcal{A} , and instance bound $n \in \mathbb{N}$. We wish to upper bound the statistical distance between the output of \mathcal{A} in the real world and in the simulated world. For that, we introduce two hybrid simulators (that take the witness as input):

- $\mathcal{S}_{\text{H2}}^f(\mathbf{x}, \mathbf{w})$ acts as $\mathcal{S}^f(\mathbf{x})$ except that it samples the view in Item 1 according to the real view $(\mathbf{x}, \rho, (Q_i)_{i \in [k]}, (\mathbf{a}_i)_{i \in [k]}) \leftarrow \text{View}_{\text{IOP}}(\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}}, \mathbf{x}, \mathbf{w})$ of the IOP.
- $\mathcal{S}_{\text{H1}}^f(\mathbf{x}, \mathbf{w})$ acts as $\mathcal{S}_{\text{H2}}^f(\mathbf{x}, \mathbf{w})$ except that it samples the Merkle commitments using MT.Commit and opening proofs using MT.Open (instead of via MT.Simulate).

We list the real-world, first-hybrid-world, second-hybrid-world, and simulated-world distributions, making explicit the operations underlying the relevant algorithms.

1. The real-world distribution:

$$\mathcal{D}_{\text{real}} := \left\{ \text{out} \mid \begin{array}{l} f \leftarrow \mathcal{U}((\lambda, (r_i)_{i \in [k]})) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}_f^f \\ \pi \leftarrow \mathcal{P}^f(\mathbf{x}, \mathbf{w}) \\ \text{out} \leftarrow \mathcal{A}^f(\text{aux}, \pi) \end{array} \right\} \equiv \left\{ \text{out} \mid \begin{array}{l} f = (f_{\text{MT}}, (f_i)_{i \in [k]}) \leftarrow \mathcal{U}((\lambda, (r_i)_{i \in [k]})) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}_f^f \\ \text{For } i = 1, \dots, k: \\ \bullet (\Pi_i, \text{aux}_i) := \begin{cases} \mathbf{P}_{\text{IOP}}(\mathbf{x}, \mathbf{w}) & \text{if } i = 1 \\ \mathbf{P}_{\text{IOP}}(\text{aux}_{i-1}, \rho_{i-1}) & \text{if } i > 1 \end{cases} \\ \bullet (\mathbf{r}_i, \mathbf{td}_i) \leftarrow \text{MT.Commit}^{f_{\text{MT}}}(\Pi_i) \\ \bullet \tau_i \leftarrow \{0, 1\}^s \\ \bullet x_i := \begin{cases} (\mathbf{x}, \mathbf{r}_1, \tau_1) & \text{if } i = 1 \\ (\rho_{i-1}, \mathbf{r}_i, \tau_i) & \text{if } i > 1 \end{cases} \\ \bullet \rho_i := f_i(x_i) \\ (Q_i)_{i \in [k]} := \text{queries of } \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) \\ (\mathbf{a}_i)_{i \in [k]} := (\Pi_i[Q_i])_{i \in [k]} \\ (\mathbf{pf}_i)_{i \in [k]} := (\text{MT.Open}^{f_{\text{MT}}}(\mathbf{td}_i, Q_i))_{i \in [k]} \\ \pi := ((\mathbf{r}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i, \tau_i))_{i \in [k]} \\ \text{out} \leftarrow \mathcal{A}^f(\text{aux}, \pi) \end{array} \right\}.$$

2. The first-hybrid-world distribution:

$$\mathcal{D}_{\text{H1}} := \left\{ \text{out} \mid \begin{array}{l} f \leftarrow \mathcal{U}((\lambda, (r_i)_{i \in [k]})) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}_f^f \\ (\pi, \mu) \leftarrow \mathcal{S}_{\text{H1}}^f(\mathbf{x}, \mathbf{w}) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\text{aux}, \pi) \end{array} \right\} \equiv \left\{ \text{out} \mid \begin{array}{l} f = (f_{\text{MT}}, (f_i)_{i \in [k]}) \leftarrow \mathcal{U}((\lambda, (r_i)_{i \in [k]})) \\ (\mathbf{x}, \mathbf{w}, \text{aux}) \leftarrow \mathcal{A}_f^f \\ \text{For } i = 1, \dots, k: \\ \bullet (\Pi_i, \text{aux}_i) := \begin{cases} \mathbf{P}_{\text{IOP}}(\mathbf{x}, \mathbf{w}) & \text{if } i = 1 \\ \mathbf{P}_{\text{IOP}}(\text{aux}_{i-1}, \rho_{i-1}) & \text{if } i > 1 \end{cases} \\ \bullet (\mathbf{r}_i, \mathbf{td}_i) \leftarrow \text{MT.Commit}^{f_{\text{MT}}}(\Pi_i) \\ \bullet \tau_i \leftarrow \{0, 1\}^s \\ \bullet x_i := \begin{cases} (\mathbf{x}, \mathbf{r}_1, \tau_1) & \text{if } i = 1 \\ (\rho_{i-1}, \mathbf{r}_i, \tau_i) & \text{if } i > 1 \end{cases} \\ \bullet \rho_i \leftarrow \{0, 1\}^{r_i} \\ \bullet \mu_i := \{(x_i, \rho_i)\} \\ (Q_i)_{i \in [k]} := \text{queries of } \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) \\ (\mathbf{a}_i)_{i \in [k]} := (\Pi_i[Q_i])_{i \in [k]} \\ (\mathbf{pf}_i)_{i \in [k]} := (\text{MT.Open}^{f_{\text{MT}}}(\mathbf{td}_i, Q_i))_{i \in [k]} \\ \mu_{\text{MT}} := \emptyset \\ \mu := (\mu_{\text{MT}}, (\mu_i)_{i \in [k]}) \\ \pi := ((\mathbf{r}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i, \tau_i))_{i \in [k]} \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\text{aux}, \pi) \end{array} \right\}.$$

3. The second-hybrid-world distribution:

$$\mathcal{D}_{H2} := \left\{ \text{out} \mid \begin{array}{l} f \leftarrow \mathcal{U}((\lambda, (r_i)_{i \in [k]})) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ (\pi, \mu) \leftarrow \mathcal{S}_{H2}^f(\mathbf{x}, \mathbf{w}) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right\} \equiv \left\{ \text{out} \mid \begin{array}{l} f = (f_{MT}, (f_i)_{i \in [k]}) \leftarrow \mathcal{U}((\lambda, (r_i)_{i \in [k]})) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ \text{For } i = 1, \dots, k: \\ \quad \bullet (\Pi_i, \mathbf{aux}_i) := \begin{cases} \mathbf{P}_{IOP}(\mathbf{x}, \mathbf{w}) & \text{if } i = 1 \\ \mathbf{P}_{IOP}(\mathbf{aux}_{i-1}, \rho_{i-1}) & \text{if } i > 1 \end{cases} \\ \quad \bullet \rho_i \leftarrow \{0, 1\}^{r_i} \\ (Q_i)_{i \in [k]} := \text{queries of } \mathbf{V}_{IOP}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) \\ (\mathbf{a}_i)_{i \in [k]} := (\Pi_i[Q_i])_{i \in [k]} \\ \text{For } i = 1, \dots, k: \\ \quad \bullet (\mathbf{rt}_i, \mathbf{pf}_i) := MT.\text{Simulate}^{f_{MT}}(Q_i, \mathbf{a}_i) \\ \quad \bullet \tau_i \leftarrow \{0, 1\}^s \\ \quad \bullet x_i := \begin{cases} (\mathbf{x}, \mathbf{rt}_1, \tau_1) & \text{if } i = 1 \\ (\rho_{i-1}, \mathbf{rt}_i, \tau_i) & \text{if } i > 1 \end{cases} \\ \quad \bullet \mu_i := \{(x_i, \rho_i)\} \\ \mu_{MT} := \emptyset \\ \mu := (\mu_{MT}, (\mu_i)_{i \in [k]}) \\ \pi := ((\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i, \tau_i))_{i \in [k]} \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right\}.$$

4. The simulated-world distribution:

$$\mathcal{D}_{sim} := \left\{ \text{out} \mid \begin{array}{l} f \leftarrow \mathcal{U}(\lambda, (r_i)_{i \in [k]}) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ (\pi, \mu) \leftarrow \mathcal{S}^f(\mathbf{x}) \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right\} \equiv \left\{ \text{out} \mid \begin{array}{l} f = (f_{MT}, (f_i)_{i \in [k]}) \leftarrow \mathcal{U}((\lambda, (r_i)_{i \in [k]})) \\ (\mathbf{x}, \mathbf{w}, \mathbf{aux}) \leftarrow \mathcal{A}^f \\ (\mathbf{x}, (\rho_i)_{i \in [k]}, (Q_i)_{i \in [k]}, (\mathbf{a}_i)_{i \in [k]}) \leftarrow \mathbf{S}_{IOP}(\mathbf{x}) \\ \text{For } i = 1, \dots, k: \\ \quad \bullet \tau_i \leftarrow \{0, 1\}^s \\ \quad \bullet x_i := \begin{cases} (\mathbf{x}, \mathbf{rt}_1, \tau_1) & \text{if } i = 1 \\ (\rho_{i-1}, \mathbf{rt}_i, \tau_i) & \text{if } i > 1 \end{cases} \\ \quad \bullet \mu_i := \{(x_i, \rho_i)\} \\ (\mathbf{rt}_i, \mathbf{pf}_i)_{i \in [k]} := (MT.\text{Simulate}^{f_{MT}}(Q_i, \mathbf{a}_i))_{i \in [k]} \\ \mu_{MT} := \emptyset \\ \mu := (\mu_{MT}, (\mu_i)_{i \in [k]}) \\ \pi := ((\mathbf{rt}_i, Q_i, \mathbf{a}_i, \mathbf{pf}_i, \tau_i))_{i \in [k]} \\ \text{out} \leftarrow \mathcal{A}^{f[\mu]}(\mathbf{aux}, \pi) \end{array} \right\}.$$

Next we analyze the statistical difference between these distributions.

- *Real versus first hybrid.* We analyze the statistical distance between \mathcal{D}_{real} and \mathcal{D}_{H1} .

The two distributions only differ in that the hybrid simulator \mathcal{S}_H programs the oracles with freshly sampled IOP challenges, whereas the argument prover \mathcal{P} does not program the oracles (it sets the IOP challenges to be the answers of the random oracles to certain queries).

For every $i \in [k]$, the distribution of the query-answer pair (x_i, ρ_i) for the oracle f_i is the same in both cases. However, \mathcal{A} has access to the oracles $f = (f_i)_{i \in [k]}$ before they are programmed. Thus, \mathcal{A} can distinguish between the two distributions *only* if \mathcal{A} queries an oracle f_i at x_i before f_i is programmed there (and also after f_i is programmed).

Suppose that \mathcal{A} makes t_i queries to oracle f_i ; note that $\sum_{i \in [k]} t_i \leq t$. For every $i \in [k]$, the query x_i includes a salt $\tau_i \in \{0, 1\}^s$ that is sampled independently and uniformly at random. Hence, for every $i \in [k]$, the probability that \mathcal{A} queries f_i at x_i before τ_i is sampled is at most $\frac{t_i}{2^s}$. We conclude that the probability that there exists $i \in [k]$ for which this happens is at most

$$\sum_{i \in [k]} \frac{t_i}{2^s} \leq \frac{t}{2^s}.$$

In particular, this is an upper bound on the statistical distance between $\mathcal{D}_{\text{real}}$ and \mathcal{D}_H .

- *First hybrid versus second hybrid.* We analyze the statistical distance between \mathcal{D}_{H1} and \mathcal{D}_{H2} .

The two distributions only differ in how $((rt_i, pf_i))_{i \in [k]}$ is sampled: \mathcal{S}_{H1} samples the Merkle commitments using $\text{MT}.\text{Commit}$ and opening proofs using $\text{MT}.\text{Open}$; whereas \mathcal{S}_{H2} samples simulated Merkle commitments and opening proofs using $\text{MT}.\text{Simulate}$.

The adversary \mathcal{A} is admissible, which implies that $|\mathbf{x}| \leq n$. By the hiding property of Merkle commitment schemes (Lemma 18.6.3), the statistical distance between \mathcal{D}_{H1} and \mathcal{D}_{H2} contributed by (rt_i, pf_i) is upper bounded by $z_{\text{MT}}(\lambda, l_i(\mathbf{x}), s, q_i(\mathbf{x}), t)$, where l_i and q_i are the proof length and query complexity in the i -th round of the IOP.

Overall the statistical distance between \mathcal{D}_{H1} and \mathcal{D}_{H2} is upper bounded by

$$\max \left\{ \sum_{i \in [k]} z_{\text{MT}}(\lambda, l_i(\mathbf{x}), s, q_i(\mathbf{x}), t) \mid \mathbf{x}, \mathbf{w} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \text{ and } (\mathbf{x}, \mathbf{w}) \in \mathcal{R} \right\}.$$

Then, if the error bound z_{MT} is a superadditive function in the proof length and query complexity (which in particular also implies that it is a monotonic function in these parameters), we can upper bound the above expression by the following one:

$$z_{\text{MT}}(\lambda, l(n), s, q(n), t).$$

(As discussed in Remark 18.6.10, the error bound z_{MT} that we provide is superadditive.)

- *Second hybrid versus simulated.* We analyze the statistical distance between \mathcal{D}_{H2} and \mathcal{D}_{sim} .

The two distributions only differ in that \mathcal{S}_{H2} samples a real IOP view and \mathcal{S} samples a simulated IOP view. By the zero-knowledge property of the IOP, the statistical distance between the IOP views for $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ is at most $z_{\text{IOP}}(\mathbf{x})$. Since \mathcal{A} outputs $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ with $|\mathbf{x}| \leq n$ (as \mathcal{A} is admissible), the statistical distance between \mathcal{D}_{H2} and \mathcal{D}_{sim} is upper bounded by $z_{\text{IOP}}(n)$.

Adding these error terms yields the statistical difference claimed in the lemma. \square

Part VII

Practical Considerations

Overview

This part includes miscellaneous discussions that inform the use of arguments in the ROM in practice. We summarize known constructions of probabilistic proofs. We discuss how to use our theorems in order to set security parameters. We discuss practical considerations for Merkle commitment schemes. Finally we discuss the strong soundness notions of special soundness and round-by-round soundness, both of which imply state-restoration soundness.

27 Constructions of probabilistic proofs	275
27.1 Interactive proofs	275
27.2 Probabilistically checkable proofs	276
27.3 Interactive oracle proofs	277
27.4 Strong soundness properties	278
28 Setting parameters	280
28.1 Asymptotic vs. concrete security	280
28.2 Security levels	281
28.3 Concrete security for argument systems	285
29 Merkle commitment scheme optimizations	291
29.1 Any message length	291
29.2 Path pruning	295
30 Special soundness	308
30.1 Extraction from forks and trees	308
30.2 Knowledge soundness for SPs	311
30.3 State-restoration knowledge soundness for SPs	314
30.4 Knowledge soundness for IPs	320
30.5 State-restoration knowledge soundness for IPs	330
31 Round-by-round soundness	351
31.1 Definition	351
31.2 RBR soundness implies SR soundness	355
31.3 RBR knowledge soundness implies SR knowledge soundness	357

27 Constructions of probabilistic proofs

This book describes *transformations* from probabilistic proofs to cryptographic proofs in the ROM. The design and analysis of suitable probabilistic proofs is *not* a goal of this book: we take probabilistic proofs as a given input to the transformations that we study.

The goal of this chapter is to summarize the main capabilities and roles of the different types of probabilistic proofs that we consider, in order to supply a modicum of context to the reader (but without aspiring to provide a survey of probabilistic proofs). The material in this chapter assumes basic notions in complexity theory (some complexity classes and reductions between computational problems). For more details on probabilistic proofs, we refer the reader to the cited references and to courses on probabilistic proofs (see, e.g., the summer schools at [CG21; Chi23]).

27.1 Interactive proofs

Interactive proofs (IPs) primarily play the role of a subroutine in the construction of succinct arguments. For example, the sumcheck protocol [LFKN92] is an IP (more precisely, an interactive reduction) that is used as a subroutine in numerous PCPs and IOPs suitable for succinct arguments. However, IPs used on their own have limitations when it comes to succinctness, as we outline below.

Characterization IPs capture the complexity class PSPACE: every language having an IP is decidable in polynomial space (a rather straightforward fact); moreover, Shamir's protocol [Sha92] is an IP for a PSPACE-complete language (a celebrated result in complexity theory).

The above characterization result, however, does not yield IPs suitable for succinct arguments because the honest prover in Shamir's protocol (and other similar protocols) is inefficient for complexity classes of interest. For example, the honest prover that reduces an NP-complete language to PSPACE (NP is contained in PSPACE) and runs Shamir's protocol has exponential running time (even when the honest prover receives a valid witness for the given NP instance).

Delegation for deterministic computations A line of works studies *doubly-efficient IPs*, which are IPs where the honest prover is efficient (i.e., runs in polynomial time) and the honest verifier is super fast (e.g., runs in quasilinear time). Doubly-efficient IPs exist for notable classes of computations. For example, the GKR protocol [GKR15] is a doubly-efficient IP for bounded-depth computations; and the RRR protocol [RRR21] is a doubly-efficient IP for bounded-space computations. These and other results show that delegation of computation via IPs is possible for limited but useful classes of computations. A characterization of which languages admit doubly-efficient IPs remains an open problem, but unfortunately, doubly-efficient IPs do not exist for all deterministic computations (given plausible complexity-theoretic assumptions). For example, such results rule out IPs for T -step deterministic computations where the verifier

runs in time $o(T)$ (i.e., saving in running time compared to simply checking the deterministic computation itself).

What about nondeterministic computations? There do not exist IPs that yield delegation of computation for nondeterministic computations (given plausible complexity-theoretic assumptions), such as NP languages. Informally, the prover in an IP for a nondeterministic computation must communicate to the verifier at least as much information as the nondeterministic witness [GH98; GVV02], which precludes any savings in verification time compared to directly checking the witness.

Nevertheless, IPs have other benefits for nondeterministic computations: zero knowledge. On the one hand, statistical zero knowledge is possible only for languages in $\text{AM} \cap \text{coAM}$ [For89; AH87] (which rules output NP-complete languages, given plausible assumptions). On the other hand, given essentially minimal cryptographic assumptions (the existence of one-way functions), computational zero knowledge is possible for all of NP [GMW91], and indeed all of PSPACE [BGGHKMR88]. However, these zero knowledge IPs are not succinct for the aforementioned reasons, and thus they are not useful towards constructing succinct arguments for nondeterministic languages.

Take away IPs primarily play the role of a subroutine in the construction of succinct arguments in the ROM (they appear as building blocks of suitable PCPs or IOPs).

27.2 Probabilistically checkable proofs

Probabilistically checkable proofs (PCPs) enable strong asymptotic results about succinct arguments but (at the time of writing) are not relevant for practical realizations. Known PCP constructions have good asymptotic efficiency but bad concrete efficiency. Therefore, from the perspective of this book, PCPs primarily serve a pedagogical purpose: they underlie elegant constructions of succinct arguments (covered in Part V) that are useful to understand before studying constructions of succinct arguments based on IOPs (covered in Part VI), which do enjoy good concrete efficiency as we discuss in Section 27.3. Below, we outline the main capabilities of PCPs.

Characterization PCPs capture the complexity class NEXP: every language having a PCP is decidable in nondeterministic exponential time (a rather straightforward fact); moreover, the BFL protocol [BFL91] gives a PCP for a NEXP-complete problem (also a celebrated result), where the polynomial-time probabilistic PCP verifier makes a polynomial number of queries to a PCP string of exponential size (incurring a constant soundness error).

The above characterization result does not by itself offer a PCP system suitable for the construction of succinct arguments in the ROM, because the honest PCP prover in the BFL protocol is inefficient when applied to, e.g., an NP language (the proof length would be super polynomial).

Delegation of nondeterministic computations The BFLS protocol [BFLS91], building on the BFL protocol, provides a PCP for general nondeterministic computations: the prover algorithm, given a valid witness for the nondeterministic computation, is efficient (i.e., runs in polynomial time) and the verifier algorithm is exponentially faster compared to checking

a nondeterministic witness. In more detail, for every time bound function T , the BFLS protocol gives, for the complexity class $\text{NTIME}(T)$, a PCP system where the PCP prover runs in time $\text{poly}(T)$ (outputting a PCP string of length $\text{poly}(T)$) and the PCP verifier runs in time $\text{poly}(n, \log T)$ (making $\text{poly}(\log T)$ queries).

This directly implies constructions of succinct arguments for all nondeterministic computations: (a) plugging the BFLS protocol into the Kilian transformation yields a succinct *interactive* argument for $\text{NTIME}(T)$; and (b) plugging the BFLS protocol into the Micali transformation yields a succinct *non-interactive* argument for $\text{NTIME}(T)$. In both cases, to achieve negligible soundness error, the PCP verifier is repeated several times depending on the security parameter. Moreover, the BFLS protocol has (straightline) knowledge soundness and can be made honest-verifier zero knowledge with little overhead. In this case, the Micali transformation yields a zkSNARK for $\text{NTIME}(T)$, which is a powerful feasibility result.

Beyond BFLS PCPs with improved asymptotics are known, which in turn improves the asymptotics of succinct arguments (or else offer tradeoffs). For example, the PCP Theorem [AS98; ALMSS98] achieves polynomial proof length and *constant query complexity* for every NP language (which corresponds to $\text{NTIME}(T)$ when T is polynomially bounded); this theorem is elegantly reproved via mostly combinatorial techniques in [Din07]. For $\text{NTIME}(T)$ the proof length can be improved to $T \cdot \text{poly}(\log T)$ [BS08] while keeping the verifier time complexity $\text{poly}(n, \log T)$ (and query complexity $\text{poly}(\log T)$) [BGHSV05] and reducing query complexity to $O(1)$ [Din07; Mie09]. Increasing alphabet size can improve soundness while maintaining small query complexity [RS97; AS03; DFKRS11; DHK15] (which is useful for succinct arguments).

Much more is known (PCPs have been studied for over three decades) than what we recall here. It suffices to say that there are many beautiful constructions of PCPs improving different parameters (soundness error, query complexity, proof length, and so on), including for other applications such as hardness of approximation (which have somewhat different needs compared to succinct arguments).

Take away PCPs achieve excellent asymptotics for all nondeterministic computations (a rich class of computations). However it remains an open question to additionally achieve good concrete efficiency for PCPs suitable for succinct arguments. Overall, PCPs have an important pedagogical value, with the BFLS protocol underlying feasibility results for succinct arguments in the ROM.

27.3 Interactive oracle proofs

Interactive oracle proofs (IOPs) simultaneously generalize interactive proofs (IPs) and probabilistically checkable proofs (PCPs), and so anything achieved by IPs and PCPs can be achieved by IOPs. In fact IOPs go beyond that: known IOPs achieve parameter regimes that at present are out of reach of known PCPs (e.g., linear proof length) and, amazingly, IOPs also achieve good concrete efficiency. These capabilities have made IOPs a key component of practically efficient succinct arguments, and improvements to IOPs directly result in improvements to succinct arguments. There is much active research on IOPs so parts of the landscape sketched below may become dated soon.

Characterization IOPs capture the complexity class NEXP: every language having an IOP is decidable in nondeterministic exponential time (a rather straightforward fact); and a PCP is in particular an IOP, so the BFL protocol gives an IOP for a NEXP-complete problem. In particular, as in Section 27.2, this characterization result is not suitable for succinct arguments.

Delegation via IOPs While IOPs do not capture “more languages” compared to PCPs, they achieve better asymptotic efficiency compared to PCPs and, importantly, good concrete efficiency.

For example, IOPs with linear proof length are known for notable languages such as boolean circuit satisfiability [BCGRS17; RR20; CG22]. In fact, IOPs can also achieve a linear-time prover (which in particular implies linear proof length) [BCGGHJ17; BCG20; BCL22; RR22; GLSTW23]. Also notable is the fact that achieving zero knowledge for IOPs is much cheaper than for PCPs [BCGV16; BCFGRS17], and in particular preserves the concrete efficiency of the underlying IOP. The Reed–Solomon code plays an important role across many efficient IOP constructions: highly-efficient IOPs of proximity for the Reed–Solomon code have enabled a first generation of concretely efficient IOPs [BBHR18; BBHR19; BCRSVW19; COS20; ACFY24].

Numerous other works study the capabilities and limitations of IOPs, which we omit in this short discussion. IOPs are a very active research area, and the landscape of IOP results is likely to evolve further in the near future.

Take away IOPs can be concretely efficient and underlie many constructions of succinct arguments. In light of this, Part VI of this book is the culmination of many ideas and is the part likely to be most relevant for practitioners.

27.4 Strong soundness properties

This book discusses three transformations that construct non-interactive arguments in the ROM: the Fiat–Shamir transformation; the Micali transformation; and the BCS transformation. These transformations require the underlying probabilistic proof to satisfy state-restoration soundness (or knowledge soundness), which is stronger than standard soundness (or knowledge soundness). Intuitively, this is because in all these cases, a malicious argument prover can attack the underlying probabilistic proof multiple times “in its head” before outputting an argument string.

For PCPs, state-restoration soundness and knowledge soundness are tightly implied from the standard notions (see Section 19.2). However, this is not the case for *multi-round* probabilistic proofs such as IPs and IOPs (see Section 13.2.2). In principle, an IP or IOP may have small soundness error and yet have large state-restoration soundness error (and ditto for knowledge soundness). Therefore, in order to use an IP or IOP in the relevant transformation above, one must specifically prove that the IP or IOP has small state-restoration soundness error (and ditto for knowledge soundness).

Fortunately, many constructions of IPs and IOPs do have small state-restoration soundness error (and knowledge soundness error). This often follows from other strong notions of soundness that imply state-restoration soundness, such as *special soundness* (discussed in Chapter 30) and *round-by-round soundness* (discussed in Chapter 31). Here is a small selection of examples.

- The GMW protocol [GMW91], realized with a perfectly binding and computationally hiding commitment scheme, is an IP for the NP-complete problem of graph 3-colorability

that is computational zero knowledge. This protocol satisfies special soundness (with large arity). Similarly for Blum’s protocol for the NP-complete problem of graph hamiltonicity.

- The GKR protocol, which is an IP for delegating bounded-depth computations mentioned in Section 27.1, satisfies round-by-round soundness [CCHLRR18].
- The FRI protocol, which is a useful subroutine in IOP constructions, satisfies round-by-round soundness [BGKTTZ23].

In addition, many interactive arguments (IPs where soundness is computational) satisfy useful computational relaxations of special soundness. Examples include Schnorr’s protocol [Sch89] (an SP for proving in zero knowledge the knowledge of a discrete logarithm) and Bulletproofs [BBB_PWM18] (a succinct interactive argument for NP based on the hardness of discrete logarithms).

28 Setting parameters

We discuss how the definitions and results about non-interactive arguments in this book can be used to set parameters in order to achieve desired levels of concrete security in practice.

In Section 28.1 we explain the difference between asymptotic security and concrete security. In Section 28.2 we describe different notions of *security level*, and illustrate them via a simple example (the basic commitment scheme). In Section 28.3 we describe how to set parameters for the Micali transformation and the BCS transformation in order to achieve a desired security level. Similar discussions are straightforward for the other constructions that we study in this book.

28.1 Asymptotic vs. concrete security

Asymptotic security An asymptotic security guarantee for a cryptographic primitive typically takes the following form: every adversary whose resources are polynomially bounded in the security parameter can “break” the security guarantee with a probability that is negligible in the security parameter (a probability smaller than the inverse of any polynomial in the security parameter).

The meaning of “resources” depends on the setting. In our setting (unconditional security in the ROM), “resources” is the number of queries that the adversary makes to the random oracle. For example, the asymptotic security variant of (adaptive) soundness for a non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} has the following form.

For every $a, b, c \in \mathbb{N}$ there exists a threshold $\lambda_0 \in \mathbb{N}$ such that for every security parameter $\lambda \in \mathbb{N}$ with $\lambda \geq \lambda_0$ and λ^a -query malicious argument prover $\tilde{\mathcal{P}}$,

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq \lambda^b \\ \wedge \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \\ \wedge \mathcal{V}^f(\mathbf{x}, \pi) = 1 \end{array} \middle| \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{x}, \pi) \leftarrow \tilde{\mathcal{P}}^f \end{array} \right] \leq \lambda^{-c}.$$

The above definition sets the output size of the random oracle σ to equal the security parameter λ . More generally, the output size σ could be some other function of λ .

While asymptotic security guarantees suffice for the study of theoretical cryptography (e.g., to establish which cryptographic primitives imply the existence of other cryptographic primitives), they are not informative enough to set security parameters in practice. For example, given the above definition, what value should the security parameter λ have in order to ensure that the winning probability is at most 2^{-128} for every 2^{128} -query adversary that outputs an instance of size at most 2^{64} ? *The definition does not specify how λ_0 depends on a, b, c .* Maybe λ_0 is doubly exponential in $a + b + c$, in which case there is no hope to set λ to be a value of practical interest.

Concrete security A concrete security guarantee is a refined statement that includes an explicit bound on the adversary’s success probability in terms of the adversary’s resources (and any other relevant parameters).

For example, Definition 4.1.3 is the concrete security variant of (adaptive) soundness for a non-interactive argument $\text{NARG} = (\mathcal{P}, \mathcal{V})$ for a relation \mathcal{R} : the definition requires specifying an error function ϵ_{ARG} such that $\epsilon_{\text{ARG}}(\sigma, t, n)$ bounds the adversary’s success probability when the output size of the random oracle is σ , the adversary makes at most t queries, and the adversary outputs an instance of size at most n . The aforementioned definition takes the output size σ of the random oracle as the security parameter λ . More generally, the security parameter λ and instance size bound n determine, via some configuration function cnf , the output size of one or more random oracles; see Definition 7.1.4 in Section 7.1. In sum, the soundness error is a function $\epsilon_{\text{ARG}}(\lambda, t, n)$.

For a “good” error function ϵ_{ARG} , it will hold that if t and n are polynomially bounded in λ then $\epsilon_{\text{ARG}}(\lambda, t, n)$ is a negligible function in λ . This recovers asymptotic security as a special case.

All security analyses in this book follow the concrete security approach: we establish explicit upper bounds on the success probability of an adversary given certain resource bounds and capabilities. The motivation for this is that concrete security conveys more information about the cryptographic primitive compared to asymptotic security, and in particular conveys enough information about the security of a cryptographic primitive *to set the security parameter to achieve a desired concrete error bound* (a bound on the adversary’s success probability).

For example, suppose that we wish to ensure a maximum error ϵ_{\max} . Since ϵ_{ARG} is also a function of the query bound and instance size, we must specify a maximum query bound t_{\max} and instance size bound n_{\max} (and deliberately disregard what happens to the error beyond these). Then we can set λ to be large enough so that $\epsilon_{\text{ARG}}(\lambda, t_{\max}, n_{\max}) \leq \epsilon_{\max}$.¹ The choice of λ is then a function of $t_{\max}, n_{\max}, \epsilon_{\max}$ that we can usually determine given an explicit expression for the function ϵ_{ARG} .

Suppose, e.g., that $\epsilon_{\text{ARG}}(\lambda, t, n) = \frac{nt^2}{2^\lambda}$. Suppose further that we set $t_{\max} := 2^{128}$, $n_{\max} := 2^{64}$, and $\epsilon_{\max} := 2^{-128}$. These choices imply that $\epsilon_{\text{ARG}}(\lambda, t_{\max}, n_{\max}) = \frac{2^{64} \cdot (2^{128})^2}{2^\lambda} = \frac{2^{320}}{2^\lambda}$, so that $\lambda \geq 448$ ensures that $\epsilon_{\text{ARG}}(\lambda, t_{\max}, n_{\max}) \leq 2^{-128} = \epsilon_{\max}$.

28.2 Security levels

The above discussion illustrates how, in the regime of concrete security, we can determine choices of parameters for a cryptographic primitive (e.g., the output size of the random oracle) in order to achieve a desired error bound, provided concrete choices of restrictions on the adversary.

But what do these choices mean for the “security level” achieved by the cryptographic primitive?

There is no best definition for the “security level” of a cryptographic primitive. The concrete security of a cryptographic primitive depends on multiple parameters, creating a multi-dimensional “security-scape” whose different points may give incomparable guarantees.

¹The function ϵ_{ARG} is typically monotone in t and n , in which case $\epsilon_{\text{ARG}}(\lambda, t_{\max}, n_{\max}) = \max\{\epsilon_{\text{ARG}}(\lambda, t, n) : t \leq t_{\max} \wedge n \leq n_{\max}\}$. If ϵ_{ARG} is not monotone then one can consider the maximum value rather than $\epsilon_{\text{ARG}}(\lambda, t_{\max}, n_{\max})$.

Nevertheless, there are definitions that usefully distill information about a certain “security level”, as we now explain.

Challenge: security is a function of multiple parameters Given an error function $\epsilon: \mathbb{N} \rightarrow [0, 1]$, a cryptographic primitive is ϵ -secure if, for every security parameter $\lambda \in \mathbb{N}$ and resource bound $t \in \mathbb{N}$, every t -bounded adversary can break the security guarantee of the cryptographic primitive instantiated with security parameter λ with probability at most $\epsilon(\lambda, t)$. The function ϵ is non-decreasing in t (because adversaries with more resources are more powerful). Figure 28.1 illustrates two common error bound functions: linear growth in t and quadratic growth in t .

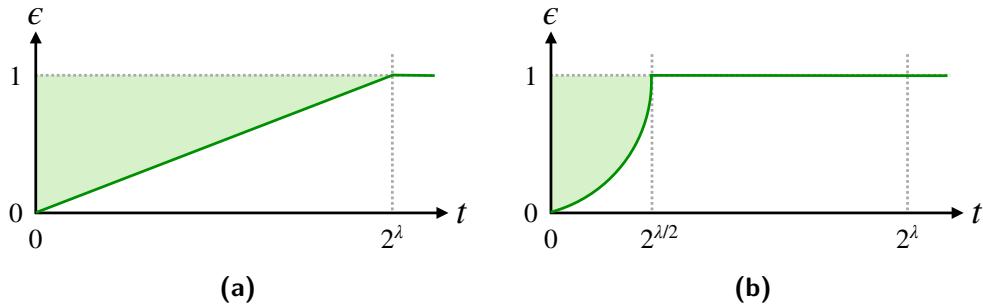


Figure 28.1: Common error bound types: $\epsilon(\lambda, t) = t/2^\lambda$ on the left and $\epsilon(\lambda, t) = t^2/2^\lambda$ on the right.

Typically, for every $\lambda \in \mathbb{N}$, $\lim_{t \rightarrow \infty} \epsilon(\lambda, t) = 1$, in other words, an adversary with enough resources can break the security guarantee of the given cryptographic primitive with probability 1, regardless of the security parameter λ used to instantiate the cryptographic primitive. Consider, for example, the case of a non-interactive argument in the ROM: for a given instance not in the language, an adversary can iterate over every possible argument string in search of an argument string that convinces the argument verifier. (As discussed in Section 6.4, this attack implies a lower bound on argument size of any non-trivial non-interactive argument.)

Therefore, any discussion of security must be limited to adversaries whose resources are bounded by an appropriate maximum bound t_{\max} . In other words, the goal is to set the security parameter λ of the cryptographic primitive in order to achieve security only against adversaries whose resources are bounded by t_{\max} . Thus, we care about the values of the error function $\epsilon(\lambda, t)$ only for $t \leq t_{\max}$.

But what does “secure” mean? It is *not* enough to know only a few values of $\epsilon(\lambda, t)$ for $t \leq t_{\max}$. For example, setting $\lambda = 256$ for concreteness, we cannot conclude anything if we know that $\epsilon(256, 2) \leq 2^{-127}$ and $\epsilon(256, 2^{127}) \leq 2^{-1}$; indeed, it might be that $\epsilon(256, t) = 2^{-1}$ for every $t \in \{3, \dots, 2^{127}\}$, which no definition should consider secure.

A reasonable answer would be to assess security by considering all values of $\epsilon(\lambda, t)$ on the interval $[t_{\max}] := \{1, \dots, t_{\max}\}$. However, this is not very informative: it is unclear how to compare the security achieved by cryptographic primitives with different error functions. For example, which of two error functions in Figure 28.2 is “more secure”?

Defining a security level An alternative answer is to distill from the error function ϵ a single number that captures the “security level” of a primitive. While it does not carry as much information as the error function ϵ itself, a single number is a more convenient way to convey

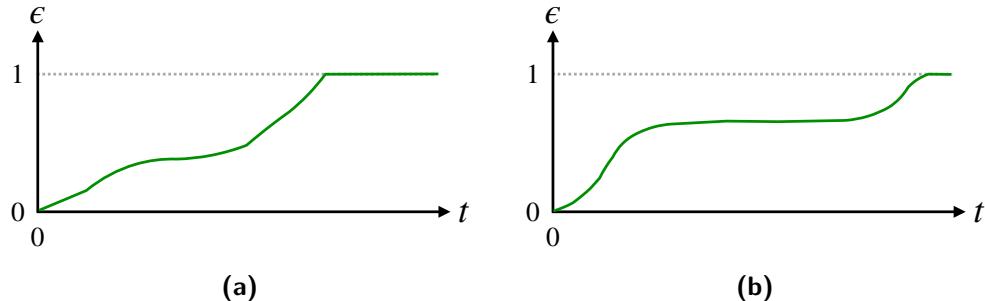


Figure 28.2: Two different error bound functions $\epsilon(\lambda, t)$. The left-side error bound is better for smaller values of t while the right-side error bound is better for larger values of t . Neither is “better” than the other.

and compare security of cryptographic primitives. Below we describe two approaches to define security level.

- *Worst-case security level.* A cryptographic primitive instantiated with security parameter λ has **worst-case security level of κ bits** if $\epsilon(\lambda, t) \leq 2^{-\kappa}$ for every $t \in [2^\kappa]$. Since we assume that ϵ is non-decreasing in t , the condition is equivalent to $\epsilon(\lambda, 2^\kappa) \leq 2^{-\kappa}$.
 - *Average-case security level.* A cryptographic primitive instantiated with security parameter λ has **average-case security level of κ bits** if $t/\epsilon(\lambda, t) \geq 2^\kappa$ for every $t \in [2^\kappa]$. The motivation behind this definition is that $t/\epsilon(\lambda, t)$ is the expected amount of resources for a successful attack: an attack with resources bounded by t succeeds (at best) with probability $\epsilon(\lambda, t)$, so in expectation a successful attack takes $t/\epsilon(\lambda, t)$ resources.

If a cryptographic primitive has κ bits of worst-case security then it has κ bits of average-case security; indeed, for every $t \in \{1, \dots, 2^\kappa\}$, $t/\epsilon(\lambda, t) \geq t2^\kappa \geq 2^\kappa$. Figure 28.3 illustrates how this implication is straightforward to see visually.

Worst-case security is stronger than average-case security but the more stringent requirement translates to a larger security parameter, and in turn a more expensive instantiation. In practice it is rather common to settle for average-case security.

Example: basic commitments Consider the basic commitment scheme $\text{CM} := \text{CM}[\sigma, \ell, s]$ described and analyzed in Chapter 17. Recall that σ is the output size of the random oracle, ℓ is the message length, and s is the salt size. We apply the above discussion to CM .

First we discuss the binding property: by Lemma 17.1.1 the probability that a t -query algorithm breaks the binding property of CM is at most $\frac{1}{2} \cdot \frac{t^2}{2^\sigma}$. We take the security parameter λ to be σ (the output size of the random oracle).

- κ bits of worst-case security. If we set $\sigma = 3\kappa - 1$ then

$$\forall t \in [2^\kappa], \frac{1}{2} \cdot \frac{t^2}{2^\sigma} \leq \frac{1}{2} \cdot \frac{(2^\kappa)^2}{2^\sigma} = \frac{2^{2\kappa}}{2^{3\kappa}} = \frac{1}{2^\kappa}.$$

- κ bits of average-case security. If we set $\sigma = 2\kappa - 1$ then

$$\forall t \in [2^\kappa], \frac{t}{\frac{1}{2} \cdot \frac{t^2}{2^\sigma}} = \frac{2 \cdot 2^\sigma}{t} \geq \frac{2 \cdot 2^\sigma}{2^\kappa} = 2^\kappa.$$

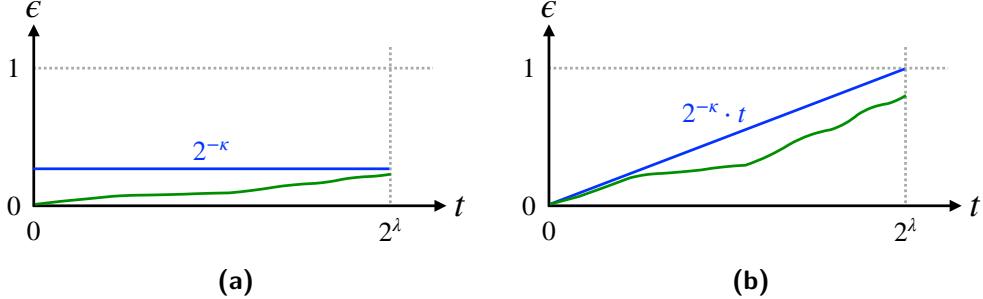


Figure 28.3: Diagrams showing the requirements for κ bits of security according to the two definitions. The left-side diagram shows worst-case security: the error bound (green line) must not go above the horizontal (blue) line at height $2^{-\kappa}$, within the interval $[2^\kappa]$. The right-side diagram shows average-case security: the error bound (green line) must not go above the oblique (blue) line, within the interval $[2^\kappa]$.

Therefore, for the binding property, to achieve (for example) $\kappa = 128$ bits of worst-case security, we need output size $\sigma = 383$, and to achieve $\kappa = 128$ bits of average-case security, we need output size $\sigma = 255$. More generally, the output size of the random oracle (here taken as the security parameter λ) for achieving worst-case security is about $1.5 \times$ bigger compared to achieving average-case security.

Next, we discuss the extractability property. Lemma 17.2.1 states that the error bound is the same as that of the binding property. This implies that the same computations as above hold for establishing security levels for the extractability property.

Finally, we discuss the hiding property: by Lemma 17.3.1 the statistical distance between a real commitment and a simulated commitment is at most $\frac{t}{2^s}$ (for the case $t < \infty$). We take the security parameter λ to be s (the salt size).

- κ bits of worst-case security. If we set $s = 2\kappa$ then

$$\forall t \in [2^\kappa], \frac{t}{2^s} \leq \frac{2^\kappa}{2^s} = \frac{2^\kappa}{2^{2\kappa}} = \frac{1}{2^\kappa}.$$

- κ bits of average-case security. If we set $s = \kappa$ then

$$\forall t \in [2^\kappa], \frac{t}{2^s} = 2^s = 2^\kappa.$$

Therefore, for the hiding property, to achieve (for example) $\kappa = 128$ bits of worst-case security we need salt size $s = 256$ and to achieve $\kappa = 128$ bits of average-case security we need salt size $s = 128$. More generally, the salt size s for achieving worst-case security is $2 \times$ bigger compared to achieving average-case security.

Additional parameters A security definition may involve other parameters affecting the error bound (besides the adversary's resource bound t).

For example, this is the case for the multi-extractability property of CM: by Lemma 17.2.2, the probability that a t -query algorithm breaks the multi-extractability property of the basic commitment CM is at most $\frac{1}{2} \cdot \frac{t^2}{2^\sigma} + \frac{n \cdot t}{2^\sigma}$, where n is the number of commitments. Taking the

security parameter λ to be the output size of the random oracle σ , the error bound is a function $\epsilon(\lambda, t, n)$.

In general, the error bound for the security property may be a function $\epsilon(\lambda, t, p)$ where p denotes some vector of parameters that specifies quantities that appear in the security definition. This affects how a security level is defined (and achieved).

We describe how to extend the definitions of security level to consider this more general case.

Denote by S the set of parameters to consider for deriving the security level. The definitions below consider the worst-case choice of parameters within the set S .

- *Worst-case security level.* A cryptographic primitive instantiated with security parameter λ has **worst-case security level of κ bits** if $\max_{p \in S} \{\epsilon(\lambda, t, p)\} \leq 2^{-\kappa}$ for every $t \in [2^\kappa]$.
- *Average-case security level.* A cryptographic primitive instantiated with security parameter λ has **average-case security level of κ bits** if $\min_{p \in S} \{t/\epsilon(\lambda, t, p)\} \geq 2^\kappa$ for every $t \in [2^\kappa]$.

We illustrate how to use these definitions by returning to the multi-extractability property of the basic commitment CM (captured by Lemma 17.2.2). Here the number of commitments n is the (single) additional parameter, and we set $S := [2^{40}]$, which means that we consider security against adversaries that output up to 2^{40} commitments (and make the minor assumption that $2^{40} \leq 2^{\kappa-1}$).

- *κ bits of worst-case security.* If we set $\sigma = 3\kappa$ then

$$\forall t \in [2^\kappa], \max_{n \in [2^{40}]} \left\{ \frac{1}{2} \cdot \frac{t^2}{2^\sigma} + \frac{n \cdot t}{2^\sigma} \right\} = \frac{1}{2} \cdot \frac{t^2}{2^\sigma} + \frac{2^{40} \cdot t}{2^\sigma} \leq \frac{2^{-1} \cdot (2^\kappa)^2 + 2^{40} \cdot 2^\kappa}{2^\sigma} \leq \frac{1}{2^\kappa}.$$

- *κ bits of average-case security.* If we set $\sigma = 2\kappa$ then

$$\forall t \in [2^\kappa], \min_{n \in [2^{40}]} \left\{ \frac{t}{\frac{1}{2} \cdot \frac{t^2}{2^\sigma} + \frac{n \cdot t}{2^\sigma}} \right\} = \frac{t}{\frac{1}{2} \cdot \frac{t^2}{2^\sigma} + \frac{2^{40} \cdot t}{2^\sigma}} = \frac{2^\sigma}{t/2 + 2^{40}} \geq \frac{2^\sigma}{2^{\kappa-1} + 2^{40}} \geq 2^\kappa.$$

Therefore, for the multi-extractability property, to achieve (for example) $\kappa = 128$ bits of worst-case security we need output size $\sigma = 384$ and to achieve $\kappa = 128$ bits of average-case security we need output size $\sigma = 256$. In general, the output size of the random oracle (here the security parameter λ) for achieving worst-case security is $1.5 \times$ bigger compared to achieving average-case security.

28.3 Concrete security for argument systems

We illustrate how to apply the ideas in Section 28.2 to achieve (worst-case or average-case) security levels for non-interactive arguments in the ROM. In Section 28.3.1, we use the example of the Micali transformation (see Chapter 21), and in Section 28.3.2 the example of the BCS transformation (see Chapter 25). Similar discussions are straightforward for other constructions studied in this book.

The security definitions for arguments in the ROM that we provide in Chapter 4 and Chapter 5 consider the simple setting where there is a single random oracle with output size σ , which can be identified as the security parameter λ . (That is, those definitions set $\lambda = \sigma$.)

More generally, as we discuss in Chapter 7, the number of random oracles and their output sizes for an argument in the ROM are determined by an oracle configuration function cnf that receives as input the security parameter λ and instance size bound n (and in this case, the argument prover and argument verifier implicitly receive λ and n). For example, in the Micali transformation, there are two oracles: one with output size λ and one with output size r (the randomness complexity of the underlying PCP verifier used to construct the non-interactive argument). This leads to security definitions for soundness, knowledge soundness, and zero knowledge in the multi-oracle setting. While in Chapter 7, we show how the multi-oracle setting can be derived from the single-oracle setting, in this chapter, it is simpler to directly consider the multi-oracle setting.

In addition, an argument in the ROM may have other parameters that must be set, possibly based on the security parameter λ ; in particular, many constructions in this book additionally have a privacy parameter s that affects (among other things) the zero-knowledge error. For example, in the Micali transformation, s determines the salt size in the underlying Merkle commitment scheme and in the Fiat–Shamir query.

Each security property of an argument has a corresponding security level. Therefore, ensuring that the argument overall achieves a desired security level entails ensuring that each security property achieves that security level. Thus, for a given argument, we discuss, in turn, the relevant properties: soundness, knowledge soundness, and zero knowledge.

28.3.1 Example: the Micali transformation

Soundness (Definition 7.1.4) The error bound is a function $\epsilon_{\text{ARG}}(\lambda, t, n)$ that depends on the security parameter λ , query bound t for the adversary, and instance size bound n .

Theorem 21.2.1 gives the following soundness error bound for the Micali transformation (under the minor parameter condition in the theorem):

$$\epsilon_{\text{ARG}}(\lambda, t, n) \leq (t+1) \cdot \epsilon_{\text{PCP}}(n) + 2 \cdot \frac{t^2}{2^\lambda}.$$

Suppose we are happy with $n \leq 2^{64}$ (instances of size up to 2^{64}).

- *Worst-case security.* Suppose that the underlying PCP is set to have soundness error ϵ_{PCP} such that $\max_{n \in [2^{64}]} \{\epsilon_{\text{PCP}}(n)\} = \epsilon_{\text{PCP}}(2^{64}) \leq 2^{-2\kappa-2}$. Setting $\lambda = 3\kappa + 2$ suffices for κ bits of worst-case security for the soundness property:

$$\begin{aligned} \forall t \in [2^\kappa], \max_{n \in [2^{64}]} \left\{ (t+1) \cdot \epsilon_{\text{PCP}}(n) + 2 \cdot \frac{t^2}{2^\lambda} \right\} &= (t+1) \cdot \epsilon_{\text{PCP}}(2^{64}) + 2 \cdot \frac{t^2}{2^\lambda} \\ &\leq (2^\kappa + 1) \cdot \frac{1}{2^{2\kappa+2}} + 2 \cdot \frac{(2^\kappa)^2}{2^\lambda} = \frac{2^\kappa + 1}{2^{2\kappa+2}} + \frac{2^{2\kappa+1}}{2^{3\kappa+2}} \\ &\leq \frac{1}{2^{\kappa+1}} + \frac{1}{2^{\kappa+1}} = \frac{1}{2^\kappa}. \end{aligned}$$

- *Average-case security.* Suppose that the underlying PCP is set to have soundness error ϵ_{PCP} such that $\max_{n \in [2^{64}]} \{\epsilon_{\text{PCP}}(n)\} = \epsilon_{\text{PCP}}(2^{64}) \leq 2^{-\kappa-1}$. Setting $\lambda = 2\kappa + 3$ suffices for κ bits of average-case security for the soundness property:

$$\forall t \in [2^\kappa], \min_{n \in [2^{64}]} \left\{ \frac{t}{(t+1) \cdot \epsilon_{\text{PCP}}(n) + 2 \cdot \frac{t^2}{2^\lambda}} \right\} = \frac{t}{(t+1) \cdot \epsilon_{\text{PCP}}(2^{64}) + 2 \cdot \frac{t^2}{2^\lambda}}$$

$$\begin{aligned}
&= \frac{2^\lambda}{(1+t^{-1}) \cdot \epsilon_{\text{PCP}}(2^{64}) \cdot 2^\lambda + 2 \cdot t} \geq \frac{2^\lambda}{(1+2^{-\kappa}) \cdot 2^{-\kappa-1} \cdot 2^\lambda + 2 \cdot 2^\kappa} \\
&= \frac{2^{2\kappa+3}}{(1+2^{-\kappa}) \cdot 2^{\kappa+2} + 2^{\kappa+1}} \geq \frac{2^{2\kappa+3}}{3/2 \cdot 2^{\kappa+2} + 1/2 \cdot 2^{\kappa+2}} = 2^\kappa.
\end{aligned}$$

Knowledge soundness (Definition 7.1.5) We consider *straightline* knowledge soundness, which applies to the Micali transformation (due to the knowledge soundness property of the underlying PCP). The error bound in this case is a function $\kappa_{\text{ARG}}(\lambda, t, n)$ that takes as input the same parameters as the error bound in the soundness case (which is a function $\epsilon_{\text{PCP}}(\lambda, t, n)$).

Theorem 22.1.1 gives the knowledge soundness error bound for the Micali transformation:

$$\kappa_{\text{ARG}}(\lambda, t, n) \leq (t+1) \cdot \kappa_{\text{PCP}}(n) + 2 \cdot \frac{t^2}{2^\lambda}.$$

This is the same expression as the soundness error, except that the PCP knowledge soundness error κ_{PCP} replaces the PCP soundness error ϵ_{PCP} . The same computations as above directly carry over, in particular showing how to set parameters to achieve, for the knowledge soundness property, κ bits of worst-case security and κ bits of average-case security.

Zero knowledge (Definition 7.1.8) The error bound for zero knowledge is a function $z_{\text{ARG}}(\lambda, t, n)$ that depends on the security parameter λ , query bound t for the adversary, and instance size bound n . Theorem 22.2.1 states that the zero-knowledge error bound for the Micali transformation is

$$z_{\text{ARG}}(\lambda, t, n) \leq z_{\text{PCP}}(n) + z_{\text{MT}}(\lambda, l(n), s, q(n), t) + \frac{t}{2^s}$$

where:

- z_{PCP} is the error bound for honest-verifier zero knowledge of the underlying PCP;
- z_{MT} is the hiding error of the Merkle commitment scheme from Lemma 18.6.3;
- l and q are the proof length and query complexity of the underlying PCP; and
- s is the privacy parameter used in the Micali transformation.

In turn, Remark 18.6.9 tells us that, if $t < \infty$, $z_{\text{MT}}(\lambda, l(n), s, q(n), t) \leq \frac{q(n) \cdot l(n) \cdot t}{2^s} + \frac{q(n) \cdot l(n) \cdot t}{2^{2\sigma}}$. Hence the zero-knowledge error bound becomes

$$z_{\text{PCP}}(n) + \frac{q(n) \cdot l(n) \cdot t}{2^s} + \frac{q(n) \cdot l(n) \cdot t}{2^{2\sigma}} + \frac{t}{2^s}.$$

In turn, if we assume that $s \leq 2\sigma$ (a minor condition that we will satisfy), the above is at most

$$z_{\text{PCP}}(n) + \frac{2 \cdot q(n) \cdot l(n) \cdot t}{2^s} + \frac{t}{2^s}.$$

We take the security parameter λ to be s (the privacy parameter). We take the (very conservative) upper bounds $q(n) \leq 2^{20}$ and $l(n) \leq 2^{40}$.

- *Worst-case security.* Suppose that the underlying PCP is set to have zero-knowledge error $\max_{n \in [2^{64}]} \{z_{\text{PCP}}(n)\} = z_{\text{PCP}}(2^{64}) \leq 2^{-\kappa-2}$. Setting $s = 2\kappa + 63$ suffices for κ bits of worst-case security for the zero-knowledge property:

$$\forall t \in [2^\kappa], \max_{n \in [2^{64}]} \left\{ z_{\text{PCP}}(n) + \frac{2 \cdot q(n) \cdot l(n) \cdot t}{2^s} + \frac{t}{2^s} \right\}$$

$$\begin{aligned}
&= z_{\text{PCP}}(2^{64}) + \frac{2 \cdot q(2^{64}) \cdot l(2^{64}) \cdot t}{2^s} + \frac{t}{2^s} \\
&\leq \frac{1}{2^{\kappa+2}} + \frac{2 \cdot 2^{20} \cdot 2^{40} \cdot 2^\kappa}{2^s} + \frac{2^\kappa}{2^s} \\
&= \frac{2^{\kappa+61}}{2^{2\kappa+63}} + \frac{2^{\kappa+61}}{2^{2\kappa+63}} + \frac{2^\kappa}{2^{2\kappa+63}} \leq \frac{1}{2^\kappa}.
\end{aligned}$$

- *Average-case security.* Suppose that the underlying PCP is set to have zero-knowledge error $\max_{n \in [2^{64}]} \{z_{\text{PCP}}(n)\} = z_{\text{PCP}}(2^{64}) \leq 2^{-\kappa-2}$. Setting $s = \kappa + 63$ suffices for κ bits of average-case security for the zero-knowledge property:

$$\begin{aligned}
&\forall t \in [2^\kappa], \min_{n \in [2^{64}]} \left\{ \frac{t}{z_{\text{PCP}}(n) + \frac{2 \cdot q(n) \cdot l(n) \cdot t}{2^s} + \frac{t}{2^s}} \right\} \\
&= \frac{t}{z_{\text{PCP}}(2^{64}) + \frac{2 \cdot q(2^{64}) \cdot l(2^{64}) \cdot t}{2^s} + \frac{t}{2^s}} \\
&= \frac{t \cdot 2^s}{z_{\text{PCP}}(2^{64}) \cdot 2^s + 2 \cdot q(2^{64}) \cdot l(2^{64}) \cdot t + t} \\
&= \frac{2^s}{z_{\text{PCP}}(2^{64}) \cdot 2^s \cdot t^{-1} + 2 \cdot q(2^{64}) \cdot l(2^{64}) + 1} \\
&\geq \frac{2^s}{2^{-\kappa-2} \cdot 2^s \cdot 1 + 2 \cdot 2^{20} \cdot 2^{40} + 1} \\
&= \frac{2^{\kappa+63}}{2^{-\kappa-2} \cdot 2^{\kappa+63} + 2 \cdot 2^{20} \cdot 2^{40} + 1} \\
&= \frac{2^{\kappa+63}}{2^{61} + 2^{61} + 1} \geq 2^\kappa.
\end{aligned}$$

28.3.2 Example: the BCS transformation

Soundness (Definition 7.1.4) The error bound is a function $\epsilon_{\text{ARG}}(\lambda, t, n)$ that depends on the security parameter λ , query bound t for the adversary, and instance size bound n .

Theorem 25.2.1 gives the following soundness error bound for the BCS transformation (under the minor parameter condition in the theorem):

$$\epsilon_{\text{ARG}}(\lambda, t, n) \leq \epsilon_{\text{IOP}}^{\text{sr}}(\lambda + s, t, n) + 3 \cdot \frac{t^2}{2^\lambda}.$$

Suppose we are happy with $n \leq 2^{64}$ (instances of size up to 2^{64}).

- *Worst-case security.* Suppose that the underlying IOP is set to have state-restoration soundness error $\epsilon_{\text{IOP}}^{\text{sr}}$ such that, for every λ, s and every $t \in [2^\kappa]$, $\max_{n \in [2^{64}]} \{\epsilon_{\text{IOP}}^{\text{sr}}(\lambda + s, t, n)\} = \epsilon_{\text{IOP}}^{\text{sr}}(\lambda + s, t, 2^{64}) \leq 2^{-\kappa-1}$. Setting $\lambda = 3\kappa + 2$ suffices for κ bits of worst-case security for the soundness property:

$$\begin{aligned}
&\forall t \in [2^\kappa], \max_{n \in [2^{64}]} \left\{ \epsilon_{\text{IOP}}^{\text{sr}}(\lambda + s, t, n) + 2 \cdot \frac{t^2}{2^\lambda} \right\} = \epsilon_{\text{IOP}}^{\text{sr}}(\lambda + s, t, 2^{64}) + 2 \cdot \frac{t^2}{2^\lambda} \\
&\leq \frac{1}{2^{\kappa+1}} + 2 \cdot \frac{(2^\kappa)^2}{2^\lambda} = \frac{1}{2^{\kappa+1}} + \frac{2^{2\kappa+1}}{2^{3\kappa+2}} \\
&= \frac{1}{2^{\kappa+1}} + \frac{1}{2^{\kappa+1}} = \frac{1}{2^\kappa}.
\end{aligned}$$

- *Average-case security.* Suppose that the underlying IOP is set to have state-restoration soundness error $\epsilon_{\text{IOP}}^{\text{sr}}$ such that, for every λ, s and every $t \in [2^\kappa]$, $\max_{n \in [2^{64}]} \{\epsilon_{\text{IOP}}^{\text{sr}}(\lambda + s, t, n)\} = \epsilon_{\text{IOP}}^{\text{sr}}(\lambda + s, t, 2^{64}) \leq 2^{-\kappa-1}$. Setting $\lambda = 2\kappa + 2$ suffices for κ bits of average-case security for the soundness property:

$$\begin{aligned} \forall t \in [2^\kappa], \min_{n \in [2^{64}]} \left\{ \frac{t}{\epsilon_{\text{IOP}}^{\text{sr}}(\lambda + s, t, n) + 2 \cdot \frac{t^2}{2^\lambda}} \right\} &= \frac{t}{\epsilon_{\text{IOP}}^{\text{sr}}(\lambda + s, t, 2^{64}) + 2 \cdot \frac{t^2}{2^\lambda}} \\ &= \frac{2^\lambda \cdot t}{\epsilon_{\text{IOP}}^{\text{sr}}(\lambda + s, t, 2^{64}) \cdot 2^\lambda + 2 \cdot t^2} \geq \frac{2^\lambda \cdot t}{2^{-\kappa-1} \cdot 2^\lambda + 2 \cdot t^2} \\ &= \frac{2^\lambda}{2^{-\kappa-1} \cdot 2^\lambda \cdot t^{-1} + 2 \cdot t} \geq \frac{2^\lambda}{2^{-\kappa-1} \cdot 2^\lambda \cdot 1 + 2 \cdot 2^\kappa} \\ &= \frac{2^{2\kappa+2}}{2^{\kappa+1} + 2^{\kappa+1}} = 2^\kappa. \end{aligned}$$

Knowledge soundness (Definition 7.1.5) We consider *straightline* knowledge soundness, which applies for the BCS transformation whenever the underlying IOP satisfies *straightline* state-restoration knowledge soundness (IOPs of practical interest usually satisfy this notion). The error bound in this case is a function $\kappa_{\text{ARG}}(\lambda, t, n)$ that takes as input the same parameters as the error bound in the soundness case (which is a function $\epsilon_{\text{ARG}}(\lambda, t, n)$).

Theorem 26.1.1 gives the knowledge soundness error bound for the BCS transformation:

$$\kappa_{\text{ARG}}(\lambda, t, n) \leq \kappa_{\text{IOP}}^{\text{sr}}(\lambda + s, t, n) + 3 \cdot \frac{t^2}{2^\lambda}.$$

This is the same expression as the soundness error, except that the IOP state-restoration knowledge soundness error $\kappa_{\text{IOP}}^{\text{sr}}$ replaces the IOP state-restoration soundness error $\epsilon_{\text{IOP}}^{\text{sr}}$. The same computations as above directly carry over, in particular showing how to set parameters to achieve, for the knowledge soundness property, κ bits of worst-case security and κ bits of average-case security.

Zero knowledge (Definition 7.1.8) The error bound for zero knowledge is a function $z_{\text{ARG}}(\lambda, t, n)$ that depends on the security parameter λ , query bound t for the adversary, and instance size bound n . Theorem 26.2.1 states that the zero-knowledge error bound for the BCS transformation is

$$z_{\text{ARG}}(\lambda, t, n) \leq z_{\text{IOP}}(n) + z_{\text{MT}}(\lambda, l(n), s, q(n), t) + \frac{t}{2^s}$$

where:

- z_{IOP} is the error bound for honest-verifier zero knowledge of the underlying IOP;
- z_{MT} is the hiding error of the Merkle commitment scheme from Lemma 18.6.3;
- l and q are the proof length and query complexity of the underlying IOP; and
- s is the privacy parameter used in the BCS transformation.

This is the same error bound as the Micali transformation, except that z_{IOP} replaces z_{PCP} . Therefore, we obtain analogous conclusions below because the same simplifications and derivations apply (which we omit). We similarly take the security parameter λ to be s (the privacy parameter) and take the (very conservative) upper bounds $q(n) \leq 2^{20}$ and $l(n) \leq 2^{40}$.

- *Worst-case security.* Suppose that the underlying IOP is set to have zero-knowledge error $\max_{n \in [2^{64}]} \{z_{\text{IOP}}(n)\} = z_{\text{IOP}}(2^{64}) \leq 2^{-\kappa-2}$. Setting $s = 2\kappa + 63$ suffices for κ bits of worst-case security for the zero-knowledge property:

$$\forall t \in [2^\kappa], \max_{n \in [2^{64}]} \left\{ z_{\text{IOP}}(n) + \frac{2 \cdot \mathbf{q}(n) \cdot \mathbf{l}(n) \cdot t}{2^s} + \frac{t}{2^s} \right\} \leq \frac{1}{2^\kappa}.$$

- *Average-case security.* Suppose that the underlying IOP is set to have zero-knowledge error $\max_{n \in [2^{64}]} \{z_{\text{IOP}}(n)\} = z_{\text{IOP}}(2^{64}) \leq 2^{-\kappa-2}$. Setting $s = \kappa + 63$ suffices for κ bits of average-case security for the zero-knowledge property:

$$\forall t \in [2^\kappa], \min_{n \in [2^{64}]} \left\{ \frac{t}{z_{\text{IOP}}(n) + \frac{2 \cdot \mathbf{q}(n) \cdot \mathbf{l}(n) \cdot t}{2^s} + \frac{t}{2^s}} \right\} \geq 2^\kappa.$$

29 Merkle commitment scheme optimizations

We discuss two optimizations of the Merkle commitment scheme MT from Chapter 18 that are often used in practice. In Section 29.1, we discuss how to support any message length without padding. In Section 29.2, we discuss *path pruning*, a method to reduce the size of Merkle opening proofs that authenticate multiple locations of the committed message vector.

29.1 Any message length

The description of the Merkle commitment scheme MT in Chapter 18 involves a binary tree over ℓ leaves, where ℓ is the number of entries in the message vector. For simplicity, ℓ was assumed to be a power of 2. In general, though, ℓ need not be a power of 2. Here, we discuss how to handle messages with any message length ℓ .

As noted in Remark 18.1.7, one option is to pad the message with dummy values (an arbitrary symbol from the alphabet Σ) until the padded message length reaches a power of 2; this, at most, doubles the length relative to the original (unpadded) message. All security analyses of MT in Chapter 18 hold for the padded message, and in particular for the original (unpadded) message.

While padding is not expensive, cheaper solutions are used in practice.

The definition of a Merkle commitment scheme in Section 18.1 straightforwardly extends to work with *any* tree, not just with a perfect binary tree (whose number of leaves is a power of 2). The number of leaves of the tree determines the message length, and the “shape” of the tree affects efficiency and security properties. Therefore, in order to support messages of arbitrary length, it suffices to define a family of trees $(T_\ell)_{\ell \in \mathbb{N}}$ where T_ℓ has precisely ℓ leaves.

We make the above discussion precise in two steps. First, we state the generalized definition of a Merkle commitment scheme for any given tree T_ℓ with ℓ leaves and comment on its efficiency and security. Then, we describe a family of binary trees $(T_\ell)_{\ell \in \mathbb{N}}$ that is often used in practice. Combining these two steps yields efficient and secure Merkle commitment schemes for any message length.

(1) Merkle commitment scheme from any tree Let $T_\ell = (V_\ell, E_\ell)$ be a (rooted) tree with ℓ leaves. We use the tree T_ℓ to describe a *generalized Merkle commitment scheme* $\text{MT} := \text{MT}[\sigma, \Sigma, T_\ell, s]$ for messages of length ℓ ; note that the tree T_ℓ is arbitrary and, in particular, need not be binary or balanced. We begin with some graph notation associated with the tree T_ℓ ; this generalizes the notation in Definition 18.1.2 for perfect binary trees to the case of any tree.

Definition 29.1.1. *We make the following notational definitions for T_ℓ :*

- *The root vertex of T_ℓ is the vertex denoted v_0 .*
- *The leaf vertices of T_ℓ are the vertices denoted $(v_i)_{i \in [\ell]}$.*
- *The path of the i -th leaf vertex v_i , denoted $\text{path}(i)$, is the list of vertices on the (unique) path from v_i to the root vertex v_0 . (The path includes the leaf vertex and the root vertex.)*

- The **copath** of the i -th leaf vertex v_i , denoted $\text{copath}(i)$, is a list of lists that contains, for each vertex v in $\text{path}(i)$, the list of siblings of v . (The root vertex has no siblings, so its empty list of siblings is omitted from $\text{copath}(i)$.)
- The **depth** of the leaf vertex v_i is $d_i := |\text{path}(i)| - 1$ (the minus 1 appears because the root vertex is defined to have depth 0 rather than depth 1). Different leaves may have different depths. The **depth of the tree** is $d := \max_{i \in [\ell]} d_i$.

We describe each of the algorithms in the generalized Merkle commitment scheme

$$\text{MT} = (\text{MT.Commit}, \text{MT.Open}, \text{MT.Check})$$

based on the (rooted) tree T_ℓ .

- $\text{MT.Commit}^f(\mathbf{m}) \rightarrow (\text{rt}, \text{td})$
 1. For each message location $i = 1, \dots, \ell$:
 - sample a salt $\tau_i \in \{0, 1\}^s$;
 - compute the (hiding) commitment $c_{v_i} := f(\mathbf{m}[i], \tau_i) \in \{0, 1\}^\sigma$.
 2. Label the leaves of T_ℓ with $(c_{v_i})_{i \in [\ell]}$. Then label each unlabeled vertex $v \in V$ as follows: if all children v^1, \dots, v^a of v are already labeled, then set the label of v to be

$$c_v := f(c_{v^1}, \dots, c_{v^a}) \in \{0, 1\}^\sigma.$$

3. Set the Merkle commitment rt to be the label of the root vertex v_0 of T_ℓ .
4. Set the salts $\boldsymbol{\tau} := (\tau_i)_{i \in [\ell]}$.
5. Set the commitments $C := (c_v)_{v \in V_\ell}$.
6. Set the opening trapdoor $\text{td} := (\boldsymbol{\tau}, C)$.
7. Output (rt, td) .

- $\text{MT.Open}^f(\text{td}, I) \rightarrow \text{pf}$.
 1. For every $i \in I$, set the authentication path for location i :

$$\text{auth}_i := (\tau_i, (c_v)_{v \in \text{copath}(i)}). \quad (29.1)$$

2. Output the opening proof $\text{pf} := (\text{auth}_i)_{i \in I}$.
- $\text{MT.Check}^f(\text{rt}, I, \mathbf{a}, \text{pf}) \rightarrow b$.
 1. Parse pf as $(\text{auth}_i)_{i \in I}$.
 2. For every $i \in I$, check that $\text{auth}_i = (\tau_i, (c_v)_{v \in \text{copath}(i)})$ authenticates the value $\mathbf{a}[i] \in \Sigma$ as the i -th opening relative to the Merkle commitment $\text{rt} \in \{0, 1\}^\sigma$ by computing $(c_v)_{v \in \text{path}(i)}$:
 - a) compute the commitment $c_{v_i} := f(\mathbf{a}[i], \tau_i)$;
 - b) for each $v \in \text{path}(i) \setminus \{v_i\}$, letting v^1, \dots, v^a be the children of v (one of which is in $\text{path}(i)$ and all others are in $\text{copath}(i)$), set the commitment

$$c_v := f(c_{v^1}, \dots, c_{v^a}) \in \{0, 1\}^\sigma.$$

- c) check that $\text{rt} = c_{v_0}$ where v_0 is the root vertex of T_ℓ .

We discuss the efficiency of each algorithm.

- The algorithm $\text{MT}.\text{Commit}$ makes ℓ queries of size $\log |\Sigma| + s$, and $|V_\ell| - \ell$ queries of varying size (the size is a multiple of σ that depends on the number of children of a given vertex). The output Merkle commitment rt has size σ and opening trapdoor td has size $s\ell + |V_\ell| \cdot \sigma$.
- The algorithm $\text{MT}.\text{Open}$ makes no oracle calls, and simply includes in the opening proof an authentication path for each index in the set I . An authentication path for location $i \in I$ has size $s + |\text{copath}(i)| \cdot \sigma$, since it includes the salt and a hash per vertex in the copath.
- The algorithm $\text{MT}.\text{Check}$, for each location $i \in I$, makes a single query of size $\log |\Sigma| + s$ and $|\text{path}(i)| - 1$ queries of varying size (the size is a multiple of σ that depends on the number of children of a given vertex).

We do not prove the security properties associated to the generalized Merkle commitment scheme. Instead, we comment on how the security properties differ from the case of Merkle commitment schemes based on perfect binary trees that we study in detail in Chapter 18.

- *Collision lemma.* Lemma 18.3.2 and its special case Lemma 18.3.1 hold as stated for the generalized Merkle commitment scheme. Extending the proofs of those lemmas (which are for a perfect binary tree) to work for an arbitrary tree T_ℓ is straightforward.
- *Binding.* Lemma 18.4.1 holds as stated for the generalized Merkle commitment scheme, with d now referring to the depth of the tree T_ℓ . The analysis carries over essentially with no changes. Worth remarking here is that the upper bound $\Pr[E \wedge \overline{E_{\text{col}}}] \leq \frac{(d+1)^2}{2^\sigma}$ continues to hold because it is still true that $|S_0|, |S_1| \leq d+1$. Indeed, $|S_0|, |S_1|$ are both at most $d_i + 1$ (i is the minimal index in $I_0 \cap I_1$ such that $\mathbf{a}_0[i] \neq \mathbf{a}_1[i]$); moreover, $d_i + 1 \leq d + 1$.
- *Extractability.* Lemma 18.5.1 (single extraction) and Lemma 18.5.6 (multi-extraction) hold as stated for the generalized Merkle commitment scheme, with d now referring to the depth of the tree T_ℓ . These extensions are also straightforward.

First, the MT extractor $\text{MT}.\text{Extract}$ (Construction 18.5.3) directly extends to work for the tree T_ℓ ; the main difference is that, when considering a query-answer pair $(x, y) \in \text{tr}_{\text{inner}}$ in the main loop, the query x should be parsed in multiple blocks, depending on the arity of the vertex under consideration (in the case of a perfect binary tree, the arity is always two). Second, the analysis of Lemma 18.5.1 carries over essentially with no changes, with the one remark that the size of the query-answer trace S_i is now at most $d_i + 1$, which in turn is at most $d + 1$. A similar consideration holds for carrying over the analysis of Lemma 18.5.6.

- *Hiding.* We discuss separately the two hiding lemmas. Here it is convenient to assume that *every internal vertex of T_ℓ (including the root vertex) has at least two children*; this is essentially without loss of generality because there is no reason for an internal vertex to have a single child (that vertex can be replaced by its child).

Lemma 18.6.1 holds as stated for the generalized Merkle commitment scheme. First we discuss the case of $t = \infty$. The analysis of this case directly carries over to show the bound

$$\ell \cdot z_{\text{CM}}(\sigma, s, \infty) + \sum_{v \in V_\ell \setminus \{v_i\}_{i \in [\ell]}} z_{\text{CM}}((a_v - 1) \cdot \sigma, \sigma, \infty)$$

where a_v is the number of children of the vertex v . The second term can be bounded by $\ell \cdot z_{\text{CM}}(\sigma, \sigma, \infty)$ because $|V_\ell \setminus \{v_i\}_{i \in [\ell]}| \leq \ell$ (the number of non-leaf vertices in V_ℓ is at most ℓ

and $z_{\text{CM}}((a_v - 1) \cdot \sigma, \sigma, \infty) \leq z_{\text{CM}}(\sigma, \sigma, \infty)$ (reducing the salt size from $(a_v - 1) \cdot \sigma$ to σ does not decrease the hiding error). Next we discuss the case of $t < \infty$. The analysis of this case directly carries over to show the bound

$$\frac{\ell \cdot t}{2^s} + \sum_{v \in V_\ell \setminus \{v_i\}_{i \in [\ell]}} \frac{t}{2^{a_v \sigma}}$$

where a_v is the number of children of the vertex v . The second term can be bounded by $\frac{\ell \cdot t}{2^{2\sigma}}$ because $|V_\ell \setminus \{v_i\}_{i \in [\ell]}| \leq \ell$ (as above) and $a_v \geq 2$ (the arity of an internal vertex is at least two).

Lemma 18.6.3 holds for the generalized Merkle commitment scheme but with a similar expression for the simulation error. Define $z_{\text{rt}}(\sigma, T, s, t)$ to be the error from the generalization of Lemma 18.6.1 for a given (rooted) tree T (not necessarily equal to T_ℓ).

- If $q = 0$ then the upper bound is (trivially) $z_{\text{MT}}(\sigma, \ell, s, q, t) := z_{\text{rt}}(\sigma, T_\ell, s, t)$.
- If $q > 0$ then the upper bound is

$$z_{\text{MT}}(\sigma, \ell, s, q, t) := \max_{I \in \binom{[\ell]}{q}} \sum_{v \in V^*(T_\ell, I)} z_{\text{rt}}(\sigma, T_\ell[v], s, t)$$

where $V^*(T_\ell, I)$ are the vertices in the copaths for I that are not in the paths for I , and $T_\ell[v]$ is the subtree of T_ℓ obtained by taking v as the root and all the vertices that appear as descendants of v .

The depth $d = \max_{i \in [\ell]} d_i$ of the tree T_ℓ directly affects efficiency and security of the generalized Merkle commitment scheme. Hence it is *beneficial to choose a tree T_ℓ that minimizes depth* (under the requirement that there be ℓ leaves). Moreover, it is *beneficial to choose a tree T_ℓ that is binary*. For example, if every internal vertex in T_ℓ has arity $a \geq 2$ then each authentication path contains $a - 1$ vertices per layer, and the number of layers in the tree is $\lceil \log_a \ell \rceil$. Higher arities are generally not advantageous for the size of opening proofs because the tree has fewer layers but each layer requires more sibling vertices for an overall bigger opening proof size. (Nevertheless, there may be other reasons to consider trees of higher arity, e.g., fewer calls to the random oracle.)

(2) Trees for any message length We describe a family of binary trees $(T_\ell)_{\ell \in \mathbb{N}}$ such that T_ℓ has depth $\lceil \log_2 \ell \rceil$.¹ In Figure 29.1 we see the first 8 trees for message lengths $\ell \in [8]$. More generally, the tree T_ℓ is recursively defined as follows.

Definition 29.1.2. Let T_1 and T_2 be as in Figure 29.1. For $\ell > 2$ define T_ℓ as the binary tree obtained by introducing a root vertex v and setting its left child to be the root vertex of T_a and its right child to be the root vertex of $T_{\ell-a}$, where $a := 2^{\lceil \log_2 \ell / 2 \rceil}$ is the largest power of two less than ℓ .

This recursive pattern is evident in Figure 29.1; for example, in T_7 the vertex $(1, 1)$ is the root vertex of $T_{2^{\lceil \log_2 7 / 2 \rceil}} = T_4$ and the vertex $(1, 2)$ is the root vertex of $T_{7-4} = T_3$. In other words, the left subtree is a perfect binary tree and the right subtree equals the tree from this

¹This is the best possible depth because any binary tree with depth less than $\lceil \log_2 \ell \rceil$ has less than ℓ leaves. This fulfills the depth-minimizing desideratum outlined above.

family appropriate for the remaining number of leaves (possibly itself a perfect binary tree if ℓ is a power of 2).

There are other naturally defined families of binary trees $(T_\ell)_{\ell \in \mathbb{N}}$ such that T_ℓ has depth $\lceil \log_2 \ell \rceil$. In Definition 29.1.3 and Definition 29.1.4 are two families that balance the number of leaves in each subtree. (The two families differ where leaves are placed.)

Definition 29.1.3. Let T_1 and T_2 be as in Figure 29.1. For $\ell > 2$ define T_ℓ as the binary tree obtained by introducing a root vertex v and setting its left child to be the root vertex of $T_{\lceil \ell/2 \rceil}$ and its right child to be the root vertex of $T_{\lfloor \ell/2 \rfloor}$.

Definition 29.1.4. For every $\ell \in \mathbb{N}$, T_ℓ is the binary tree defined as follows. Let $\ell' := 2^{\lceil \log_2 \ell \rceil}$ be smallest power of 2 that is at least ℓ (if ℓ is a power of 2 then $\ell' = \ell$). Let $T_{\ell'}^\star$ be the perfect binary tree with ℓ' leaves. Run breadth-first search (BFS) starting from the root of $T_{\ell'}$ and halt when the BFS tree has ℓ leaves. The tree T_ℓ is defined to be the BFS tree after this process halts.

The choice of which family to use in any particular context may depend on programming convenience, the distribution from which leaves to be opened are sampled, and other considerations.

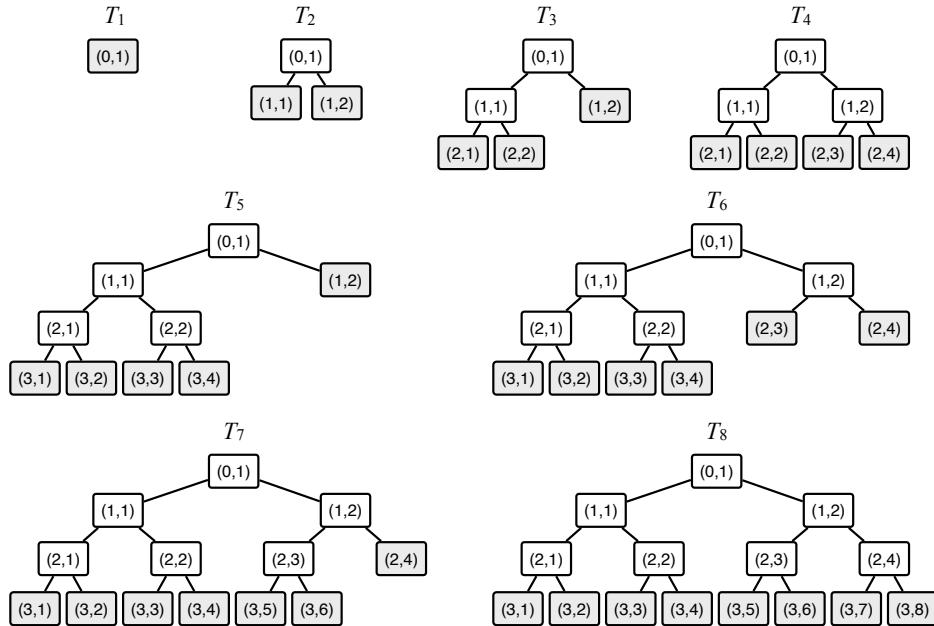


Figure 29.1: The trees $(T_\ell)_{\ell \in [8]}$ for messages lengths $\ell \in [8]$ from Definition 29.1.3. The trees T_1, T_2, T_4, T_8 coincide with the trees defined in Chapter 18 for messages whose length is a power of 2 (see Definition 18.1.1).

29.2 Path pruning

We discuss **path pruning**, a method to reduce the size of Merkle opening proofs that authenticate multiple locations of the committed message vector. For simplicity, we focus on the case of the Merkle commitment scheme discussed in Chapter 18, which involves perfect binary trees

that support message lengths that are powers of two. However, the method of path pruning straightforwardly extends to work for the generalized Merkle tree based on any tree discussed in Section 29.1.

An authentication path for location i has the following structure (Equation 18.2):

$$\mathbf{auth}_i := (\tau_i, (c_v)_{v \in \text{copath}(i)}) = (\tau_i, (c_{\bar{p}(i,j)})_{j \in \{1, \dots, d\}})$$

where τ_i is a salt string and $(c_v)_{v \in \text{copath}(i)}$ are the commitments associated to the d vertices in the copath of i . In particular, \mathbf{auth}_i has size $s + d\sigma$ because the salt string consists of s bits and each of the d commitments consists of σ bits. All of this information is necessary to validate \mathbf{auth}_i . Nevertheless, if $|I| > 1$ an opening proof $\mathbf{pf} = (\mathbf{auth}_i)_{i \in I}$ contains redundant information *across* authentication paths,² enabling a “path pruning” optimization that reduces the size of \mathbf{pf} .

In more detail, without any optimizations, the size of an opening proof $\mathbf{pf} = (\mathbf{auth}_i)_{i \in I}$ for a subset $I \subseteq [\ell]$ is $|I| \cdot (s + d\sigma)$ because it consists of $|I|$ authentication paths each of size $s + d\sigma$. The salt strings $(\tau_i)_{i \in I}$ that contribute a size of $|I| \cdot s$ are necessary and cannot be removed. (Though if privacy is not a goal, then there are no salts as $s = 0$, so the term $|I| \cdot s$ disappears.) If $|I| > 1$ then not all $|I| \cdot d\sigma$ bits contributed by $((c_v)_{v \in \text{copath}(i)})_{i \in I}$ are necessary. There are redundant commitments across multiple authentication paths that can be removed.

In Section 29.2.1, we build intuition on the optimization via some examples. Then, in Section 29.2.2, we discuss the general case of the optimization and the savings that it brings. Finally, in Section 29.2.3, we discuss how the optimization impacts security properties.

29.2.1 Examples for $\ell = 8$

Consider the Merkle commitment scheme for message length $\ell = 8$. In this case the binary tree is T_3 , since $d = \log_2 8 = 3$. We discuss how to eliminate redundancy among the commitments $((c_v)_{v \in \text{copath}(i)})_{i \in I}$ for three different examples of sets $I \subseteq [8]$.

Duplicate commitments The same vertex may appear in multiple copaths, and thus the same commitment is included in each of the corresponding authentication paths. For example, in Figure 29.2 we consider the case of $I = \{5, 6\}$. The corresponding co-paths are:

$$\begin{aligned} \text{copath}(5) &= ((3, 5), (2, 4), (1, 1)) , \\ \text{copath}(6) &= ((3, 6), (2, 4), (1, 1)) . \end{aligned}$$

We see that the vertices $(2, 4)$ and $(1, 1)$ appear in both copaths, which means that the commitments $c_{(2,4)}$ and $c_{(1,1)}$ are included in both authentication paths \mathbf{auth}_5 and \mathbf{auth}_6 . Of course, we can omit duplicate copies of commitments in an opening proof. Overall, the opening proof \mathbf{pf} only needs to include commitments for the following vertices (blue in Figure 29.3):

$$\{(1, 1), (2, 4)\} .$$

²This redundancy causes the delicate technical details in the discussion leading up to the design of MT.Simulate in Construction 18.6.5, which samples a simulated Merkle commitment and opening proof.

Derivable commitments A commitment computed as part of the copath for an entry may be the result of an oracle query to check the copath of another entry. For example, in Figure 29.3 we consider the case of $I = \{3, 6\}$. The corresponding copaths are:

$$\begin{aligned}\text{copath}(3) &= ((3, 4), (2, 1), (1, 2)) , \\ \text{copath}(6) &= ((3, 6), (2, 4), (1, 1)) .\end{aligned}$$

Here there are no duplicate vertices. Nevertheless, the computation to check auth_3 produces the commitment $c_{(1,1)}$ as an answer from the oracle, and therefore there is no need to include it in auth_6 . Similarly, the computation to check auth_3 produces $c_{(1,2)}$ as an answer from the oracle, and therefore there is no need to include it in auth_3 . Overall, the opening proof pf only needs to include commitments for the following vertices (blue in Figure 29.3):

$$\{(3, 4), (3, 5), (2, 1), (2, 4)\} .$$

In the extreme, suppose one considers the authentication paths for all leaves in a particular subtree. In that case, it suffices to provide the authentication path for the root vertex of that subtree, and the rest can be recovered by the checking algorithm by computing the commitments of the relevant inner vertices from the leaf commitments.

Duplicate and derivable There could be both duplicate commitments and derivable commitments across multiple authentication paths. For example, in Figure 29.4 we consider the case of $I = \{1, 3, 5, 6\}$. The corresponding copaths are:

$$\begin{aligned}\text{copath}(1) &= ((3, 2), (2, 2), (1, 2)) , \\ \text{copath}(3) &= ((3, 4), (2, 1), (1, 2)) , \\ \text{copath}(5) &= ((3, 5), (2, 4), (1, 1)) , \\ \text{copath}(6) &= ((3, 6), (2, 4), (1, 1)) .\end{aligned}$$

By removing duplicate commitments and derivable commitments, we see that the opening proof pf only needs to include commitments for the following vertices (blue in Figure 29.4):

$$\{(3, 2), (3, 4), (2, 4)\} .$$

29.2.2 The general case

The path pruning optimization is captured by a function MT.CoSet that maps an index set $I \subseteq [\ell]$ to a vertex set $B^* \subseteq V_\ell$ that is *the minimal set of vertices for authenticating every index in I* . (Recall that V_ℓ is the vertex set of the perfect binary tree T_ℓ in Definition 18.1.1.)

Informally, the computation of B^* from I proceeds layer by layer, starting from the leaves. In the leaf layer, mark the vertices corresponding to the set I , and include in B^* any unmarked vertex whose sibling is marked; then mark the parent of any vertex that is marked. Then repeat this procedure for the next layer. Output the set B^* when this recursive procedure terminates.

$\text{MT.CoSet}(I)$:

1. Initialize $A_d := \{(d, i)\}_{i \in I}$ and empty sets A_{d-1}, \dots, A_1, A_0 .
2. Initialize empty sets B_d^*, \dots, B_1^* .

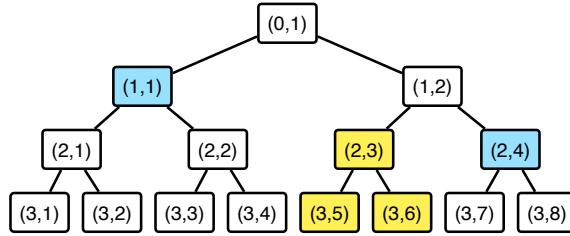


Figure 29.2: The binary tree T_3 with the leaves corresponding to locations $I = \{5, 6\}$ highlighted in yellow, and the minimal set of vertices to authenticate these locations highlighted in blue.

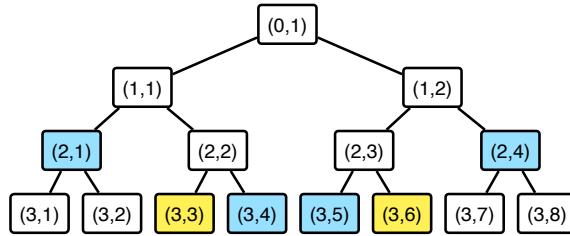


Figure 29.3: The binary tree T_3 with the leaves corresponding to locations $I = \{3, 6\}$ highlighted in yellow, and the minimal set of vertices to authenticate these locations highlighted in blue.

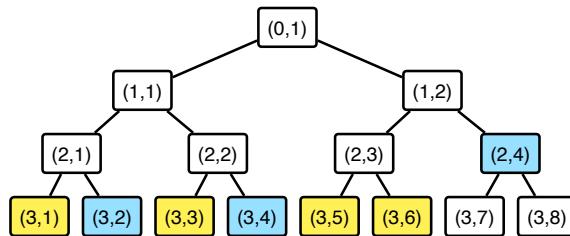


Figure 29.4: The binary tree T_3 with the leaves corresponding to locations $I = \{1, 3, 5, 6\}$ highlighted in yellow, and the minimal set of vertices to authenticate these locations highlighted in blue.

3. For $j = d - 1, \dots, 1, 0$ (in this order) and for $i \in [2^j]$:
 - a) If $(j + 1, 2i - 1) \notin A_{j+1}$ and $(j + 1, 2i) \notin A_{j+1}$ then do nothing.
 - b) If $(j + 1, 2i - 1) \in A_{j+1}$ or $(j + 1, 2i) \in A_{j+1}$ then:
 - if $(j + 1, 2i - 1) \notin A_{j+1}$ then add $(j + 1, 2i - 1)$ to B_{j+1}^* ;
 - if $(j + 1, 2i) \notin A_{j+1}$ then add $(j + 1, 2i)$ to B_{j+1}^* ;
 - add (j, i) to A_j .
4. Output $B^* := \cup_{j \in [d]} B_j^*$.

The minimal set of vertices output by $\text{MT.CoSet}(I)$ has an elegant characterization: the vertices in the copaths for I but not the paths for I . In fact, not by chance, this set of vertices has appeared before in the context of the hiding property of Merkle commitment schemes in Lemma 18.6.3.

Lemma 29.2.1. $\text{MT.CoSet}(I) = \text{copath}(I) \setminus \text{path}(I)$.

Proof. The sets A_d, \dots, A_0 consist of the vertices in $\text{path}(I)$. Indeed, A_d is initialized with the leaf vertices corresponding to the indices in I . Then, for $j = d - 1, \dots, 1, 0$, the algorithm sets A_j to equal the parents of the vertices in A_{j+1} . We conclude that $\cup_{j \in \{0, 1, \dots, d\}} A_j = \text{path}(I)$.

The output of the algorithm is $B^* := \cup_{j \in [d]} B_j^*$. We argue that $B^* = \text{copath}(I) \setminus \text{path}(I)$.

- If a vertex is in $\cup_{j \in \{0, 1, \dots, d\}} A_j = \text{path}(I)$ then the algorithm does not add the vertex to B^* .
- If the algorithm adds a vertex to B^* then the vertex is the sibling of a vertex in $\cup_{j \in \{0, 1, \dots, d\}} A_j = \text{path}(I)$, and hence the vertex is in $\text{copath}(I)$.
- Fix $v \in \text{copath}(I) \setminus \text{path}(I)$. If v is an odd vertex we can write $v = (j + 1, 2i - 1)$ for $j \in \{0, 1, \dots, d - 1\}$ and $i \in [2^j]$, or else if v is an even vertex we can write $v = (j + 1, 2i)$. Suppose without loss of generality that v is odd (the even case is similar). The sibling $v' = (j + 1, 2i)$ of v is in $A_{j+1} \subseteq \text{path}(I)$, so in the (j, i) -th iteration of the algorithm the condition in Item 3b holds. Since $v \notin \text{path}(I)$, we know that $v \notin A_{j+1}$. Hence the algorithm adds v to $B_{j+1}^* \subseteq B^*$.

□

Path pruning requires modifying the algorithms MT.Open and MT.Check of the Merkle commitment scheme in Section 18.1 to the constructions MT.Open_\star and MT.Check_\star below.

- $\text{MT.Open}_\star^f(\text{td}, I) \rightarrow \text{pf}$
 1. Compute the set of vertices $B^* := \text{MT.CoSet}(I)$.
 2. Output the opening proof

$$\text{pf} := \left((\tau_i)_{i \in I}, (c_{(j,i)})_{(j,i) \in \text{MT.CoSet}(I)} \right). \quad (29.2)$$

Namely, the opening proof pf includes the salts for all the authenticated leaves and the commitments for vertices in the set $\text{MT.CoSet}(I)$. The size of the opening proof pf is

$$|I| \cdot s + |\text{MT.CoSet}(I)| \cdot \sigma. \quad (29.3)$$

If privacy is not desired then $s = 0$, in which case the size of pf is $|\text{MT.CoSet}(I)| \cdot \sigma$.

- $\text{MT.Check}_\star^f(\text{rt}, I, \mathbf{a}, \text{pf}) \rightarrow b$
 1. Parse pf as in Equation 29.2.

2. For every $i \in I$, compute the commitment $c_{(d,i)} := f(\mathbf{a}[i], \tau_i)$.
3. For $j = d - 1, \dots, 0$ (in this order) and $i \in [2^j]$:
 - if $(j, i) \in \text{path}(I)$ then set $c_{(j,i)} := f(c_{(j+1,2i-1)}, c_{(j+1,2i)}) \in \{0, 1\}^\sigma$.
4. Check that $\text{rt} = c_{(0,1)}$. (Note that $(0, 1) \in \text{path}(I)$.)

The algorithm MT.Check_* makes $|\text{path}(I)|$ queries to the random oracle, and runs in time proportional to $|\text{copath}(I) \cup \text{path}(I)|$.

Efficiency of path pruning The size of an opening proof pf with path pruning is $|I| \cdot s + |\text{MT.CoSet}(I)| \cdot \sigma$ (see Equation 29.3), compared to $|I| \cdot s + |I| \cdot d \cdot \sigma$ without path pruning. Therefore the effectiveness of path pruning is determined by the size of the set $\text{MT.CoSet}(I)$.

It is clear that $|\text{MT.CoSet}(I)| \leq |I| \cdot d$. But how does $|\text{MT.CoSet}(I)|$ behave?

Intuitively, as the size of I increases so does the size of $\text{MT.CoSet}(I)$. However when $|I| = [\ell]$ (all locations) then $\text{MT.CoSet}(I) = \emptyset$ because there is no need to provide any commitments to recover the Merkle commitment rt starting all message entries and salt strings. Hence, as I increases in size, the size of $\text{MT.CoSet}(I)$ eventually decreases and becomes zero.

In order to build intuition it is useful to discuss two extreme settings.

- *Good case.* The good case is when the locations in I are all the leaves of a subtree of T_ℓ (in which case $|I|$ is a power of 2). In this case, the commitment corresponding to the root vertex of the subtree can be deduced from the message entries and salt strings over I , without the need for any additional information. From thereon it suffices to provide commitments for vertices in the copath of the root vertex, and this copath consists of $d - \log_2 |I|$ vertices. Overall in this case $|\text{MT.CoSet}(I)| = d - \log_2 |I|$.

Figure 29.5 illustrates this phenomenon for message length $\ell = 16$ and index set $I = \{1, 2, 3, 4\}$, in which case the tree depth is $d = 4$. The yellow leaf vertices are the opened locations and the blue vertices are those whose commitments are included in the opening proof. We see that the size of $\text{MT.CoSet}(I)$ is 2, which agrees with the expression $d - \log_2 |I| = 4 - \log_2 4 = 4 - 2 = 2$.

- *Bad case.* The bad case is when the locations in I are as spread out as possible, for example take the set $I = \{j \cdot \frac{\ell}{|I|}\}_{j \in |I|}$ (and assume that $|I|$ divides ℓ , in which case $|I|$ is a power of 2). In this case each location is “alone” in a subtree of $\frac{\ell}{|I|}$ leaves; for each of these $|I|$ subtrees, computing the commitment of the root vertex requires obtaining commitments corresponding to a copath of length $\log_2 \frac{\ell}{|I|} = \log_2 \ell - \log_2 |I| = d - \log_2 |I|$. Thereon, given the commitments of the root vertices of the $|I|$ subtrees, one can compute the commitment of the root vertex of T_ℓ without any additional information. Overall in this case $|\text{MT.CoSet}(I)| = |I| \cdot (d - \log_2 |I|)$.

Figure 29.6 illustrates this phenomenon for message length $\ell = 16$ and index set $I = \{2, 7, 12, 14\}$, in which case the tree depth is $d = 4$. The yellow leaf vertices are the opened locations and the blue vertices are those whose commitments are included in the opening proof. We see that the size of $\text{MT.CoSet}(I)$ is 8, which agrees with the expression $|I| \cdot (d - \log_2 |I|) = 4 \cdot (4 - \log_2 4) = 8$.

The above examples represent extremal situations, as captured by the following lemma. The upper bound in the lemma improves on the prior (trivial) upper bound $|I| \cdot d$.

Lemma 29.2.2. $d - \log_2 |I| \leq |\text{MT.CoSet}(I)| \leq |I| \cdot (d - \log_2 |I|)$.

Proof. We establish the lower bound and the upper bound separately.

Lower bound First we assume that $|I|$ is a power of 2, and then show how to handle the general case. The case where $|I|$ is a power of 2 follows from the claim below, where we set $k := \log_2 |I|$.

Claim 29.2.3. *There exists a sequence of subsets $I_0, \dots, I_k \subseteq [\ell]$ that satisfies the following.*

1. $I_0 = I$ and $|I_0| = \dots = |I_k|$.
2. For every $j \in \{0, 1, \dots, k-1\}$, $|\text{MT.CoSet}(I_{j+1})| \leq |\text{MT.CoSet}(I_j)|$.
3. $|\text{MT.CoSet}(I_k)| = d - \log_2 |I|$.
4. For every $j \in \{0, 1, \dots, k\}$ and every subtree of T_ℓ with 2^j leaf vertices, either no leaf vertex of the subtree is in I_j or all of the leaf vertices of the subtree are in I_j .

The claim suffices because

$$d - \log_2 |I| = |\text{MT.CoSet}(I_k)| \leq |\text{MT.CoSet}(I_0)| = |\text{MT.CoSet}(I)| .$$

Proof of Claim 29.2.3. We define $I_0 := I$ and then recursively define the other subsets.

Given I_j that satisfies the above properties (for index j), we initially set $I_{j+1} := I_j$ and then modify I_{j+1} so that it satisfies the above properties (for index $j+1$).

Suppose that there exists a subtree T of size 2^{j+1} whose leaf vertices partially overlap with I_{j+1} . (If no such subtree T exists, then we are done since I_{j+1} satisfies all the required properties.) Let T_L be its left subtree and T_R be its right subtree. Either no leaf vertex of T_L is in I_{j+1} and all leaf vertices of T_R are in I_{j+1} , or vice versa (otherwise there would be a subtree of size 2^j whose leaf vertices partially overlap with I_j). Moreover, if one such subtree T exists, then another such subtree T' also exists (as $|I|$ is a power of 2). Overall, there are subtrees T and T' , with left and right subtrees for T_L, T_R (for T) and T'_L, T'_R (for T'). Moreover, assume (without loss of generality) that all leaf vertices of T_L and T'_L are in I_{j+1} and no leaf vertices of either T_R or T'_R are in I_{j+1} .

We remove the leaf vertices of T'_L from I_{j+1} and add to I_{j+1} the leaf vertices of T_R . This ensures that no leaf vertex of T' is in I_{j+1} , and all leaf vertices of T are in I_{j+1} . Moreover, this swap does not increase the size of $\text{MT.CoSet}(I_{j+1})$. The root vertex of T_R is in $\text{MT.CoSet}(I_j)$ (in order to authenticate the leaf vertices of T_L). After adding the leaf vertices of T_R to I_{j+1} , this root vertex can be derived from the leaf vertices of T_R , so it is not in $\text{MT.CoSet}(I_{j+1})$. Since the leaves of T'_R were removed from I_{j+1} , commitments of other vertices might be added to $\text{MT.CoSet}(I_{j+1})$. The only vertex that might be added is the root vertex of T'_R . Overall, one vertex has been removed and (at most) one is added, so $|\text{MT.CoSet}(I_{j+1})| \leq |\text{MT.CoSet}(I_j)|$ (the size did not increase).

We continue this process until no such subtrees exist. At the end, I_{j+1} satisfies all properties.

Finally, since $2^k = |I| = |I_k|$, I_k consists of all the leaf vertices of a subtree with 2^k leaf vertices, so, as discussed in the “good case” above the lemma, $|\text{MT.CoSet}(I_k)| = d - \log_2 |I_k| = d - \log_2 |I|$. \square

If $|I|$ is not a power of 2, then we define a subset $I' \subseteq I$ whose cardinality is a power of 2 such that $|\text{MT.CoSet}(I)| \geq |\text{MT.CoSet}(I')|$. We derive I' from I by removing indices as follows. Initially set $I' := I$. If 2 does not divide $|I'|$, there exists a leaf vertex in I' whose sibling is not in I' ; we remove this leaf vertex, which does not increase the size of $\text{MT.CoSet}(I')$. Next, if 4 does not divide $|I'|$, there exists a pair of sibling leaf vertices in I' for which the children of

their parent's sibling both are not in I' ; we remove this pair of siblings (which are leaf vertices), which does not increase the size of $\text{MT.CoSet}(I')$. This continues until $|I'|$ is a power of 2.

At the end of this process, we obtain I' for which we can apply the proved lower bound:

$$|\text{MT.CoSet}(I)| \geq |\text{MT.CoSet}(I')| \geq d - \log_2 |I'| \geq d - \log_2 |I| .$$

Upper bound First we assume that $|I|$ is a power of 2, and then we discuss the general case.

Again setting $k := \log_2 |I|$, we perform a sequence of swaps on I that do not decrease the size of $\text{MT.CoSet}(I)$. Suppose that there exists a subtree T with 2^{d-k} leaf vertices with at least two leaf vertices in I , denoted v_1, v_2 . Then, there exists a subtree T' of the same size with no leaf vertex in I (as otherwise the total number of elements in I would be larger than $2^k = |I|$). We remove from I the index for v_2 and add to I the index for an arbitrary leaf vertex v of T' . Since T' had no leaf vertices in I , $\text{MT.CoSet}(I)$ now contains $d - k$ new vertices for the copath of v . We removed v_2 , and thus vertices from $\text{MT.CoSet}(I)$ might be removed. However, since v_1, v_2 are both in T , at most $d - k$ vertices are removed. Overall, the size of $\text{MT.CoSet}(I)$ did not decrease.

We continue in this manner until all subtrees T with 2^{d-k} leaves have exactly one leaf in I . At this point, the size of $\text{MT.CoSet}(I)$ is upper bounded by $|I| \cdot (d - \log_2 |I|)$ as discussed in the “bad case” above the lemma.

If $|I|$ is not a power of 2, then let K be the smallest power of two with $K \geq |I|$. As explained in the “bad case” above, each location of I is “alone” in a subtree of $\frac{\ell}{K}$ leaves. Thus, each copath has size $d - \log_2 K$, and subtrees that do not have a leaf in I (there are $K - |I|$ such subtrees) cause their root vertex to be in $\text{MT.CoSet}(I)$. Overall, we get the bound

$$\begin{aligned} |\text{MT.CoSet}(I)| &= |I| \cdot (d - \log_2 K) + (K - |I|) \\ &= |I| \cdot \left(d - \log_2 |I| - \log_2 \frac{K}{|I|} \right) + (K - |I|) \\ &= |I| \cdot (d - \log_2 |I|) - |I| \cdot \left(\log_2 \frac{K}{|I|} - \frac{K}{|I|} + 1 \right) \\ &\geq |I| \cdot (d - \log_2 |I|) - |I| \cdot 0 \\ &= |I| \cdot (d - \log_2 |I|) . \end{aligned}$$

Above, the inequality holds since the function $\log_2(x) - x + 1$ is non-negative for $x \in (1, 2)$. \square

The upper bound $|I| \cdot (d - \log_2 |I|)$ is a concave function of $q := |I|$ over the interval $\{0, 1, \dots, \ell\}$: it equals 0 when $q = 0$ or $q = \ell = 2^d$ (the two endpoints of the interval); and in between the function increases up to some threshold q_0 and after that decreases. The threshold is where the derivative $d - \log_2 e - \log_2 q$ of $q \cdot (d - \log_2 q)$ vanishes. Hence the (non-integer) threshold is $q_0 := \frac{2^d}{e} = \frac{\ell}{e}$. Since $e \approx 2.72$, we know that $q_0 \in [2^{d-2}, 2^{d-1}]$. This computation provides an indication for the location of the threshold for $|\text{MT.CoSet}(I)|$, provided I is sufficiently spread out (in which case $|\text{MT.CoSet}(I)|$ is close to its upper bound $|I| \cdot (d - \log_2 |I|)$).

Concrete efficiency In typical scenarios the set I is the result of sampling locations from some high-entropy distribution, and so I is likely to be reasonably spread out. In these cases we expect $|\text{MT.CoSet}(I)|$ to be closer to the upper bound $|I| \cdot (d - \log_2 |I|)$ rather than the lower bound $d - \log_2 |I|$. We illustrate this phenomenon via some empirical data.

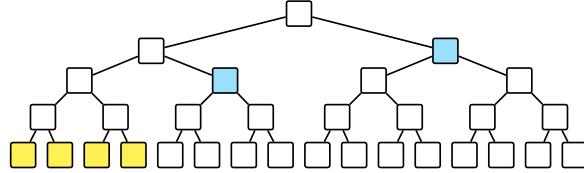


Figure 29.5: Example of path pruning for message length $\ell = 16$ and index set $I = \{1, 2, 3, 4\}$.

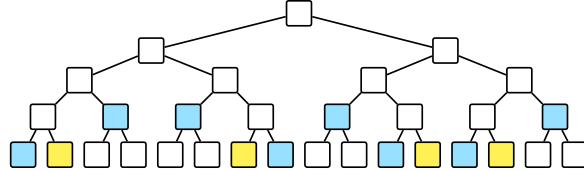


Figure 29.6: Example of path pruning for message length $\ell = 16$ and index set $I = \{2, 7, 12, 14\}$.

We perform the following experiment. Consider the Merkle commitment scheme applied to a message of length $\ell = 2^{20}$. The message alphabet Σ does not affect the size of an opening proof, so we ignore it. We fix representative parameters: the output size of the random oracle is $\sigma = 256$, and the salt size is $s = 128$. We consider a random subset I of $[\ell] = [2^{20}]$ of size q for progressively larger values: we consider $q \in \{2^i\}_{i \in \{0, 1, \dots, 20\}}$. Note that $q = 2^0 = 1$ corresponds to the case where one entry of the message is opened, and $q = 2^{20}$ corresponds to the case where all entries of the message are opened. For each value of q we collect several metrics.

- Salts contribute $g_0 := |I| \cdot s = q \cdot s$ bits to the opening proof. These bits must be included in an opening proof (with or without path pruning).
- Without the path-pruning optimization, an opening proof includes a list of authentication paths that contribute $g_1 := |I| \cdot d \cdot \sigma = q \cdot d \cdot \sigma$ bits.
- With the path-pruning optimization, an opening proof includes (instead of the authentication paths) a list of commitments that contribute $g_2 := |\text{MT.CoSet}(I)| \cdot \sigma$ bits. Note that g_2 is a random variable that depends on the specific (random) choice of set I , because $|\text{MT.CoSet}(I)|$ depends on which elements are in I . Hence we compute the average of g_2 across 20 trials over a random choice of I for the given size $|I| = q$. We denote by \hat{g}_2 the empirical average of g_2 , which is a function of q .

In sum, the size of an opening proof without path pruning is $g_1 + g_0$ while with path pruning is $g_2 + g_0$. We additionally define g'_2 to be the upper bound $|I| \cdot (d - \log_2 |I|) \cdot \sigma$ on g_2 implied by Lemma 29.2.2. All are functions of q .

In Figure 29.7 we provide several graphs to illustrate our data.

- *Top row.* In the left graph we plot $g_1 + g_0$ in black, $\hat{g}_2 + g_0$ in blue, $g'_2 + g_0$ in grey, and g_0 in red; in the right graph we plot g_1 in black, \hat{g}_2 in blue, and g'_2 in grey. The left graph is on a log-log scale, while the right graph is on a symlog scale (so we can include the value zero in the vertical axis).

We see that path pruning contributes bigger savings as q increases, and that the empirical average is rather close to the worst-case upper bound from Lemma 29.2.2. Excluding the salts isolates the savings of path pruning.

- *Bottom row.* In the left graph we plot $\frac{g_1+g_0}{g_2+g_0}$ and in the right graph we plot $\frac{g_1}{g_2}$. Both graphs are on a log-log scale; the right graph excludes the ratio for $q = 2^{20}$ because $g_2 = 0$ for that size (when all entries of the message are revealed).

The ratios show that path pruning contributes improvements that are attractive in practice even when the number of opened locations is a few dozen out of a million locations.

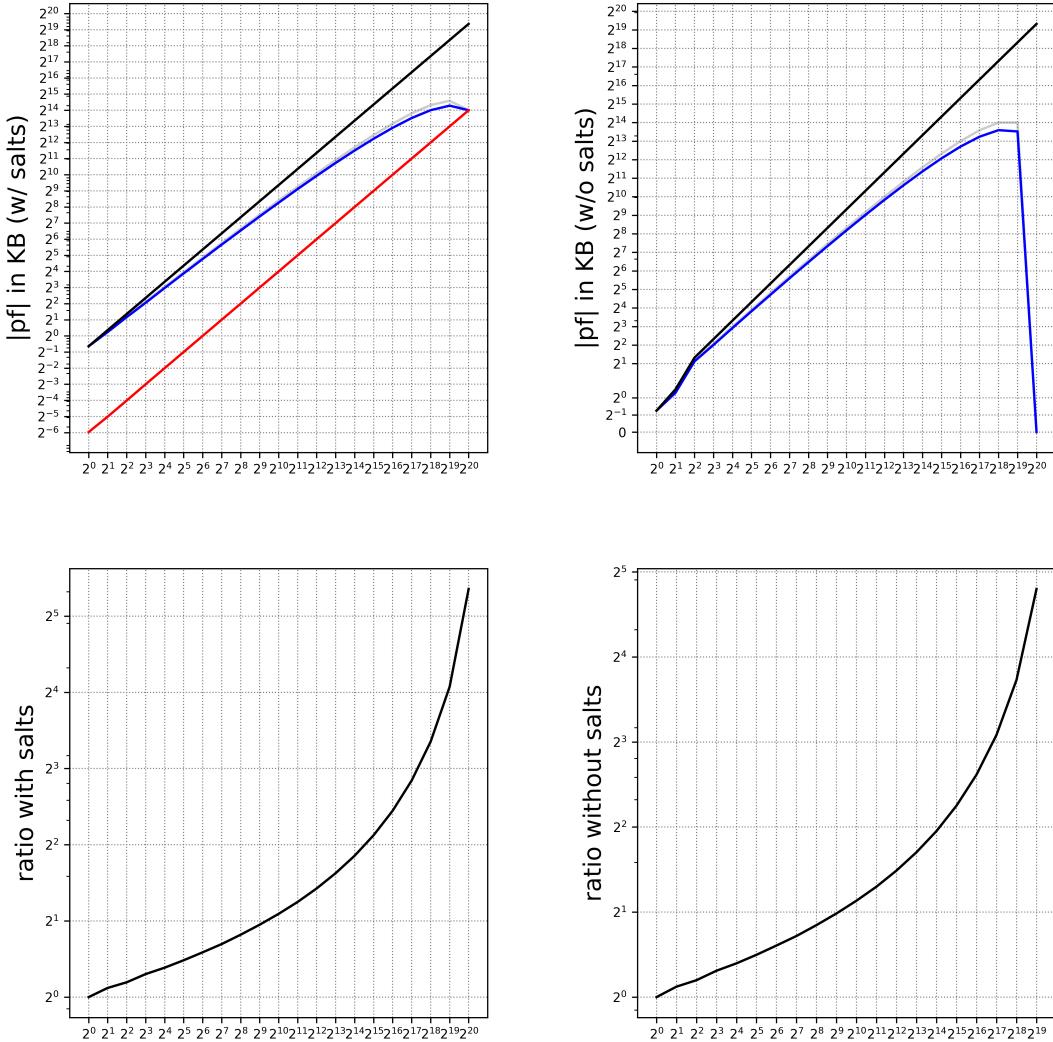


Figure 29.7: The graphs report the size of an opening proof pf as a function of the number of opened queries for the Merkle commitment scheme applied to a message of length $\ell = 2^{20}$. The top row shows the size of pf without path pruning (black line) and with path pruning (blue line); the left graph includes the contribution from salts (red line) and the right graph omits salts. The bottom row shows the relative savings due to path pruning: the black line shows the ratio of the size of pf without path pruning over the size of pf with path pruning; as in the top row, the left graph includes the contribution from salts and the right graph excludes it. See text for more discussion.

29.2.3 Security

Path pruning required changing the algorithms MT.Open and MT.Check into corresponding variants MT.Open_* and MT.Check_* . Here we discuss how these modifications do not impact the security properties of the Merkle commitment scheme that we proved in Chapter 18.

Compressing and expanding opening proofs The path pruning optimization can be viewed as *compressing* an “uncompressed” opening proof and, conversely, the path pruning optimization can be undone by *expanding* a compressed opening proof. We use this structure further below to discuss the security of path pruning.

- *Compression.* The algorithm $\text{MT.Open}_*(\text{td}, I)$ outputs an opening proof pf_* that includes the salt strings for locations in I and the commitments corresponding to vertices in $\text{MT.CoSet}(I)$. Since $\text{MT.CoSet}(I) \subseteq \text{copath}(I)$ (see Lemma 29.2.1), the “compressed” opening proof pf_* can be derived from an “uncompressed” opening proof pf output by $\text{MT.Open}^f(\text{td}, I)$ by omitting the commitments corresponding to vertices that are not in $\text{MT.CoSet}(I)$ (without knowing the opening trapdoor td).

In more detail, define MT.Compress to be the function that omits redundant commitments.

$\text{MT.Compress}(\text{pf}, I)$:

1. Parse pf as $(\text{auth}_i)_{i \in I}$, and each auth_i as $(\tau_i, (c_v)_{v \in \text{copath}(i)})$.
2. Compute $B^* := \text{MT.CoSet}(I)$.
3. Output the opening proof $\text{pf}_* := ((\tau_i)_{i \in I}, (c_{(j,i)})_{(j,i) \in \text{MT.CoSet}(I)})$.

The output of MT.Compress agrees with the output of MT.Open_* :

$$\text{MT.Open}_*(\text{td}, I) = \text{MT.Compress}(\text{MT.Open}^f(\text{td}, I), I).$$

- *Expansion.* The path pruning optimization can be undone: given query access to the random oracle, an opening proof pf_* in the path-pruning format can be *expanded* into an opening proof pf that is a list of authentication paths. This is not surprising because path pruning removes redundancy without any information loss, so the redundancy can be added back.

In more detail, define MT.Expand to be the function that adds missing commitments.

$\text{MT.Expand}^f(\text{pf}_*, I)$:

1. Parse pf_* as $((\tau_i)_{i \in I}, (c_{(j,i)})_{(j,i) \in \text{MT.CoSet}(I)})$.
2. For every $i \in I$, compute the commitment $c_{(d,i)} := f(\mathbf{a}[i], \tau_i)$.
3. For $j = d - 1, \dots, 0$ (in this order) and $i \in [2^j]$:
 - if $(j, i) \in \text{path}(I)$ then set $c_{(j,i)} := f(c_{(j+1,2i-1)}, c_{(j+1,2i)}) \in \{0, 1\}^\sigma$.
4. Output $\text{pf} := (\text{auth}_i)_{i \in I}$, where $\text{auth}_i := (\tau_i, (c_v)_{v \in \text{copath}(i)})$.

Note that the algorithm has all the commitments needed for the authentication paths because the algorithm computes the commitments corresponding to vertices in $\text{path}(I)$ and $\text{copath}(I) \setminus \text{MT.CoSet}(I) \subseteq \text{path}(I)$ (see Lemma 29.2.1).

Expanding a compressed opening proof recovers the starting opening proof:

$$\text{MT.Expand}^f(\text{MT.Compress}(\text{pf}, I), I) = \text{pf}.$$

In particular, the expanded proof is equivalent to the compressed proof in the following sense:

$$\begin{aligned} \text{MT.Check}_\star^f(\text{rt}, I, \mathbf{a}, \text{pf}_\star) &= \text{MT.Check}^f(\text{rt}, I, \mathbf{a}, \text{MT.Expand}^f(\text{pf}_\star, I)), \text{ and} \\ \text{MT.Check}^f(\text{rt}, I, \mathbf{a}, \text{pf}) &= \text{MT.Check}_\star^f(\text{rt}, I, \mathbf{a}, \text{MT.Compress}(\text{pf}, I)). \end{aligned}$$

Moreover, the query-answer trace S_\star of MT.Check_\star and the query-answer trace S of MT.Check are equal (up to the order of query-answer pairs). Specifically, in each case the query-answer trace contains the following $|\text{path}(I)|$ entries:

- for every $i \in I$, the query $(\mathbf{a}[i], \tau_i)$ yielding the answer $c_{(d,i)} := f(\mathbf{m}[i], \tau_i)$;
- for every $(j, i) \in \cup_{i \in I} \text{path}(i) \setminus \{(d, i)\}$, the query $(c_{(j+1,2i-1)}, c_{(j+1,2i)})$ yielding the answer $c_{(j,i)} := f(c_{(j+1,2i-1)}, c_{(j+1,2i)})$.

Properties Since the path-pruning optimization can be viewed as compressing opening proofs and these can be expanded to recover the starting opening proofs, there is a correspondence between adversaries against the Merkle commitment scheme without or with path pruning. The correspondence has a (small) cost, because MT.Expand queries the random oracle. This means that the query bound of adversaries may increase by a small additive factor that equals the size of the query-trace of MT.Expand . However this cost can be avoided by inspecting the proof of each the security properties of interest to us.

- *Collision lemma.* Lemma 18.3.2 and its special case Lemma 18.3.1 hold as stated for the Merkle commitment scheme with path pruning, that is, with MT.Check_\star instead of MT.Check . Indeed, for every $b \in \{0, 1\}$, we have that $\text{MT.Check}_\star^f(\text{rt}, I_b, \mathbf{a}_b, \text{pf}_b)$ equals $\text{MT.Check}^f(\text{rt}, I_b, \mathbf{a}_b, \text{MT.Expand}^f(\text{pf}_b, I))$, and the query-answer traces of both executions are equal when viewed as sets.
- *Binding.* Lemma 18.4.1 holds as stated for the Merkle commitment scheme with path pruning, that is, with MT.Check_\star instead of MT.Check . The analysis carries over with minimal changes. Each appearance of MT.Check is replaced with MT.Check_\star . Next, the analysis of the event $E \wedge \overline{E_{\text{col}}}$ refers to authentication paths for single locations, which do not exist in an opening proof in path-pruning format; instead, one refers to authentication paths in the expanded opening proof. In more detail, for every $b \in \{0, 1\}$, auth_b is the authentication path for location i in $\text{MT.Expand}^f(\text{pf}_b, I)$; then $\text{MT.Check}_\star^f(\text{rt}, I_b, \mathbf{a}_b, \text{pf}_b) = 1$ implies that $\text{MT.Check}^f(\text{rt}, i, \mathbf{a}_b[i], \text{auth}_b) = 1$. Finally, one relies on the collision lemma for MT.Check (even though the statement is about MT.Check_\star).

- *Extractability.* Lemma 18.5.1 (single extraction) and Lemma 18.5.6 (multi-extraction) hold as stated for the Merkle commitment scheme with path pruning, that is, with MT.Check_\star and MT.Open_\star instead of MT.Check and MT.Open . The extractor MT.Extract in Construction 18.5.3 remains the same. The analyses of the lemmas carry over with minimal changes. We outline the changes for the case of Lemma 18.5.1; the case of Lemma 18.5.6 is similar.

Each appearance of MT.Check is replaced with MT.Check_\star , and similarly each appearance of MT.Open is replaced with MT.Open_\star . Next, the analysis of the event $E_{\text{sub}} \mid \overline{E_{\text{col}}}$ refers to authentication paths for single locations, which do not exist in an opening proof in path-pruning format; instead, one refers to authentication paths in the expanded opening proof.

In more detail, for $i \in I$, auth_i is the authentication path for location i in $\text{MT}.\text{Expand}^f(\text{pf}, I)$; then $\text{MT}.\text{Check}_{\star}^f(\text{rt}, I, \mathbf{a}, \text{pf}) = 1$ implies that $\text{MT}.\text{Check}^f(\text{rt}, i, \mathbf{a}[i], \text{auth}_i) = 1$. Similarly, in the proof of Claim 18.5.4, auth'_i is the authentication path for location i in $\text{MT}.\text{Expand}^f(\text{pf}', I)$; then $\text{MT}.\text{Check}_{\star}^f(\text{rt}, I, \mathbf{m}[I], \text{pf}') = 1$ implies that $\text{MT}.\text{Check}^f(\text{rt}, i, \mathbf{m}[i], \text{auth}'_i) = 1$. Finally, one relies on the collision lemma for $\text{MT}.\text{Check}$ (even though the statement is about $\text{MT}.\text{Check}_{\star}$).

- *Hiding.* We discuss separately the two hiding lemmas.

Lemma 18.6.1 only involves the commitment algorithm $\text{MT}.\text{Commit}$, which is unaffected by path pruning. Hence this lemma continues to hold without change.

Lemma 18.6.3 additionally involves the opening algorithm $\text{MT}.\text{Open}$, which is replaced by $\text{MT}.\text{Open}_{\star}$. After modifying the simulator $\text{MT}.\text{Simulate}$ in Construction 18.6.5 to reflect the format of opening proofs after path pruning, the same bound holds for $\text{MT}.\text{Open}_{\star}$ because $\text{MT}.\text{Open}_{\star}$ omits from the opening proof pf duplicate commitments or commitments that can be computed from other commitments (and the random oracle). In fact, path pruning makes explicit why the upper bound in Lemma 18.6.3 holds: the only errors that contribute statistical distance are those corresponding to commitments of vertices in $V^*(T_{\ell}, I) = \text{MT}.\text{CoSet}(I)$.

30 Special soundness

Special soundness is a knowledge soundness property satisfied by many sigma protocols (SPs) and public-coin interactive proofs (IPs) of interest (see Chapter 27). Informally, special soundness states that there exists an efficient knowledge extractor that finds a valid witness, for a given instance, when provided with a suitable “tree” of accepting interaction transcripts of the proof system. Such trees can then be produced from any prover that is sufficiently convincing, by rerunning the prover multiple times on suitably correlated inputs.

A crucial implication, from the perspective of this book, is that *special soundness implies (rewinding) state-restoration knowledge soundness*. In turn, this means that: (i) SPs that have special soundness yield non-interactive arguments that have (adaptive and rewinding) knowledge soundness via the Fiat–Shamir transformation for SPs (discussed in Part II); and (ii) public-coin IPs that have special soundness yield non-interactive arguments that have (adaptive and rewinding) knowledge soundness via the Fiat–Shamir transformation for public-coin IPs (discussed in Part III).

In this chapter we discuss definitions of special soundness and establish the implication to rewinding state-restoration knowledge soundness.¹

Organization In Section 30.1 we define special soundness for SPs and public-coin IPs. First, we study special soundness for SPs: in Section 30.2 we show that it implies (standard) rewinding knowledge soundness; then in Section 30.3 we show that it implies state-restoration rewinding knowledge soundness. Second, we study special soundness for public-coin IPs: in Section 30.4 we show that it implies (standard) rewinding knowledge soundness; then in Section 30.5 we show that it implies state-restoration rewinding knowledge soundness.

Remark 30.0.1 (extraction tradeoffs). All results in this chapter achieve a notion of rewinding knowledge soundness that is slightly stronger than the general definitions used earlier in this book: the knowledge soundness error does not depend on the failure probability of the adversary. Informally, this is because the rewinding knowledge extractor runs in expected time until it succeeds and, as long as the adversary convinces the verifier with high enough probability, the rewinding knowledge extractor will succeed. However this is merely a point in a tradeoff space. All extractors in this chapter could be modified to run in less time at the cost of a larger knowledge soundness error. Intuitively, the extractor will give up after some number of rewinds, and the larger the adversary’s failure probability then the larger the knowledge soundness error. Analyses for such knowledge extractors (which we do not include in this book) would result in a knowledge soundness error that depends on the adversary’s failure probability.

30.1 Extraction from forks and trees

We define the notion of special soundness for SPs and then for public-coin IPs.

¹All definitions and results about special soundness for public-coin IPs naturally extend to the case of special soundness for public-coin IOPs (which can sometimes be of interest).

The case of SPs An SP is a 3-message protocol between a prover and a verifier (see Chapter 8). An SP has special soundness if a witness can be efficiently deduced from a *fork* of accepting interaction transcripts for the given instance. A fork consists of multiple accepting interaction transcripts that share the same SP prover commitment and have pairwise distinct SP verifier challenges. (The SP prover responses need not be distinct.) The number of transcripts of the fork is determined by an *arity parameter* $a \in \mathbb{N}$. See Figure 30.1 for a diagram.

Definition 30.1.1. Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP and $a \in \mathbb{N}$ an arity parameter. A a -fork for an instance \mathbf{x} is a tuple

$$F = \left(\alpha_1, ((\rho_j, \alpha_{2,j}))_{j \in [a]} \right)$$

where:

- α_1 is an SP prover commitment;
- $(\rho_j)_{j \in [a]}$ are pairwise distinct SP verifier challenges;
- $(\alpha_{2,j})_{j \in [a]}$ are SP prover responses (not necessarily distinct);
- $\forall j \in [a], \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho_j, \alpha_{2,j}) = 1$.

Definition 30.1.2. $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ has **special soundness with arity a** with extraction time $\text{et}_{\text{SP}}^{\text{ss}}$ if there exists a deterministic algorithm $\mathbf{E}_{\text{SP}}^{\text{ss}}$ such that, for every instance \mathbf{x} and a -fork F for \mathbf{x} , $\mathbf{E}_{\text{SP}}^{\text{ss}}(\mathbf{x}, F)$ outputs \mathbf{w} such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ in time $\text{et}_{\text{SP}}^{\text{ss}}(\mathbf{x})$.

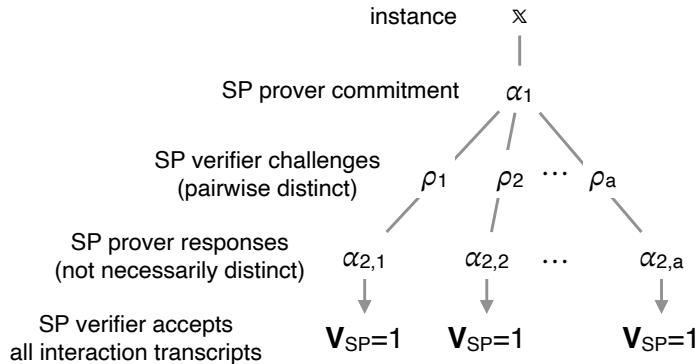


Figure 30.1: Diagram of an a -fork for an SP.

The case of public-coin IPs An IP is a multi-round protocol between a prover and a verifier (see Chapter 13). A public-coin IP has special soundness if a witness can be efficiently deduced from a *tree* of accepting interaction transcripts for the given instance. This latter consists of a tree where vertices are labeled with IP prover messages, edges are labeled IP verifier messages, and every path from the root to a leaf is an accepting transcript; edges sharing the same vertex are labeled with pairwise distinct IP verifier messages. A list of parameters $a_1, \dots, a_k \in \mathbb{N}$ determines the number of children that each vertex in the tree has, depending on the layer. See Figure 30.2 for a diagram.

Definition 30.1.3. Let $a_1, \dots, a_k \in \mathbb{N}$. A (a_1, \dots, a_k) -tree is a tree T where: (1) there are $k + 1$ layers, with layer 1 being the root layer and layer $k + 1$ being the leaf layer; (2) for every $i \in [k]$, every vertex in layer i has a_i children. In particular, the tree has $\prod_{i \in [k]} a_i$ leaves.

Definition 30.1.4. Let $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ be a public-coin IP with round complexity k . Let $a_1, \dots, a_k \in \mathbb{N}$ be arity parameters. A (a_1, \dots, a_k) -tree of transcripts for an instance \mathbf{x} is a (a_1, \dots, a_k) -tree T that comes with vertex labels and edge labels that yield accepting transcripts: (1) for every $i \in [k]$, every vertex in layer i is labeled with an IP prover message suitable for round i (and the leaf vertices are unlabeled); (2) for every $i \in [k]$ and every vertex in layer i , the a_i outgoing edges are labeled with distinct choices of IP verifier challenge suitable for round i ; and (3) every path from the root to a leaf corresponds to an accepting view for the IP verifier (on instance \mathbf{x}).

Definition 30.1.5. Let $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ be a public-coin IP with round complexity k . The IP IP has **special soundness with arity** (a_1, \dots, a_k) with **extraction time** $\text{et}_{\text{IP}}^{\text{ss}}$ if there exists a deterministic algorithm $\mathbf{E}_{\text{IP}}^{\text{ss}}$ such that, for every instance \mathbf{x} and (a_1, \dots, a_k) -tree T of transcripts for \mathbf{x} , $\mathbf{E}_{\text{IP}}^{\text{ss}}(\mathbf{x}, T)$ outputs \mathbf{w} such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ in time $\text{et}_{\text{IP}}^{\text{ss}}(\mathbf{x})$.

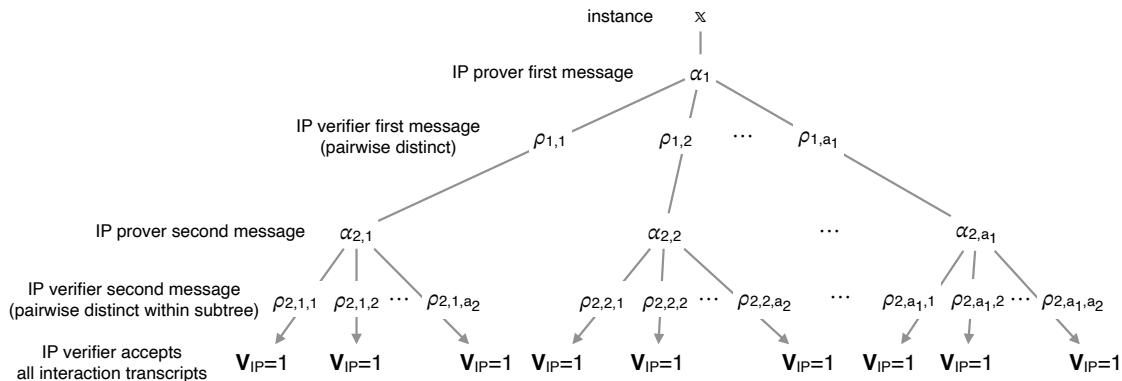


Figure 30.2: Diagram of an (a_1, a_2) -tree of transcripts for a public-coin IP.

We also define the notion of a *subtree* of transcripts, which considers multiple accepting completions of a single partial transcript. The definition below generalizes Definition 30.1.4 and, while not needed to define special soundness, we rely on it in various analyses in this chapter.

Definition 30.1.6. For $j \in \{0, 1, \dots, k\}$, a (a_{j+1}, \dots, a_k) -subtree of transcripts for an instance \mathbf{x} and partial IP transcript $((\alpha_i)_{i=1}^j, (\rho_i)_{i=1}^j)$ is a (a_{j+1}, \dots, a_k) -tree T that comes with vertex labels and edge labels that yield accepting transcripts: (1) for every $i \in \{j+1, \dots, k\}$, every vertex in layer $i - j$ is labeled with an IP prover message suitable for round i (and the leaf vertices are unlabeled); (2) for every $i \in \{j+1, \dots, k\}$ and every vertex in layer $i - j$, the a_i outgoing edges are labeled with distinct choices of IP verifier challenge suitable for round i ; and (3) every path from the root to a leaf completes the partial transcript $((\alpha_i)_{i=1}^j, (\rho_i)_{i=1}^j)$ into an accepting view for the IP verifier (on instance \mathbf{x}).

Negative hypergeometric distribution In this chapter, when analyzing certain probabilistic algorithms, the properties of the following distribution will be useful.

Definition 30.1.7. The negative hypergeometric distribution describes the following probabilistic experiment. Elements are drawn one after another, without replacement, from a set of N elements out of which B are colored blue and the rest are colored red.

Let X_{NHG} be the number of samples until b blue elements are drawn, and let Y_{NHG} be the number of red balls sampled. Then,

$$\Pr[X_{\text{NHG}} = x] = \binom{x-1}{b-1} \cdot \frac{\binom{N-x}{B-b}}{\binom{N}{B}}, \quad \mathbb{E}[X_{\text{NHG}}] = \frac{b \cdot (N+1)}{B+1}, \quad \text{and} \quad \mathbb{E}[Y_{\text{NHG}}] \leq \frac{b \cdot (N-B)}{B+1}.$$

30.2 Knowledge soundness for SPs

We prove that if an SP has special soundness then the SP has rewinding knowledge soundness. The rewinding knowledge soundness error and extraction time increase as the arity of the special soundness increases; neither depends on the failure probability of the SP prover.

Theorem 30.2.1. *Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP with verifier randomness complexity r . If SP has special soundness with arity a with extraction time $\mathbf{et}_{\text{SP}}^{\text{ss}}$ (see Definition 30.1.2) then SP has rewinding knowledge soundness error κ_{SP} with extraction time \mathbf{et}_{SP} (see Definition 8.1.4) such that*

$$\begin{aligned} \kappa_{\text{SP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{SP}}}) &\leq \frac{a-1}{2^r}, \\ \mathbf{et}_{\text{SP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{SP}}}, \tau_{\tilde{\mathbf{P}}_{\text{SP}}}) &\leq (a-1) \cdot \left(\tau_{\tilde{\mathbf{P}}_{\text{SP}}} + \text{poly}(\text{len}(\mathbf{x})) \right) + \mathbf{et}_{\text{SP}}^{\text{ss}}(\mathbf{x}). \end{aligned}$$

Fix an instance \mathbf{x} and SP prover $\tilde{\mathbf{P}}_{\text{SP}}$. The main idea to prove the theorem is to rewind $\tilde{\mathbf{P}}_{\text{SP}}$ several times with correlated inputs in order to obtain an a -fork F for \mathbf{x} . This suffices because the knowledge extractor $\mathbf{E}_{\text{SP}}^{\text{ss}}$ for special soundness can find a witness w from \mathbf{x} and F .

The lemma below provides a probabilistic procedure **ForkFinder** that outputs an a -fork F . The procedure **ForkFinder** receives as input the instance \mathbf{x} , an interaction transcript $(\alpha_1, \rho, \alpha_2)$ between the SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ and SP verifier \mathbf{V}_{SP} , and black-box access to $\tilde{\mathbf{P}}_{\text{SP}}$. We describe how to upper bound the probability that $\tilde{\mathbf{P}}_{\text{SP}}$ convinces \mathbf{V}_{SP} and the output of **ForkFinder** is not an a -fork F for \mathbf{x} . Informally, **ForkFinder** invokes $\tilde{\mathbf{P}}_{\text{SP}}$ with different SP verifier challenges (but the same SP prover commitment) until it obtains $a-1$ additional accepting interaction transcripts. (The first accepting interaction transcript comes “for free” when the SP verifier \mathbf{V}_{SP} accepts in the real interaction.)

Lemma 30.2.2. *The algorithm **ForkFinder** in Construction 30.2.3 satisfies the following property. For every SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ and instance \mathbf{x} ,*

$$\Pr \left[\begin{array}{l} F \text{ is not an } a\text{-fork for } \mathbf{x} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}} \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \\ F \leftarrow \text{ForkFinder}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}) \end{array} \right] \leq \frac{a-1}{2^r}.$$

Moreover, **ForkFinder** runs in expected time $(a-1) \cdot (\tau_{\tilde{\mathbf{P}}_{\text{SP}}} + \text{poly}(\text{len}(\mathbf{x})))$.

Construction 30.2.3. The algorithm **ForkFinder**, given as input an instance \mathbf{x} , an SP interaction transcript $(\alpha_1, \rho, \alpha_2)$, and black-box access to an SP prover $\tilde{\mathbf{P}}_{\text{SP}}$, works as follows.

ForkFinder($\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}$):

1. If $\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 0$ abort.

2. Compute $\tilde{\mathbf{P}}_{\text{SP}}$'s first message and intermediate state: $(\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}$.
3. Set $S := \{0, 1\}^r \setminus \{\rho\}$ and $B := \{(\rho, \alpha_2)\}$.
4. While S is non-empty and $|B| < a$:
 - a) Sample an SP verifier challenge $\rho' \in S$, and remove ρ' from S .
 - b) Compute $\tilde{\mathbf{P}}_{\text{SP}}$'s second message: $\alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho')$.
 - c) Compute the decision bit of the SP verifier: $b \leftarrow \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho', \alpha_2)$.
 - d) If $b = 1$ then add (ρ', α'_2) to B .
5. If $|B| < a$ then abort; else output $F := (\alpha_1, B)$.

Proof of Lemma 30.2.2. We say that an SP verifier challenge $\rho \in \{0, 1\}^r$ is *good* for $(\mathbf{x}, \tilde{\mathbf{P}}_{\text{SP}})$ if $\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1$, where $(\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}}$ and $\alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho)$. Denote by $Y \in \{0, 1, \dots, 2^r\}$ the number of SP verifier challenges in $\{0, 1\}^r$ that are good for $(\mathbf{x}, \tilde{\mathbf{P}}_{\text{SP}})$. Denote by $Z \in \{0, 1\}$ the indicator for the event that ρ (sampled in the experiment of the lemma statement) is good for $(\mathbf{x}, \tilde{\mathbf{P}}_{\text{SP}})$. By definition, for every $\ell \in \{0, 1, \dots, 2^r\}$,

$$\Pr[Z = 1 \mid Y = \ell] = \frac{\ell}{2^r}. \quad (30.1)$$

Error probability We upper bound the error probability of **ForkFinder**. The event in the probability in the lemma statement corresponds to $Z = 1 \wedge Y \leq a - 1$. Indeed, provided that ρ is good for $(\mathbf{x}, \tilde{\mathbf{P}}_{\text{SP}})$, **ForkFinder** outputs an a -fork for \mathbf{x} if and only if there are at least a SP verifier challenges that are good for $(\mathbf{x}, \tilde{\mathbf{P}}_{\text{SP}})$. We upper bound the error probability as follows:

$$\begin{aligned} & \Pr \left[\begin{array}{l} F \text{ is not an } a\text{-fork for } \mathbf{x} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} (\alpha_1, \text{aux}) \leftarrow \tilde{\mathbf{P}}_{\text{SP}} \\ \rho \leftarrow \{0, 1\}^r \\ \alpha_2 \leftarrow \tilde{\mathbf{P}}_{\text{SP}}(\text{aux}, \rho) \\ F \leftarrow \text{ForkFinder}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}}) \end{array} \right] \\ &= \Pr[Z = 1 \wedge Y \leq a - 1] \\ &\leq \Pr[Z = 1 \mid Y \leq a - 1] \\ &= \sum_{\ell=0}^{a-1} \Pr[Z = 1 \mid Y = \ell] \cdot \Pr[Y = \ell] \\ &= \sum_{\ell=0}^{a-1} \frac{\ell}{2^r} \cdot \Pr[Y = \ell] \quad (\text{by Equation 30.1}) \\ &\leq \frac{a-1}{2^r} \cdot \sum_{\ell=0}^{a-1} \Pr[Y = \ell] \\ &\leq \frac{a-1}{2^r}. \end{aligned}$$

The while loop Let X be the number of times **ForkFinder** enters the while loop. We argue that $\mathbb{E}[X] \leq a - 1$. The expectation is over all randomness in the experiment: the SP verifier challenge ρ and the internal randomness ζ of **ForkFinder**.

If ρ is not good for $(\mathbf{x}, \tilde{\mathbf{P}}_{\text{SP}})$ then **ForkFinder** aborts in Item 1, so $\mathbb{E}[X \mid Z = 0] = 0$. If instead ρ is good for $(\mathbf{x}, \tilde{\mathbf{P}}_{\text{SP}})$ then **ForkFinder** reaches the while loop. We argue that, for every $\ell \in \{1, \dots, 2^r\}$,

$$\mathbb{E}[X \mid Z = 1 \wedge Y = \ell] \leq \frac{(a-1) \cdot 2^r}{\ell}. \quad (30.2)$$

We consider two cases.

- If $\ell < a$ then **ForkFinder** enters the while loop $2^r - 1$ times and then aborts (there are not enough good SP verifier challenges). Hence, $\mathbb{E}[X \mid Y = \ell] = 2^r - 1 = \frac{\ell \cdot (2^r - 1)}{\ell} \leq \frac{(a-1) \cdot (2^r - 1)}{\ell} < \frac{(a-1) \cdot 2^r}{\ell}$.
- If $\ell \geq a$ then **ForkFinder** enters the while loop until it finds $a - 1$ SP verifier challenges that are good for $(\mathbf{x}, \tilde{\mathbf{P}}_{\text{SP}})$. Specifically, **ForkFinder** samples without replacement from a set of cardinality $2^r - 1$ until it obtains $a - 1$ “successes”. Therefore, X follows the random variable X_{NHG} defined for the negative hypergeometric distribution (see Definition 30.1.7) with a set size $2^r - 1$, a total of $\ell - 1$ blue elements, and a target of $a - 1$ blue elements. Hence the expected number of times **ForkFinder** enters the while loop is $\mathbb{E}[X \mid Y = \ell] = \frac{(a-1) \cdot 2^r}{\ell}$.

Therefore, by the total expectation theorem,

$$\begin{aligned}
\mathbb{E}[X] &= \mathbb{E}[X \mid Z = 0] \cdot \Pr[Z = 0] + \mathbb{E}[X \mid Z = 1] \cdot \Pr[Z = 1] \\
&= 0 + \mathbb{E}[X \mid Z = 1] \cdot \Pr[Z = 1] \\
&= \sum_{\ell=1}^{2^r} \mathbb{E}[X \mid Z = 1 \wedge Y = \ell] \cdot \Pr[Z = 1 \wedge Y = \ell] \quad (Z = 1 \text{ implies } Y > 0) \\
&\leq \sum_{\ell=1}^{2^r} \frac{(a-1) \cdot 2^r}{\ell} \cdot \Pr[Z = 1 \wedge Y = \ell] \quad (\text{by Equation 30.2}) \\
&= \sum_{\ell=1}^{2^r} \frac{(a-1) \cdot 2^r}{\ell} \cdot \Pr[Z = 1 \mid Y = \ell] \cdot \Pr[Y = \ell] \\
&= \sum_{\ell=1}^{2^r} \frac{(a-1) \cdot 2^r}{\ell} \cdot \frac{\ell}{2^r} \cdot \Pr[Y = \ell] \quad (\text{by Equation 30.1}) \\
&= (a-1) \cdot \sum_{\ell=1}^{2^r} \Pr[Y = \ell] = a-1.
\end{aligned}$$

Running time We analyze the expected running time of **ForkFinder**. First we analyze how **ForkFinder** invokes $\tilde{\mathbf{P}}_{\text{SP}}$. The algorithm **ForkFinder** computes a first message of $\tilde{\mathbf{P}}_{\text{SP}}$ in Item 1; this is the only time **ForkFinder** computes a first message of $\tilde{\mathbf{P}}_{\text{SP}}$. Moreover, **ForkFinder** computes a second message of $\tilde{\mathbf{P}}_{\text{SP}}$ every time **ForkFinder** executes the while loop, in Item 4b. In expectation this happens $\mathbb{E}[X] \leq a - 1$ times. Overall, since $\tau_{\tilde{\mathbf{P}}_{\text{SP}}}$ denotes the total running time of $\tilde{\mathbf{P}}_{\text{SP}}$ (i.e., the running time to compute the first and the second message), the total time spent by **ForkFinder** invoking $\tilde{\mathbf{P}}_{\text{SP}}$ is upper bounded by $(a-1) \cdot \tau_{\tilde{\mathbf{P}}_{\text{SP}}}$. All other operations in **ForkFinder** take time $\text{poly}(\text{len}(\mathbf{x}))$, which in expectation across all while loop iterations add up to time $(a-1) \cdot \text{poly}(\text{len}(\mathbf{x}))$. \square

Construction 30.2.4. Let $\mathbf{E}_{\text{SP}}^{\text{ss}}$ be the special soundness extractor for SP, and let **ForkFinder** be the algorithm in Construction 30.2.3. The rewinding knowledge extractor \mathbf{E}_{SP} receives as input an instance \mathbf{x} , an interaction transcript $(\alpha_1, \rho, \alpha_2)$, and black-box access to an SP prover $\tilde{\mathbf{P}}_{\text{SP}}$, and works as follows.

$\mathbf{E}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}})$:

1. Run $F \leftarrow \text{ForkFinder}(\mathbf{x}, \alpha_1, \rho, \alpha_2, \tilde{\mathbf{P}}_{\text{SP}})$; if ForkFinder aborts then abort.
2. Run $\mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}^{\text{ss}}(\mathbf{x}, F)$.
3. Output \mathbf{w} .

Proof of Theorem 30.2.1. The knowledge extractor \mathbf{E}_{SP} succeeds if the algorithm ForkFinder succeeds, so the knowledge soundness error of \mathbf{E}_{SP} is at most the failure probability of ForkFinder . This latter, by Lemma 30.2.2, is at most $\frac{a-1}{2^r}$. Hence the knowledge soundness error of \mathbf{E}_{SP} is as claimed in Theorem 30.2.1.

Moreover, the expected running time of \mathbf{E}_{SP} is the expected running time of ForkFinder plus the running time of $\mathbf{E}_{\text{SP}}^{\text{ss}}$. By Lemma 30.2.2, the expected running time of ForkFinder is at most $(a-1) \cdot (\tau_{\tilde{\mathbf{P}}_{\text{SP}}} + \text{poly}(\text{len}(\mathbf{x})))$. By definition, the running time of $\mathbf{E}_{\text{SP}}^{\text{ss}}$ is $\mathbf{et}_{\text{SP}}^{\text{ss}}(\mathbf{x})$. We conclude that the expected running time of \mathbf{E}_{SP} is as claimed in Theorem 30.2.1. \square

30.3 State-restoration knowledge soundness for SPs

We prove that if an SP has special soundness then the SP has rewinding *state-restoration* knowledge soundness. The rewinding state-restoration knowledge soundness error and extraction time increase as the arity of the special soundness increases; neither depends on the failure probability of the state-restoration SP prover.

Theorem 30.3.1. *Let $\text{SP} = (\mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be an SP with verifier randomness complexity r . If SP has special soundness with arity a with extraction time $\mathbf{et}_{\text{SP}}^{\text{ss}}$ (see Definition 30.1.2) then SP has rewinding state-restoration knowledge soundness error $\kappa_{\text{SP}}^{\text{sr}}$ with extraction time $\mathbf{et}_{\text{SP}}^{\text{sr}}$ (see Definition 12.1.5) such that*

$$\begin{aligned}\kappa_{\text{SP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}) &\leq (t+1) \cdot \frac{a-1}{2^r}, \\ \mathbf{et}_{\text{SP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}, \tau_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}) &\leq (t+1) \cdot (a-1) \cdot \left(\tau_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}} + \text{poly}(\text{len}(\mathbf{x})) \right) + \mathbf{et}_{\text{SP}}^{\text{ss}}(\mathbf{x}).\end{aligned}$$

The proof of this theorem is similar to the proof of Theorem 30.2.1: rewind the given SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ several times with correlated inputs in order to obtain an a -fork F for the instance \mathbf{x} ; this suffices because the knowledge extractor $\mathbf{E}_{\text{SP}}^{\text{ss}}$ for special soundness can find a witness \mathbf{w} from \mathbf{x} and F . However, there are notable differences that make the proof more technical. First, in rewinding state-restoration knowledge soundness (Definition 12.1.5), the SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ chooses the instance \mathbf{x} in the SP state-restoration game; in contrast, in rewinding knowledge soundness (Definition 8.1.4) the instance \mathbf{x} is fixed. Moreover, the SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ can make multiple attempts at convincing the SP verifier, not only with different instances but also with different SP commitments; this complicates the rewinding strategy.

The lemma below provides a probabilistic procedure SRForkFinder that outputs an a -fork F , which is analogous to Lemma 30.2.2. The procedure SRForkFinder receives as input the output $(\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2)$ of the SP state-restoration game played by $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$, the list of move-response pairs tr^{sr} of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$, and black-box access to $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$. The main challenge is that each time SRForkFinder reruns $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ we get a new output whose instance and first SP prover message may not agree with those received as input, which would not contribute progress towards the desired a -fork

for \mathbf{x} .² Informally, **SRForkFinder** identifies the first move (if any) corresponding to $(\mathbf{x}, \alpha_1, \sigma)$, and then reruns $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ with freshly sampled answers (consistently answering other moves as needed) until enough accepting SP interaction transcripts with the same instance and SP prover message are found. Intuitively, this takes a multiplicative factor $(t + 1)$ more reruns compared to Lemma 30.2.2, as $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ makes at most t moves and can also output something corresponding to no move. Analyzing the success probability and expected running time of **SRForkFinder** is more technical, and the simpler analysis for Lemma 30.2.2 is as a useful warmup.

Lemma 30.3.2. *The algorithm **SRForkFinder** in Construction 30.3.3 satisfies the following property. For every SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ that makes at most t moves,*

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge F \text{ is not an } a\text{-fork for } \mathbf{x} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} \mathsf{rnd} \leftarrow \mathcal{U}(r) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \xleftarrow{\text{tr}^{\text{sr}}} \mathbf{Game}_{\text{SP}}^{\text{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \\ F \leftarrow \mathbf{SRForkFinder}(\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2, \text{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right] \leq (t + 1) \cdot \frac{a - 1}{2^r}.$$

Moreover, **SRForkFinder** runs in expected time $(t + 1) \cdot (a - 1) \cdot (\tau_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}} + \text{poly}(\text{len}(\mathbf{x})))$.

Construction 30.3.3. The algorithm **SRForkFinder**, given as input a game output $(\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2)$ of the SP state-restoration game, a move-response trace tr^{sr} , and black-box access to an SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$, works as follows.

SRForkFinder($\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2, \text{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$):

1. If $\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 0$ or $|\mathbf{x}| > n$, abort.
2. If the move-response pair $((\mathbf{x}, \alpha_1, \sigma), \rho)$ is not in tr^{sr} then append it to tr^{sr} .
3. Lazily sample an oracle $\mathsf{rnd} \leftarrow \mathcal{U}(r)$ that is consistent with the move-response trace tr^{sr} . (Lazily sample an oracle $\mathsf{rnd}' \leftarrow \mathcal{U}(r)$ and program the oracle $\mathsf{rnd} := \mathsf{rnd}'[\text{tr}^{\text{sr}}]$.)
4. Set $S := \{0, 1\}^r \setminus \{\rho\}$ and $B := \{(\rho, \alpha_2)\}$.
5. While S is non-empty and $|B| < a$:
 - a) Sample an SP verifier challenge $\rho' \in S$, and remove ρ' from S .
 - b) Set $\mu := \{((\mathbf{x}, \alpha_1, \sigma), \rho')\}$.
 - c) Run the SP state-restoration game: $(\mathbf{x}', \alpha'_1, \sigma', \rho', \alpha'_2) \leftarrow \mathbf{Game}_{\text{SP}}^{\text{sr}}(s, \mathsf{rnd}[\mu], \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$.
 - d) Compute the decision bit of the SP verifier: $b \leftarrow \mathbf{V}_{\text{SP}}(\mathbf{x}', \alpha'_1, \rho', \alpha'_2)$.
 - e) If $b = 1 \wedge \mathbf{x}' = \mathbf{x} \wedge \alpha'_1 = \alpha_1$ then add (ρ', α'_2) to B .
6. If $|B| < a$ then abort; else output $F := (\alpha_1, B)$.

The above description creates some redundant computation: the moves of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ in every run of the SP state-restoration game $\mathbf{Game}_{\text{SP}}^{\text{sr}}(s, \mathsf{rnd}[\mu], \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$ are the same up to the first move that equals $(\mathbf{x}, \alpha_1, \sigma)$ (if any); hence, this part of the game could be emulated before the while loop, and the remaining part of the game could be emulated in each while loop (each time with a different answer).

Proof of Lemma 30.3.2. Fix any state-restoration randomness rnd , instance \mathbf{x} , SP prover commitment α_1 , and salt σ . We say that $\rho \in \{0, 1\}^r$ is *good* for $(\mathsf{rnd}, \mathbf{x}, \alpha_1, \sigma)$ if there exists an SP prover response α_2 such that:

²This is not an issue in the proof of Lemma 30.2.2 because there **ForkFinder** can rerun only the part of the SP prover $\tilde{\mathbf{P}}_{\text{SP}}$ corresponding to the second round.

1. $|\mathbf{x}| \leq n$;
2. $\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1$; and
3. The output of $\text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}[\mu], \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$ is $(\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2)$ where $\mu = \{((\mathbf{x}, \alpha_1, \sigma), \rho)\}$. (The notation $\text{rnd}[\mu]$ denotes the function rnd re-programmed with the pairs in μ .)

Additionally, define $Y_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma} \in \{0, 1, \dots, 2^r\}$ to be the number of SP verifier challenges $\rho \in \{0, 1\}^r$ that are good for $(\text{rnd}, \mathbf{x}, \alpha_1, \sigma)$; we write Y (without subscripts) to refer to $Y_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma}$ where rnd is the randomness for the SP state-restoration game and $(\mathbf{x}, \alpha_1, \sigma)$ are in the output of the SP state-restoration game.

Let $T := \{(\mathbf{x}', \alpha'_1, \sigma') : |\mathbf{x}'| \leq n, \alpha'_1 \in \{0, 1\}^{p_{V_1}}, \sigma' \in \{0, 1\}^s\}$. We show that

$$\sum_{(\mathbf{x}', \alpha'_1, \sigma') \in T} \Pr \left[Y_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma'} > 0 \mid \text{rnd} \leftarrow \mathcal{U}(r) \right] \leq t + 1. \quad (30.3)$$

For every $\text{rnd} \in \mathcal{U}(r)$, define $S(\text{rnd})$ to be the set of unique moves (i.e., no duplicates) performed by $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ in the SP state-restoration game $\text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$; the set $S(\text{rnd})$ also includes the move $(\mathbf{x}, \alpha_1, \sigma)$ contained in the output of the game in case $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ did not make this move. Note that $|S(\text{rnd})| \leq t + 1$. Moreover, for every $(\mathbf{x}', \alpha'_1, \sigma') \notin S(\text{rnd})$ and $\rho \in \{0, 1\}^r$, programming rnd with $\mu = \{((\mathbf{x}', \alpha'_1, \sigma'), \rho)\}$ does not change the output of the SP state-restoration game, i.e., the output of $\text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}[\mu], \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$ equals that of $\text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$. We argue that if $(\mathbf{x}', \alpha'_1, \sigma') \notin S(\text{rnd})$ then $Y_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma'} = 0$. Indeed, suppose by way of contradiction that $Y_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma'} > 0$, which means that there exists an SP verifier challenge $\rho \in \{0, 1\}^r$ that is good for $(\text{rnd}, \mathbf{x}', \alpha'_1, \sigma')$, which in turn means that, for $\mu = \{((\mathbf{x}', \alpha'_1, \sigma'), \rho)\}$, the output of $\text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}[\mu], \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$ contains $(\mathbf{x}', \alpha'_1, \sigma')$; since $(\mathbf{x}', \alpha'_1, \sigma') \notin S(\text{rnd})$, the output of $\text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}[\mu], \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$ equals that of $\text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$, which lets us conclude that $(\mathbf{x}', \alpha'_1, \sigma') \in S(\text{rnd})$, a contradiction.

This allows us to prove Equation 30.3 as follows:

$$\begin{aligned} & \sum_{(\mathbf{x}', \alpha'_1, \sigma') \in T} \Pr \left[Y_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma'} > 0 \mid \text{rnd} \leftarrow \mathcal{U}(r) \right] \\ &= \sum_{(\mathbf{x}', \alpha'_1, \sigma') \in T} \sum_{\text{rnd}' \in \mathcal{U}(r)} \Pr \left[Y_{\text{rnd}', \mathbf{x}', \alpha'_1, \sigma'} > 0 \right] \cdot \Pr \left[\text{rnd} = \text{rnd}' \mid \text{rnd} \leftarrow \mathcal{U}(r) \right] \\ &= \sum_{\text{rnd}' \in \mathcal{U}(r)} \Pr \left[\text{rnd} = \text{rnd}' \mid \text{rnd} \leftarrow \mathcal{U}(r) \right] \sum_{(\mathbf{x}', \alpha'_1, \sigma') \in T} \Pr \left[Y_{\text{rnd}', \mathbf{x}', \alpha'_1, \sigma'} > 0 \right] \\ &= \sum_{\text{rnd}' \in \mathcal{U}(r)} \Pr \left[\text{rnd} = \text{rnd}' \mid \text{rnd} \leftarrow \mathcal{U}(r) \right] \cdot \sum_{(\mathbf{x}', \alpha'_1, \sigma') \in S(\text{rnd}')} \Pr \left[Y_{\text{rnd}', \mathbf{x}', \alpha'_1, \sigma'} > 0 \right] \\ &\leq \sum_{\text{rnd}' \in \mathcal{U}(r)} \Pr \left[\text{rnd} = \text{rnd}' \mid \text{rnd} \leftarrow \mathcal{U}(r) \right] \cdot (t + 1) \\ &= (t + 1) \cdot \sum_{\text{rnd}' \in \mathcal{U}(r)} \Pr \left[\text{rnd} = \text{rnd}' \mid \text{rnd} \leftarrow \mathcal{U}(r) \right] \\ &= t + 1. \end{aligned}$$

Above, for a fixed rnd' , the probability $\Pr[Y_{\text{rnd}', \mathbf{x}', \alpha'_1, \sigma'} > 0]$ is 0 or 1, but it is convenient to use the probability notation.

Denote by $Z_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma, \rho} \in \{0, 1\}$ the indicator for the event that ρ is good for $(\text{rnd}, \mathbf{x}, \alpha_1, \sigma)$; we write Z (without subscripts) to refer to $Z_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma, \rho}$ where rnd is the randomness for the SP state-restoration game and $(\mathbf{x}, \alpha_1, \sigma, \rho)$ are in the output of the SP state-restoration game.

For every $(\mathbf{x}', \alpha'_1, \sigma') \in T$ and $\ell \in \{0, 1, \dots, 2^r\}$, we argue that

$$\Pr \left[\begin{array}{l} Z_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma, \rho} = 1 \\ \wedge (\mathbf{x}, \alpha_1, \sigma) = (\mathbf{x}', \alpha'_1, \sigma') \\ \text{conditioned on} \\ Y_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma} = \ell \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \leftarrow \text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right] = \frac{\ell}{2^r}. \quad (30.4)$$

First suppose that $\ell = 0$: if $Y_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma} = 0$ then $Z_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma, \rho} = 0$, so the probability equals 0 in this case. Next, if $\ell \in \{1, \dots, 2^r\}$, we argue as follows:

$$\begin{aligned} & \Pr \left[\begin{array}{l} Z_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma, \rho} = 1 \\ \wedge (\mathbf{x}, \alpha_1, \sigma) = (\mathbf{x}', \alpha'_1, \sigma') \\ \text{conditioned on} \\ Y_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma} = \ell \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \leftarrow \text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right] \\ &= \Pr \left[\begin{array}{l} Z_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma', \rho'} = 1 \\ \wedge (\mathbf{x}, \alpha_1, \sigma) = (\mathbf{x}', \alpha'_1, \sigma') \\ \text{conditioned on} \\ Y_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma'} = \ell \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \leftarrow \text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \\ \rho' := \text{rnd}(\mathbf{x}', \alpha'_1, \sigma') \end{array} \right] \\ &= \Pr \left[\begin{array}{l} Z_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma', \rho'} = 1 \\ \text{conditioned on} \\ Y_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma'} = \ell \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ \rho' := \text{rnd}(\mathbf{x}', \alpha'_1, \sigma') \end{array} \right] = \frac{\ell}{2^r}. \end{aligned}$$

Above, the second equality holds because the condition $Y_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma'} = \ell > 0$ implies that $(\mathbf{x}', \alpha'_1, \sigma')$ is part of the output of the state-restoration game and thus $(\mathbf{x}, \alpha_1, \sigma) = (\mathbf{x}', \alpha'_1, \sigma')$.

Error probability We upper bound the error probability of **SRForkFinder**. The event in the probability in the lemma statement implies the event $Z = 1 \wedge Y \leq a - 1$, that is, $Z_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma, \rho} = 1 \wedge Y_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma} \leq a - 1$ where rnd is the randomness for the SP state-restoration game and $(\mathbf{x}, \alpha_1, \sigma, \rho)$ are in the output of the SP state-restoration game. Indeed, the conditions $|\mathbf{x}| \leq n$ and $\mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1$ imply that $Z = 1$ and the condition that F is not an a -fork for \mathbf{x} implies that $Y \leq a - 1$ (since if there are at least a SP verifier challenges that are good for $(\text{rnd}, \mathbf{x}, \alpha_1, \sigma)$ then **SRForkFinder** outputs an a -fork for \mathbf{x}).

We upper bound the error probability as follows:

$$\begin{aligned} & \Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge F \text{ is not an } a\text{-fork for } \mathbf{x} \\ \wedge \mathbf{V}_{\text{SP}}(\mathbf{x}, \alpha_1, \rho, \alpha_2) = 1 \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \xleftarrow{\text{tr}^{\text{sr}}} \text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \\ F \leftarrow \text{SRForkFinder}(\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2, \text{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right] \\ &\leq \Pr \left[\begin{array}{l} Z_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma, \rho} = 1 \\ \wedge Y_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma} \leq a - 1 \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \leftarrow \text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right] \\ &= \sum_{(\mathbf{x}', \alpha'_1, \sigma') \in T} \Pr \left[\begin{array}{l} Z_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma, \rho} = 1 \\ \wedge Y_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma} \leq a - 1 \\ \wedge (\mathbf{x}, \alpha_1, \sigma) = (\mathbf{x}', \alpha'_1, \sigma') \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \leftarrow \text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right] \\ &= \sum_{(\mathbf{x}, \alpha_1, \sigma) \in T} \sum_{\ell=1}^{a-1} \Pr \left[\begin{array}{l} Z_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma, \rho} = 1 \\ \wedge (\mathbf{x}, \alpha_1, \sigma) = (\mathbf{x}', \alpha'_1, \sigma') \\ \text{conditioned on} \\ Y_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma} = \ell \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \leftarrow \text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right] \end{aligned}$$

$$\begin{aligned}
& \cdot \Pr \left[Y_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma'} = \ell \mid \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \right] \\
& \leq \sum_{(\mathbf{x}', \alpha'_1, \sigma') \in T} \sum_{\ell=1}^{a-1} \frac{\ell}{2^r} \cdot \Pr \left[Y_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma'} = \ell \mid \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \right] \quad (\text{by Equation 30.4}) \\
& \leq \frac{a-1}{2^r} \cdot \sum_{(\mathbf{x}', \alpha'_1, \sigma') \in T} \Pr \left[Y_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma'} > 0 \mid \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \right] \\
& \leq (t+1) \cdot \frac{a-1}{2^r} \quad (\text{by Equation 30.3}) .
\end{aligned}$$

The while loop Let X be the number of times **SRForkFinder** enters the while loop. We argue that $\mathbb{E}[X] \leq (t+1) \cdot (a-1)$. The expectation is over all randomness in the experiment: the randomness rnd of the SP state-restoration game and the internal randomness ζ of **SRForkFinder**.

If ρ is not good for $(\text{rnd}, \mathbf{x}, \alpha_1, \sigma)$ then **SRForkFinder** aborts in Item 1, so $\mathbb{E}[X \mid Z=0]=0$. If instead ρ is good for $(\text{rnd}, \mathbf{x}, \alpha_1, \sigma)$ then **SRForkFinder** reaches the while loop. We argue that, for every $\ell \in \{1, \dots, 2^r\}$,

$$\mathbb{E}[X \mid Z=1 \wedge Y=\ell] \leq \frac{(a-1) \cdot 2^r}{\ell}. \quad (30.5)$$

We consider two cases.

- If $\ell < a$ then **SRForkFinder** enters the while loop $2^r - 1$ times and then aborts (there are not enough good SP verifier challenges). Hence, $\mathbb{E}[X \mid Y=\ell] = 2^r - 1 = \frac{\ell \cdot (2^r - 1)}{\ell} \leq \frac{(a-1) \cdot (2^r - 1)}{\ell} < \frac{(a-1) \cdot 2^r}{\ell}$.
- If $\ell \geq a$ then **SRForkFinder** enters the while loop and stops if it finds $a-1$ SP verifier challenges that are good for $(\text{rnd}, \mathbf{x}, \alpha_1, \sigma)$ (it might stop earlier if it finds good challenges for $\sigma' \neq \sigma$).

Specifically, **SRForkFinder** samples without replacement from a set of cardinality $2^r - 1$ until it obtains $a-1$ “successes”. Therefore, X is bounded by a random variable X_{NHG} defined for the negative hypergeometric distribution (see Definition 30.1.7) with a set size $2^r - 1$, a total of $\ell - 1$ blue elements, and a target of $a-1$ blue elements. Hence, the expected number of times **SRForkFinder** enters the while loop is $\mathbb{E}[X \mid Y=\ell] \leq \frac{(a-1) \cdot 2^r}{\ell}$.

Therefore, by the total expectation theorem,

$$\begin{aligned}
& \mathbb{E}[X] \\
& = \mathbb{E}[X \mid Z=0] \cdot \Pr[Z=0] + \mathbb{E}[X \mid Z=1] \cdot \Pr[Z=1] \\
& = 0 + \mathbb{E}[X \mid Z=1] \cdot \Pr[Z=1] \\
& = \sum_{\ell=1}^{2^r} \mathbb{E}[X \mid Z=1 \wedge Y=\ell] \cdot \Pr[Z=1 \wedge Y=\ell] \quad (Z=1 \text{ implies } Y>0) \\
& \leq \sum_{\ell=1}^{2^r} \frac{(a-1) \cdot 2^r}{\ell} \cdot \Pr[Z=1 \wedge Y=\ell] \quad (\text{by Equation 30.5}) \\
& = \sum_{\ell=1}^{2^r} \frac{(a-1) \cdot 2^r}{\ell} \cdot \sum_{(\mathbf{x}', \alpha'_1, \sigma') \in T} \Pr \left[\begin{array}{l} Z_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma, \rho} = 1 \\ \wedge Y_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma} = \ell \\ \wedge (\mathbf{x}, \alpha_1, \sigma) = (\mathbf{x}', \alpha'_1, \sigma') \end{array} \mid \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \leftarrow \text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right]
\end{aligned}$$

$$\begin{aligned}
&= \sum_{\ell=1}^{2^r} \frac{(a-1) \cdot 2^r}{\ell} \cdot \sum_{(\mathbf{x}', \alpha'_1, \sigma') \in T} \Pr \left[\begin{array}{l} Z_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma, \rho} = 1 \\ \wedge (\mathbf{x}, \alpha_1, \sigma) = (\mathbf{x}', \alpha'_1, \sigma') \\ \text{conditioned on} \\ Y_{\text{rnd}, \mathbf{x}, \alpha_1, \sigma} = \ell \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \\ (\mathbf{x}, \alpha_1, \sigma, \rho, \alpha_2) \leftarrow \text{Game}_{\text{SP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}) \end{array} \right] \\
&\quad \cdot \Pr \left[Y_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma'} = \ell \mid \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \right] \\
&\leq \sum_{\ell=1}^{2^r} \frac{(a-1) \cdot 2^r}{\ell} \cdot \sum_{(\mathbf{x}', \alpha'_1, \sigma') \in T} \frac{\ell}{2^r} \cdot \Pr \left[Y_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma'} = \ell \mid \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \right] \quad (\text{by Equation 30.4}) \\
&= (a-1) \cdot \sum_{\ell=1}^{2^r} \sum_{(\mathbf{x}', \alpha'_1, \sigma') \in T} \Pr \left[Y_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma'} = \ell \mid \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \right] \\
&= (a-1) \cdot \sum_{(\mathbf{x}', \alpha'_1, \sigma') \in T} \sum_{\ell=1}^{2^r} \Pr \left[Y_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma'} = \ell \mid \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \right] \\
&= (a-1) \cdot \sum_{(\mathbf{x}', \alpha'_1, \sigma') \in T} \Pr \left[Y_{\text{rnd}, \mathbf{x}', \alpha'_1, \sigma'} > 0 \mid \text{rnd} \leftarrow \mathcal{U}(\mathbf{r}) \right] \\
&\leq (t+1) \cdot (a-1) \quad (\text{by Equation 30.3}).
\end{aligned}$$

Running time We analyze the expected running time of **SRForkFinder**. First we analyze how **SRForkFinder** invokes $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$. The algorithm **SRForkFinder**, in each iteration of the while loop, simulates an execution of the SP state-restoration game with $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ (see Item 5c). In expectation this happens $\mathbb{E}[X] \leq (t+1) \cdot (a-1)$ times. Overall, since $\tau_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}$ denotes the total running time of $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ (i.e., the running time within an execution of the SP state-restoration game), the total time spent by **SRForkFinder** invoking $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$ is upper bounded by $(t+1) \cdot (a-1) \cdot \tau_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}$. All other operations in **SRForkFinder** take time $\text{poly}(\text{len}(\mathbf{x}))$, which in expectation across all while loop iterations add up to time $(t+1) \cdot (a-1) \cdot \text{poly}(\text{len}(\mathbf{x}))$. \square

Construction 30.3.4. Let $\mathbf{E}_{\text{SP}}^{\text{ss}}$ be the special soundness extractor for SP, and let **SRForkFinder** be the algorithm in Construction 30.3.3. The rewinding knowledge extractor $\mathbf{E}_{\text{SP}}^{\text{sr}}$ receives as input an SP state-restoration game output $(\mathbf{x}, \alpha_1, \rho, \sigma, \alpha_2)$, the move-response trace tr^{sr} , and black-box access to an SP state-restoration prover $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}$, and works as follows.

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

1. Run $F \leftarrow \text{SRForkFinder}(\mathbf{x}, \alpha_1, \rho, \sigma, \alpha_2, \text{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}})$; if **SRForkFinder** aborts then abort.
2. Run $\mathbf{w} \leftarrow \mathbf{E}_{\text{SP}}^{\text{ss}}(\mathbf{x}, F)$.
3. Output \mathbf{w} .

Proof of Theorem 30.3.1. The knowledge extractor $\mathbf{E}_{\text{SP}}^{\text{sr}}$ succeeds if the algorithm **SRForkFinder** succeeds, so the knowledge soundness error of $\mathbf{E}_{\text{SP}}^{\text{sr}}$ is at most the failure probability of **SRForkFinder**. This latter, by Lemma 30.3.2, is at most $(t+1) \cdot \frac{a-1}{2^r}$. Hence the knowledge soundness error of $\mathbf{E}_{\text{SP}}^{\text{sr}}$ is as claimed in Theorem 30.3.1.

Moreover, the expected running time of $\mathbf{E}_{\text{SP}}^{\text{sr}}$ is the expected running time of **SRForkFinder** plus the running time of $\mathbf{E}_{\text{SP}}^{\text{ss}}$. By Lemma 30.3.2, the expected running time of **SRForkFinder** is

at most $(t + 1) \cdot (a - 1) \cdot (\tau_{\tilde{\mathbf{P}}_{\text{SP}}} + \text{poly}(\text{len}(\mathbf{x})))$. By definition, the running time of $\mathbf{E}_{\text{SP}}^{\text{ss}}$ is $\mathbf{et}_{\text{SP}}^{\text{ss}}(\mathbf{x})$. We conclude that the expected running time of $\mathbf{E}_{\text{SP}}^{\text{sr}}$ is as claimed in Theorem 30.3.1. \square

30.4 Knowledge soundness for IPs

We prove that if an IP has special soundness then the IP has rewinding knowledge soundness. The rewinding knowledge soundness error and extraction time increase as the arity of the special soundness increases; neither depends on the failure probability of the IP prover.

Theorem 30.4.1. *Let $\text{IP} = (\mathbf{P}_{\text{IP}}, \mathbf{V}_{\text{IP}})$ be a public-coin IP with round complexity k . If IP has special soundness with arity (a_1, \dots, a_k) with extraction time $\mathbf{et}_{\text{IP}}^{\text{ss}}$ (see Definition 30.1.5) then IP has rewinding knowledge soundness error κ_{IP} with extraction time \mathbf{et}_{IP} (see Definition 13.1.5) such that*

$$\begin{aligned}\kappa_{\text{IP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{IP}}}) &\leq 1 - \prod_{i \in [k]} \left(1 - \frac{a_i - 1}{2^{r_i}}\right) \leq \sum_{i \in [k]} \frac{a_i - 1}{2^{r_i}}, \\ \mathbf{et}_{\text{IP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{IP}}}, \tau_{\tilde{\mathbf{P}}_{\text{IP}}}) &\leq \left(\prod_{i \in [k]} a_i - 1\right) \cdot \left(\tau_{\tilde{\mathbf{P}}_{\text{IP}}} + \text{poly}(\text{len}(\mathbf{x}))\right) + \mathbf{et}_{\text{IP}}^{\text{ss}}(\mathbf{x}).\end{aligned}$$

In particular, if $r = r_1 = \dots = r_k$ and $a = a_1 = \dots = a_k$ then the knowledge soundness error and expected extraction time are

$$\begin{aligned}\kappa_{\text{IP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{IP}}}) &\leq \frac{k \cdot (a - 1)}{2^r}, \\ \mathbf{et}_{\text{IP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{IP}}}, \tau_{\tilde{\mathbf{P}}_{\text{IP}}}) &\leq (a^k - 1) \cdot \left(\tau_{\tilde{\mathbf{P}}_{\text{IP}}} + \text{poly}(\text{len}(\mathbf{x}))\right) + \mathbf{et}_{\text{IP}}^{\text{ss}}(\mathbf{x}).\end{aligned}$$

Fix an instance \mathbf{x} and IP prover $\tilde{\mathbf{P}}_{\text{IP}}$. The main idea to prove the theorem is to rewind $\tilde{\mathbf{P}}_{\text{IP}}$ several times with correlated inputs in order to obtain an (a_1, \dots, a_k) -tree T for \mathbf{x} . This suffices because the knowledge extractor $\mathbf{E}_{\text{IP}}^{\text{ss}}$ for special soundness can find a witness w from \mathbf{x} and T .

The lemma below provides a probabilistic procedure **TreeFinder** that outputs an (a_1, \dots, a_k) -tree T . The procedure **TreeFinder** receives as input the instance \mathbf{x} , an interaction transcript $((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ between the IP prover $\tilde{\mathbf{P}}_{\text{IP}}$ and IP verifier \mathbf{V}_{IP} , and black-box access to $\tilde{\mathbf{P}}_{\text{IP}}$. We describe how to upper bound the probability that $\tilde{\mathbf{P}}_{\text{IP}}$ convinces \mathbf{V}_{IP} and the output of **TreeFinder** is not an (a_1, \dots, a_k) -tree T for \mathbf{x} . Informally, **TreeFinder** invokes $\tilde{\mathbf{P}}_{\text{IP}}$ with different IP verifier challenges until it obtains $\prod_{i \in [k]} a_i - 1$ additional accepting interaction transcripts. (The first accepting interaction transcript comes “for free” when the IP verifier \mathbf{V}_{IP} accepts in the real interaction.)

Lemma 30.4.2. *The algorithm **TreeFinder** in Construction 30.4.3 satisfies the following property. For every IP prover $\tilde{\mathbf{P}}_{\text{IP}}$ and instance \mathbf{x} ,*

$$\Pr \left[\begin{array}{l} T \text{ is not an } (a_1, \dots, a_k)\text{-tree for } \mathbf{x} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} b \xleftarrow{((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})} \langle \tilde{\mathbf{P}}_{\text{IP}}, \mathbf{V}_{\text{IP}}(\mathbf{x}) \rangle_{\text{IP}} \\ T \leftarrow \text{TreeFinder}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}}) \end{array} \right] \leq 1 - \prod_{i \in [k]} \left(1 - \frac{a_i - 1}{2^{r_i}}\right).$$

Moreover, **TreeFinder** runs in expected time $(\prod_{i \in [k]} a_i - 1) \cdot (\tau_{\tilde{\mathbf{P}}_{\text{IP}}} + \text{poly}(\text{len}(\mathbf{x})))$.

The algorithm TreeFinder is the base case of a recursively defined helper function TreeFinder_j , whose goal is to find, given an instance \mathbf{x} and partial transcript $((\alpha'_i)_{i=1}^j, (\rho'_i)_{i=1}^j)$, an (a_{j+1}, \dots, a_k) -subtree of (accepting) transcripts for the instance \mathbf{x} and partial transcript $((\alpha'_i)_{i=1}^j, (\rho'_i)_{i=1}^j)$. The base case invoked by TreeFinder thus corresponds to $j = 0$.

In more detail, TreeFinder_j invokes TreeFinder_{j+1} multiple times, each time with a different IP verifier message ρ'_{j+1} , until TreeFinder_j obtains a_{j+1} subtrees for different IP verifier messages (or else TreeFinder_j outputs the error message fail). Then TreeFinder_j combines these subtrees to obtain the desired (a_{j+1}, \dots, a_k) -subtree. Two delicate technical details complicate the strategy.

- *Early abort.* If the first invocation of TreeFinder_{j+1} by TreeFinder_j results in a failure message (no valid subtree is returned) then TreeFinder_j immediately halts and outputs a failure message. (Without making any further attempts.) This *early abort* condition ensures, as will be shown in the analysis, that the expected running time of TreeFinder_j is suitably bounded.
- *Real transcript.* The algorithm TreeFinder_j additionally receives as input a complete interaction transcript $((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ that corresponds to the “real” interaction between the IP prover $\tilde{\mathbf{P}}_{\text{IP}}$ and IP verifier \mathbf{V}_{IP} . One (and only one) of the paths in the tree output by TreeFinder_j (if not a failure message) corresponds to $((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$, which can be viewed as a “free” path (provided $((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ is accepting). To accommodate for this, the left-most path of the tree is arbitrarily reserved for $((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$, which TreeFinder_j realizes by receiving a flag f that is set when on the left-most branch. Specifically, if $f = 1$ then TreeFinder_j makes the first invocation of TreeFinder_{j+1} with ρ_{j+1} (and all other invocations of TreeFinder_{j+1} with $f = 0$ and different random strings in $\{0, 1\}^{r_{j+1}}$). Instead, if $f = 0$ then TreeFinder_j ignores $((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$, and all invocations of TreeFinder_{j+1} are with different random strings in $\{0, 1\}^{r_{j+1}}$.

The above summary is made precise in the construction below.

Construction 30.4.3. The algorithm TreeFinder , given as input an instance \mathbf{x} , an IP interaction transcript $((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$, and black-box access to an IP prover $\tilde{\mathbf{P}}_{\text{IP}}$, works as follows.

$\text{TreeFinder}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}}) := \text{TreeFinder}_0(1, \mathbf{x}, \emptyset, \emptyset, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}})$, where TreeFinder_j is recursively defined for $j \in \{0, 1, \dots, k\}$ below.

$\text{TreeFinder}_j(f, \mathbf{x}, (\alpha'_i)_{i=1}^j, (\rho'_i)_{i=1}^j, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}})$:

1. If $j = k$ (the base case):
 - a) Compute the decision bit of the IP verifier: $b := \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha'_i)_{i=1}^j, (\rho'_i)_{i=1}^j)$.
 - b) If $b = 1$ output the (trivial) tree T that consists of a single (root) vertex.
 - c) If $b = 0$ output the error message fail .
2. Compute the next IP prover message $\alpha'_{j+1} := \tilde{\mathbf{P}}_{\text{IP}}((\rho'_i)_{i=1}^j)$.
3. If $f = 1$ set $\rho'_{j+1} := \rho_{j+1}$; else ($f = 0$) set ρ'_{j+1} to be a random string in $\{0, 1\}^{r_{j+1}}$.
4. Compute:

$$T_{\rho'_{j+1}} \leftarrow \text{TreeFinder}_{j+1}(f, \mathbf{x}, (\alpha'_i)_{i=1}^j \| \alpha'_{j+1}, (\rho'_i)_{i=1}^j \| \rho'_{j+1}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}}).$$

5. If $T_{\rho'_{j+1}} = \text{fail}$ then output fail .

6. Set $S := \{0, 1\}^{r_{j+1}} \setminus \{\rho'_{j+1}\}$ and $B := \{(\rho'_{j+1}, T_{\rho'_{j+1}})\}$.
7. While S is non-empty and $|B| < a_{j+1}$:
 - a) Sample a random string $\rho'_{j+1} \in S$, and remove ρ'_{j+1} from S .
 - b) Compute

$$T_{\rho'_{j+1}} \leftarrow \text{TreeFinder}_{j+1}(0, \mathbf{x}, (\alpha'_i)_{i=1}^j \| \alpha'_{j+1}, (\rho'_i)_{i=1}^j \| \rho'_{j+1}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}})$$
 c) If $T_{\rho'_{j+1}} \neq \text{fail}$ then add $(\rho'_{j+1}, T_{\rho'_{j+1}})$ to B .
8. If $|B| < a_{j+1}$ then output the error message `fail`.
9. Let T be the tree obtained as follows: (i) the root is labeled by α'_{j+1} ; (ii) for every $(\rho'_{j+1}, T_{\rho'_{j+1}}) \in B$, connect the root to the root of $T_{\rho'_{j+1}}$ and label the edge by ρ'_{j+1} . Note that the root of T has a_{j+1} children.
10. Output T .

Proof of Lemma 30.4.2. We prove a lower bound on the probability of the event negation, namely, we prove that the probability that T is an (a_1, \dots, a_k) -tree for \mathbf{x} or $b = 0$ is at least $\prod_{i \in [k]} (1 - \frac{a_i - 1}{2^{r_i}})$.

If $b = 0$ then `TreeFinder` receives a rejecting interaction transcript and outputs the error message `fail` (in particular, T is not a valid tree for \mathbf{x}), so the above two conditions are disjoint. We deduce that

$$\begin{aligned} & \Pr \left[\begin{array}{l} T \text{ is an } (a_1, \dots, a_k)\text{-tree for } \mathbf{x} \\ \vee b = 0 \end{array} \right] \\ &= \Pr[T \text{ is an } (a_1, \dots, a_k)\text{-tree for } \mathbf{x}] + \Pr[b = 0] \\ &\geq \left(\prod_{i \in [k]} \left(1 - \frac{a_i - 1}{2^{r_i}} \right) - \Pr[b = 0] \right) + \Pr[b = 0] = \prod_{i \in [k]} \left(1 - \frac{a_i - 1}{2^{r_i}} \right). \end{aligned}$$

In Claim 30.4.15 we prove the inequality above. Moreover in Claim 30.4.18 we prove that `TreeFinder` runs in expected time $(\prod_{i \in [k]} a_i - 1) \cdot (\tau_{\tilde{\mathbf{P}}_{\text{IP}}} + \text{poly}(\text{len}(\mathbf{x})))$. Establishing both claims is technical, as it involves a number of intermediate claims and computations; the details are in Section 30.4.1. \square

Construction 30.4.4. Let $\mathbf{E}_{\text{IP}}^{\text{ss}}$ be the special soundness extractor for IP , and let `TreeFinder` be the algorithm in Construction 30.4.3. The rewinding knowledge extractor \mathbf{E}_{IP} receives as input an instance \mathbf{x} , an interaction transcript $((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$, and black-box access to an IP prover $\tilde{\mathbf{P}}_{\text{IP}}$, and works as follows.

$\mathbf{E}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}})$:

1. Run $T \leftarrow \text{TreeFinder}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}})$; if $T = \text{fail}$ then abort.
2. Run $\mathbf{w} \leftarrow \mathbf{E}_{\text{IP}}^{\text{ss}}(\mathbf{x}, T)$.
3. Output \mathbf{w} .

Proof of Theorem 30.4.1. The knowledge extractor \mathbf{E}_{IP} succeeds if the algorithm `TreeFinder` succeeds, so the knowledge soundness error of \mathbf{E}_{IP} is at most the failure probability of `TreeFinder`. This latter, by Lemma 30.4.2, is at most $1 - \prod_{i \in [k]} (1 - \frac{a_i - 1}{2^{r_i}})$. Hence the knowledge soundness error of \mathbf{E}_{IP} is as claimed in Theorem 30.4.1.

Moreover, the expected running time of \mathbf{E}_{IP} is the expected running time of TreeFinder plus the running time of $\mathbf{E}_{\text{IP}}^{\text{ss}}$. By Lemma 30.4.2, the expected running time of TreeFinder is at most $(\prod_{i \in [k]} a_i - 1) \cdot (\tau_{\tilde{\mathbf{P}}_{\text{IP}}} + \text{poly}(\text{len}(\mathbf{x})))$. By definition, the running time of $\mathbf{E}_{\text{IP}}^{\text{ss}}$ is $\mathbf{et}_{\text{IP}}^{\text{ss}}(\mathbf{x})$. We conclude that the expected running time of \mathbf{E}_{IP} is as claimed in Theorem 30.4.1. \square

30.4.1 Technical details

In the proof of Lemma 30.4.2 we relied on two main claims: Claim 30.4.15 (bound on the error probability) and Claim 30.4.18 (bound on the expected running time). The goal of this section is to prove these claims; in each case, this requires additional notation and intermediate claims.

Preliminaries We establish basic claims and notation for the analysis.

Claim 30.4.5. *The following two random variables are identical:*

$$\begin{aligned} & \left\{ \text{TreeFinder}_0(1, \mathbf{x}, \emptyset, \emptyset, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}}) \mid b \xleftarrow{((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})} \langle \tilde{\mathbf{P}}_{\text{IP}}, \mathbf{V}_{\text{IP}}(\mathbf{x}) \rangle_{\text{IP}} \right\} \\ & \equiv \text{TreeFinder}_0(0, \mathbf{x}, \emptyset, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\text{IP}}). \end{aligned}$$

Proof. The equality holds because: (a) on the right side of the equality, TreeFinder₀ with flag $f = 0$ samples without replacement a fresh SP verifier message for each recursive call; (b) on the left side of the equality, TreeFinder₀ with flag $f = 1$ uses the given SP verifier messages $(\rho_i)_{i \in [k]}$ for each recursive call on the left-most branch of the recursion tree (and samples without replacement fresh SP verifier random messages for other recursive calls); (c) the SP verifier messages $(\rho_i)_{i \in [k]}$ are sampled at random. \square

Part of the analysis studies TreeFinder_j in the case when the flag $f = 0$, in which case TreeFinder_j ignores the IP transcript $((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$. We define a short-hand notation for this case. Below, for $i \in [k]$, $\boldsymbol{\rho}_i$ denotes a list of IP verifier messages (ρ_1, \dots, ρ_i) where, for each j , $\rho_j \in \{0, 1\}^{r_j}$.

Definition 30.4.6. *For every $i \in \{0, \dots, k\}$,*

$$\text{TreeFinder}_i(\mathbf{x}, \boldsymbol{\rho}_i, \tilde{\mathbf{P}}_{\text{IP}}) := \text{TreeFinder}_i(0, \mathbf{x}, \boldsymbol{\alpha}_i, \boldsymbol{\rho}_i, \cdot, \cdot, \tilde{\mathbf{P}}_{\text{IP}}),$$

where $\boldsymbol{\alpha}_i$ are the IP prover messages of $\tilde{\mathbf{P}}_{\text{IP}}$ for the IP verifier messages $\boldsymbol{\rho}_i$. The dots denote the fact that those inputs do not matter.

Definition 30.4.7. *For every $i \in \{0, 1, \dots, k-1\}$, IP verifier messages $\boldsymbol{\rho}_i$, and randomness ζ for TreeFinder_i, we define $Z_{i, \boldsymbol{\rho}_i, \zeta}$ to be the indicator for the event $\text{TreeFinder}_i(\mathbf{x}, \boldsymbol{\rho}_i, \tilde{\mathbf{P}}_{\text{IP}}; \zeta) \neq \text{fail}$.*

Definition 30.4.8. *For every round index $i \in [k]$, IP verifier messages $\boldsymbol{\rho}_{i-1}$, and randomness ζ for TreeFinder_i, we say that $\rho_i \in \{0, 1\}^{r_i}$ is **good** for $(\boldsymbol{\rho}_{i-1}, \zeta)$ if*

$$\text{TreeFinder}_i(\mathbf{x}, \boldsymbol{\rho}_{i-1} \| \rho_i, \tilde{\mathbf{P}}_{\text{IP}}; \zeta) \neq \text{fail}.$$

We define $Y_{i, \boldsymbol{\rho}_{i-1}, \zeta}$ to be the number of good $\rho_i \in \{0, 1\}^{r_i}$ for $(\boldsymbol{\rho}_{i-1}, \zeta)$.

Claim 30.4.9. *For every round index $i \in [k]$ and IP verifier messages ρ_{i-1} ,*

$$\Pr_{\zeta} [Z_{i,\rho_{i-1}\|\rho_i(\zeta),\zeta} = 1 \mid Y_{i,\rho_{i-1},\zeta} = \ell] = \frac{\ell}{2^{r_i}},$$

where $\rho_i(\zeta)$ is the first IP verifier message sampled by $\text{TreeFinder}_{i-1}(\mathbf{x}, \rho_{i-1}, \tilde{\mathbf{P}}_{\text{IP}}; \zeta)$.

Proof. The condition $Y_{i,\rho_{i-1},\zeta} = \ell$ tells us that the number of good $\rho_i \in \{0, 1\}^{r_i}$ for (ρ_{i-1}, ζ) is ℓ . Hence the probability, over a choice of randomness ζ for TreeFinder_{i-1} , that the first IP verifier message $\rho_i(\zeta)$ sampled by $\text{TreeFinder}_{i-1}(\mathbf{x}, \rho_{i-1}, \tilde{\mathbf{P}}_{\text{IP}}; \zeta)$ is good (i.e., such that the output of $\text{TreeFinder}_i(\mathbf{x}, \rho_{i-1}\|\rho_i(\zeta), \tilde{\mathbf{P}}_{\text{IP}}; \zeta)$ is not fail) is precisely $\frac{\ell}{2^{r_i}}$. \square

Error probability Here we focus on proving Claim 30.4.15.

Claim 30.4.10. *For every round index $i \in [k]$ and IP verifier messages ρ_{i-1} ,*

$$\sum_{\ell=0}^{a_i-1} \Pr_{\zeta} [Y_{i,\rho_{i-1},\zeta} = \ell] \leq \frac{2^{r_i} - \mathbb{E}_{\zeta} [Y_{i,\rho_{i-1},\zeta}]}{2^{r_i} - a_i + 1}.$$

Proof. The inequality follows by rearranging the first and last term in this derivation:

$$\begin{aligned} \mathbb{E}_{\zeta} [Y_{i,\rho_{i-1},\zeta}] &= \sum_{\ell=0}^{2^{r_i}} \ell \cdot \Pr_{\zeta} [Y_{i,\rho_{i-1},\zeta} = \ell] \quad (\text{definition of expectation}) \\ &= \sum_{\ell=0}^{a_i-1} \ell \cdot \Pr_{\zeta} [Y_{i,\rho_{i-1},\zeta} = \ell] + \sum_{\ell=a_i}^{2^{r_i}} \ell \cdot \Pr_{\zeta} [Y_{i,\rho_{i-1},\zeta} = \ell] \quad (\text{splitting the sum}) \\ &\leq (a_i - 1) \cdot \sum_{\ell=0}^{a_i-1} \Pr_{\zeta} [Y_{i,\rho_{i-1},\zeta} = \ell] + 2^{r_i} \cdot \sum_{\ell=a_i}^{2^{r_i}} \Pr_{\zeta} [Y_{i,\rho_{i-1},\zeta} = \ell] \\ &= (a_i - 1) \cdot \sum_{\ell=0}^{a_i-1} \Pr_{\zeta} [Y_{i,\rho_{i-1},\zeta} = \ell] + 2^{r_i} \cdot \left(1 - \sum_{\ell=0}^{a_i-1} \Pr_{\zeta} [Y_{i,\rho_{i-1},\zeta} = \ell]\right) \\ &= 2^{r_i} - (2^{r_i} - a_i + 1) \cdot \sum_{\ell=0}^{a_i-1} \Pr_{\zeta} [Y_{i,\rho_{i-1},\zeta} = \ell]. \end{aligned}$$

\square

Definition 30.4.11. *For every $i \in \{0, \dots, k-1\}$ and IP verifier messages ρ_i ,*

$$w(\rho_i) := \Pr \left[\langle \tilde{\mathbf{P}}_{\text{IP}}, \mathbf{V}_{\text{IP}}(\mathbf{x}; \rho_i \| \rho_{i+1} \| \dots \| \rho_k) \rangle_{\text{IP}} = 1 \mid \begin{array}{c} \rho_{i+1} \leftarrow \{0, 1\}^{r_{i+1}} \\ \vdots \\ \rho_k \leftarrow \{0, 1\}^{r_k} \end{array} \right].$$

Note that $w(\emptyset) = \Pr[b = 1]$, and $w(\rho_k) \in \{0, 1\}$ (depending on whether $\tilde{\mathbf{P}}_{\text{IP}}$ convinces $\mathbf{V}_{\text{IP}}(\mathbf{x}; \rho_k)$).

Definition 30.4.12. *For every $i \in \{0, \dots, k-1\}$, $\kappa_i := 1 - \prod_{j=i+1}^k \left(1 - \frac{a_j-1}{2^{r_j}}\right)$; moreover, $\kappa_k := 0$.*

Claim 30.4.13. For every $i \in \{0, \dots, k - 2\}$, $\kappa_i - \kappa_{i+1} = \prod_{j=i+2}^k \left(1 - \frac{a_j - 1}{2^{r_j}}\right) \cdot \left(\frac{a_{i+1} - 1}{2^{r_{i+1}}}\right)$.

Proof. The equality directly follows from Definition 30.4.12:

$$\begin{aligned} \kappa_i - \kappa_{i+1} &= \prod_{j=i+2}^k \left(1 - \frac{a_j - 1}{2^{r_j}}\right) - \prod_{j=i+1}^k \left(1 - \frac{a_j - 1}{2^{r_j}}\right) \quad (\text{by Definition 30.4.12}) \\ &= \prod_{j=i+2}^k \left(1 - \frac{a_j - 1}{2^{r_j}}\right) \cdot \left(1 - \left(1 - \frac{a_{i+1} - 1}{2^{r_{i+1}}}\right)\right) \\ &= \prod_{j=i+2}^k \left(1 - \frac{a_j - 1}{2^{r_j}}\right) \cdot \left(\frac{a_{i+1} - 1}{2^{r_{i+1}}}\right). \end{aligned}$$

□

Claim 30.4.14. For every $i \in \{0, \dots, k\}$ and IP verifier messages ρ_i ,

$$\Pr_{\zeta} [\text{TreeFinder}_i(\mathbf{x}, \rho_i, \tilde{\mathbf{P}}_{\text{IP}}; \zeta) \neq \text{fail}] \geq \left(\prod_{j=i+1}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot (w(\rho_i) - \kappa_i).$$

Proof. We prove the claim by (reverse) strong induction on i . The base case is $i = k$, in which case the right-side of the inequality is zero, and thus the claim holds (trivially). Next, we discuss the inductive case, where we assume that the claim holds for all $j \in \{i+1, \dots, k\}$.

First, we use the strong induction hypothesis to prove the following inequality:

$$\mathbb{E}_{\zeta}[Y_{i+1, \rho_i, \zeta}] \geq 2^{r_{i+1}} \cdot \left(\prod_{j=i+2}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot (w(\rho_i) - \kappa_{i+1}). \quad (30.6)$$

Indeed:

$$\begin{aligned} &\mathbb{E}_{\zeta}[Y_{i+1, \rho_i, \zeta}] \\ &= \sum_{\rho_{i+1} \in \{0, 1\}^{r_{i+1}}} \Pr_{\zeta} [\text{TreeFinder}_{i+1}(\mathbf{x}, \rho_i \| \rho_{i+1}, \tilde{\mathbf{P}}_{\text{IP}}; \zeta) \neq \text{fail}] \quad (\text{by Definition 30.4.8}) \\ &\geq \sum_{\rho_{i+1} \in \{0, 1\}^{r_{i+1}}} \left(\prod_{j=i+2}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot (w(\rho_i \| \rho_{i+1}) - \kappa_{i+1}) \quad (\text{by strong induction hypothesis}) \\ &= 2^{r_{i+1}} \cdot \left(\prod_{j=i+2}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot \left(\frac{1}{2^{r_{i+1}}} \cdot \sum_{\rho_{i+1} \in \{0, 1\}^{r_{i+1}}} w(\rho_i \| \rho_{i+1}) - \kappa_{i+1} \cdot \frac{1}{2^{r_{i+1}}} \cdot \sum_{\rho_{i+1} \in \{0, 1\}^{r_{i+1}}} 1 \right) \\ &= 2^{r_{i+1}} \cdot \left(\prod_{j=i+2}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot (w(\rho_i) - \kappa_{i+1}). \end{aligned}$$

Next, we return to the probability that TreeFinder_i outputs a valid tree. Observe that

$$\Pr_{\zeta} [\text{TreeFinder}_i(\mathbf{x}, \rho_i, \tilde{\mathbf{P}}_{\text{IP}}; \zeta) \neq \text{fail}]$$

$$= \sum_{\ell=a_{i+1}}^{2^{r_{i+1}}} \Pr_{\zeta} [Z_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} = 1 \mid Y_{i+1, \rho_i, \zeta} = \ell] \cdot \Pr_{\zeta} [Y_{i+1, \rho_i, \zeta} = \ell].$$

This is because TreeFinder_i outputs a valid tree if and only if the first invocation of TreeFinder_{i+1} outputs a valid subtree (if not then TreeFinder_i outputs the error message `fail`) and $Y_{i+1, \rho_i, \zeta} \geq a_{i+1}$ (because if there are not enough good IP verifier messages then TreeFinder_i outputs the error message `fail`). The above expression thus considers all relevant values of $Y_{i+1, \rho_i, \zeta}$.

The rest of the proof consists of manipulations starting from the above expression:

$$\begin{aligned} &= \sum_{\ell=a_{i+1}}^{2^{r_{i+1}}} \frac{\ell}{2^{r_{i+1}}} \cdot \Pr_{\zeta} [Y_{i+1, \rho_i, \zeta} = \ell] \quad (\text{by Claim 30.4.9}) \\ &= \frac{1}{2^{r_{i+1}}} \cdot \sum_{\ell=a_{i+1}}^{2^{r_{i+1}}} \ell \cdot \Pr_{\zeta} [Y_{i+1, \rho_i, \zeta} = \ell] \\ &= \frac{1}{2^{r_{i+1}}} \cdot \left(\mathbb{E}_{\zeta} [Y_{i+1, \rho_i, \zeta}] - \sum_{\ell=0}^{a_{i+1}-1} \ell \cdot \Pr_{\zeta} [Y_{i+1, \rho_i, \zeta} = \ell] \right) \quad (\text{by definition of expectation}) \\ &\geq \frac{1}{2^{r_{i+1}}} \cdot \left(\mathbb{E}_{\zeta} [Y_{i+1, \rho_i, \zeta}] - (a_{i+1} - 1) \cdot \sum_{\ell=0}^{a_{i+1}-1} \Pr_{\zeta} [Y_{i+1, \rho_i, \zeta} = \ell] \right) \\ &\geq \frac{1}{2^{r_{i+1}}} \cdot \left(\mathbb{E}_{\zeta} [Y_{i+1, \rho_i, \zeta}] - (a_{i+1} - 1) \cdot \frac{2^{r_{i+1}} - \mathbb{E}_{\zeta} [Y_{i+1, \rho_i, \zeta}]}{2^{r_{i+1}} - a_{i+1} + 1} \right) \quad (\text{by Claim 30.4.10}) \\ &= \frac{1}{2^{r_{i+1}}} \cdot \frac{1}{2^{r_{i+1}} - a_{i+1} + 1} \cdot (\mathbb{E}_{\zeta} [Y_{i+1, \rho_i, \zeta}] \cdot (2^{r_{i+1}} - a_{i+1} + 1) - (a_{i+1} - 1) \cdot (2^{r_{i+1}} - \mathbb{E}_{\zeta} [Y_{i+1, \rho_i, \zeta}])) \\ &= \frac{1}{2^{r_{i+1}}} \cdot \frac{1}{2^{r_{i+1}} - a_{i+1} + 1} \cdot (\mathbb{E}_{\zeta} [Y_{i+1, \rho_i, \zeta}] \cdot 2^{r_{i+1}} - (a_{i+1} - 1) \cdot 2^{r_{i+1}}) \\ &= \frac{1}{2^{r_{i+1}} - a_{i+1} + 1} \cdot (\mathbb{E}_{\zeta} [Y_{i+1, \rho_i, \zeta}] - (a_{i+1} - 1)) \\ &\geq \frac{1}{2^{r_{i+1}} - a_{i+1} + 1} \cdot \left(2^{r_{i+1}} \cdot \left(\prod_{j=i+2}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot (w(\rho_i) - \kappa_{i+1}) - (a_{i+1} - 1) \right) \\ &\quad (\text{by Equation 30.6}) \\ &= \frac{2^{r_{i+1}}}{2^{r_{i+1}} - a_{i+1} + 1} \cdot \left(\left(\prod_{j=i+2}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot (w(\rho_i) - \kappa_{i+1}) - \frac{a_{i+1} - 1}{2^{r_{i+1}}} \right) \\ &= \left(\prod_{j=i+1}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot \left(w(\rho_i) - \kappa_{i+1} - \left(\prod_{j=i+2}^k \frac{2^{r_j} - a_j + 1}{2^{r_j}} \right) \cdot \left(\frac{a_{i+1} - 1}{2^{r_{i+1}}} \right) \right) \\ &= \left(\prod_{j=i+1}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot (w(\rho_i) - \kappa_i) \quad (\text{by Claim 30.4.13}). \end{aligned}$$

□

Claim 30.4.15. *The following holds:*

$$\Pr \left[T \text{ is an } (a_1, \dots, a_k)\text{-tree for } \mathbf{x} \mid \begin{array}{l} b \xleftarrow{((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})} \langle \tilde{\mathbf{P}}_{\text{IP}}, \mathbf{V}_{\text{IP}}(\mathbf{x}) \rangle_{\text{IP}} \\ T \leftarrow \text{TreeFinder}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}}) \end{array} \right]$$

$$\geq \prod_{i \in [k]} \left(1 - \frac{a_i - 1}{2^{r_i}}\right) - \Pr[b = 0].$$

Proof. We rewrite the probability statement in the claim:

$$\begin{aligned} & \Pr \left[T \text{ is an } (a_1, \dots, a_k)\text{-tree for } \mathbf{x} \mid \begin{array}{l} b \xleftarrow{((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})} \langle \tilde{\mathbf{P}}_{\text{IP}}, \mathbf{V}_{\text{IP}}(\mathbf{x}) \rangle_{\text{IP}} \\ T \leftarrow \text{TreeFinder}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}}) \end{array} \right] \\ &= \Pr \left[T \text{ is an } (a_1, \dots, a_k)\text{-tree for } \mathbf{x} \mid \begin{array}{l} b \xleftarrow{((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})} \langle \tilde{\mathbf{P}}_{\text{IP}}, \mathbf{V}_{\text{IP}}(\mathbf{x}) \rangle_{\text{IP}} \\ T \leftarrow \text{TreeFinder}_0(1, \mathbf{x}, \emptyset, \emptyset, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}}) \end{array} \right] \\ &= \Pr \left[\text{TreeFinder}_0(1, \mathbf{x}, \emptyset, \emptyset, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}}) \neq \text{fail} \mid b \xleftarrow{((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})} \langle \tilde{\mathbf{P}}_{\text{IP}}, \mathbf{V}_{\text{IP}}(\mathbf{x}) \rangle_{\text{IP}} \right] \\ &= \Pr \left[\text{TreeFinder}_0(0, \mathbf{x}, \emptyset, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\text{IP}}) \neq \text{fail} \right]. \end{aligned}$$

The first equality is by definition of TreeFinder (see Construction 30.4.3). The second equality is because TreeFinder_0 outputs the error message fail when it does not succeed. The third equality is due to Claim 30.4.5.

From the last expression, we conclude the claim as follows:

$$\begin{aligned} & \Pr \left[\text{TreeFinder}_0(0, \mathbf{x}, \emptyset, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\text{IP}}) \neq \text{fail} \right] \\ &= \Pr \left[\text{TreeFinder}_0(\mathbf{x}, \emptyset, \tilde{\mathbf{P}}_{\text{IP}}) \neq \text{fail} \right] \quad (\text{by Definition 30.4.6}) \\ &\geq \left(\prod_{j=1}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot (w(\emptyset) - \kappa_0) \quad (\text{by Claim 30.4.14 with } i=0) \\ &= \left(\prod_{j=1}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot (\Pr[b = 1] - \kappa_0) \quad (\text{by Definition 30.4.11}) \\ &= \left(\prod_{j=1}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot \left(\Pr[b = 1] - 1 + \prod_{i \in [k]} \left(1 - \frac{a_i - 1}{2^{r_i}}\right) \right) \quad (\text{by Definition 30.4.12}) \\ &= \left(\prod_{j=1}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot \left(\prod_{i \in [k]} \left(1 - \frac{a_i - 1}{2^{r_i}}\right) - \Pr[b = 0] \right) \quad (\text{since } \Pr[b = 0] + \Pr[b = 1] = 1) \\ &\geq \prod_{i \in [k]} \left(1 - \frac{a_i - 1}{2^{r_i}}\right) - \Pr[b = 0] \quad (\text{multiplicative term is at least 1}). \end{aligned}$$

□

Running time Here we focus on proving Claim 30.4.18.

Definition 30.4.16. For every $i \in \{0, 1, \dots, k\}$, IP verifier messages ρ_i , and TreeFinder_i randomness ζ , we denote by $B_{i, \rho_i, \zeta}$ the running time of $\text{TreeFinder}_i(\mathbf{x}, \rho_i, \tilde{\mathbf{P}}_{\text{IP}}; \zeta)$.

Claim 30.4.17. For every $i \in \{0, 1, \dots, k-1\}$ and IP verifier messages ρ_i , $\mathbb{E}_\zeta[B_{i, \rho_i, \zeta}] \leq \prod_{j=i+1}^k a_j \cdot (\tau_{\tilde{\mathbf{P}}_{\text{IP}}} + \text{poly}(\text{len}(\mathbf{x})))$; moreover, for every ζ , $B_{k, \rho_k, \zeta} = \text{poly}(\text{len}(\mathbf{x}))$.

Proof. The algorithm TreeFinder_k performs some $\text{poly}(\text{len}(\mathbf{x}))$ -time operations and does not invoke $\tilde{\mathbf{P}}_{\text{IP}}$, so $B_{k,\rho_k,\zeta} = \text{poly}(\text{len}(\mathbf{x}))$. Next, we prove that, for every $i \in \{0, 1, \dots, k-1\}$,

$$\mathbb{E}_{\zeta}[B_{i,\rho_i,\zeta}] \leq a_{i+1} \cdot \mathbb{E}_{\zeta}[B_{i+1,\rho_i\|\rho_{i+1}(\zeta),\zeta}] + \tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(i+1)} + \text{poly}(\text{len}(\mathbf{x})) \quad (30.7)$$

where $\tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(i+1)}$ is the running time of $\tilde{\mathbf{P}}_{\text{IP}}$ for round $i+1$. (Hence $\sum_{i \in [k]} \tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(i)} = \tau_{\tilde{\mathbf{P}}_{\text{IP}}}$.)

We explain how Equation 30.7 implies the claim. Define for notational convenience $a_{k+1} := 1$ and $\tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(k+1)} := 0$. We prove, by induction on $i \in \{0, 1, \dots, k\}$ that

$$\mathbb{E}_{\zeta}[B_{i,\rho_i,\zeta}] \leq \prod_{j=i+1}^{k+1} a_j \cdot \left(\sum_{j=i+1}^{k+1} \tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(j)} + \text{poly}(\text{len}(\mathbf{x})) \right).$$

This implies the claim since $\sum_{j=i+1}^k \tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(j)} \leq \tau_{\tilde{\mathbf{P}}_{\text{IP}}}$. For the base case, we already proved that $B_{k,\rho_k,\zeta} = \text{poly}(\text{len}(\mathbf{x}))$, which is at most $a_{k+1} \cdot (\tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(k+1)} + \text{poly}(\text{len}(\mathbf{x}))) = 1 \cdot (0 + \text{poly}(\text{len}(\mathbf{x})))$. For the inductive case, we assume the inequality for $i+1 \leq k$, and prove it for i :

$$\begin{aligned} & \mathbb{E}_{\zeta}[B_{i,\rho_i,\zeta}] \\ & \leq a_{i+1} \cdot \mathbb{E}_{\zeta}[B_{i+1,\rho_i\|\rho_{i+1}(\zeta),\zeta}] + \tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(i+1)} + \text{poly}(\text{len}(\mathbf{x})) \quad (\text{by Equation 30.7}) \\ & \leq a_{i+1} \cdot \prod_{j=i+2}^{k+1} a_j \cdot \left(\sum_{j=i+2}^{k+1} \tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(j)} + \text{poly}(\text{len}(\mathbf{x})) \right) + \tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(i+1)} + \text{poly}(\text{len}(\mathbf{x})) \quad (\text{by inductive hypothesis}) \\ & \leq \prod_{j=i+1}^{k+1} a_j \cdot \left(\sum_{j=i+1}^{k+1} \tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(j)} + \text{poly}(\text{len}(\mathbf{x})) \right). \end{aligned}$$

We are left with proving Equation 30.7. In fact, for every $\ell \in \{0, 1, \dots, 2^{r_{i+1}}\}$, we prove that

$$\mathbb{E}_{\zeta}[B_{i,\rho_i,\zeta} \mid Y_{i+1,\rho_i,\zeta} = \ell] \leq a_{i+1} \cdot \mathbb{E}_{\zeta}[B_{i+1,\rho_i\|\rho_{i+1}(\zeta),\zeta} \mid Y_{i+1,\rho_i,\zeta} = \ell] + \tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(i+1)} + \text{poly}(\text{len}(\mathbf{x})).$$

Since ℓ was arbitrary, we deduce (via total probability) that Equation 30.7 holds.

First, we discuss the case where $\ell = 0$, which is special. The algorithm TreeFinder_i computes a message from $\tilde{\mathbf{P}}_{\text{IP}}$ for round $i+1$ (see Item 2 in Construction 30.4.3) and then invokes TreeFinder_{i+1} . Since there are no $\rho_{i+1} \in \{0, 1\}^{r_i}$ that are good for (ρ_{i-1}, ζ) , the invocation of TreeFinder_{i+1} fails, so TreeFinder_i outputs the error message `fail` (see Item 5 in Construction 30.4.3). We conclude that

$$\mathbb{E}_{\zeta}[B_{i,\rho_i,\zeta} \mid Y_{i+1,\rho_i,\zeta} = 0] \leq 1 \cdot \mathbb{E}_{\zeta}[B_{i+1,\rho_i\|\rho_{i+1}(\zeta),\zeta} \mid Y_{i+1,\rho_i,\zeta} = 0] + \tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(i+1)} + \text{poly}(\text{len}(\mathbf{x})).$$

Next, we discuss the case where $\ell \in \{1, \dots, 2^{r_{i+1}}\}$. The algorithm TreeFinder_i computes a message from $\tilde{\mathbf{P}}_{\text{IP}}$ for round $i+1$ (see Item 2 in Construction 30.4.3) and then invokes TreeFinder_{i+1} several times. The first such invocation is special in that, if it fails, TreeFinder_i outputs the error message `fail` (see Item 5 in Construction 30.4.3). If instead the first such invocation succeeds (i.e., $Z_{i+1,\rho_i\|\rho_{i+1}(\zeta),\zeta} = 1$) then TreeFinder_i invokes TreeFinder_{i+1} in expectation A more times. Beyond that, there are $\text{poly}(\text{len}(\mathbf{x}))$ -time operations beyond recursions. Define the shorthand

$$p_0 := \Pr_{\zeta}[Z_{i+1,\rho_i\|\rho_{i+1}(\zeta),\zeta} = 0 \mid Y_{i+1,\rho_i,\zeta} = \ell]$$

and

$$p_1 := \Pr_{\zeta} [Z_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} = 1 \mid Y_{i+1, \rho_i, \zeta} = \ell] .$$

Then, we can write:

$$\mathbb{E}_{\zeta} [B_{i, \rho_i, \zeta} \mid Y_{i+1, \rho_i, \zeta} = \ell] = \mathbb{E}_{\zeta} [B_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} \mid Y_{i+1, \rho_i, \zeta} = \ell] + p_1 \cdot A + \tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(i+1)} + \text{poly}(\text{len}(\mathbf{x})) .$$

We are left to analyze A . We argue that A is the sum of two terms:

- The first term is

$$(a_{i+1} - 1) \cdot \mathbb{E}_{\zeta} [B_{i, \rho_i \| \rho_{i+1}(\zeta), \zeta} \mid Y_{i+1, \rho_i, \zeta} = \ell \wedge Z_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} = 1] .$$

Here we count the expected time spent due to invocations of TreeFinder_{i+1} that return valid subtrees. This number equals $\mathbb{E}_{\zeta} [B_{i, \rho_i \| \rho_{i+1}(\zeta), \zeta} \mid Y_{i+1, \rho_i, \zeta} = \ell \wedge Z_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} = 1]$ (the expected time due to TreeFinder_{i+1} provided TreeFinder_{i+1} succeeds) times the number A_1 of times TreeFinder_{i+1} succeeds. If $\ell < a_{i+1}$ then $A_1 \leq \ell \leq a_{i+1} - 1$ because there are ℓ good $\rho_i \in \{0, 1\}^{r_i}$ for (ρ_{i-1}, ζ) ; if instead $\ell \geq a_{i+1}$ then $A_1 \leq a_{i+1} - 1$ because TreeFinder_i only needs $a_{i+1} - 1$ subtrees (beyond the first subtree resulting from the first call of a_{i+1} , which here we assume has returned a valid subtree). Either way, $A_1 \leq a_{i+1} - 1$.

- The second term is

$$p_0 \cdot \frac{(a_{i+1} - 1) \cdot 2^{r_{i+1}}}{\ell} \cdot \mathbb{E}_{\zeta} [B_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} \mid Y_{i+1, \rho_i, \zeta} = \ell \wedge Z_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} = 0] .$$

Here we count the expected time spent due to invocations of TreeFinder_{i+1} that fail. This number equals $\mathbb{E}_{\zeta} [B_{i, \rho_i \| \rho_{i+1}(\zeta), \zeta} \mid Y_{i+1, \rho_i, \zeta} = \ell \wedge Z_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} = 0]$ (the expected number of invocations due to TreeFinder_{i+1} provided TreeFinder_{i+1} fails) times the number A_2 of times TreeFinder_{i+1} fails.

We argue that $A_2 \leq p_0 \cdot \frac{(a_{i+1} - 1) \cdot 2^{r_{i+1}}}{\ell}$. If $\ell < a_{i+1}$ then TreeFinder_i invokes TreeFinder_{i+1} for $2^{r_{i+1}} - 1$ times before realizing that there are not enough good $\rho_i \in \{0, 1\}^{r_i}$ for (ρ_{i-1}, ζ) ; out of these, $A_2 = (2^{r_{i+1}} - 1) - (\ell - 1) = 2^{r_{i+1}} - \ell$ fail. If instead $\ell \geq a_{i+1}$ then TreeFinder_i invokes TreeFinder_{i+1} for a number of times that equals $\frac{(a_{i+1} - 1) \cdot 2^{r_{i+1}}}{\ell}$, which is the expectation of the random variable X_{NHG} defined for the negative hypergeometric distribution (see Definition 30.1.7) from a set with $2^{r_{i+1}} - 1$ elements out of which $\ell - 1$ are colored blue, until $a_{i+1} - 1$ blue elements are drawn; out of these, $A_2 = \frac{(a_{i+1} - 1) \cdot 2^{r_{i+1}}}{\ell} - (a_{i+1} - 1)$ fail.

Finally, if $\ell < a_{i+1}$ then

$$\begin{aligned} 2^{r_{i+1}} - \ell &= \frac{2^{r_{i+1}} - \ell}{2^{r_{i+1}}} \cdot 2^{r_{i+1}} \\ &= p_0 \cdot 2^{r_{i+1}} \quad (\text{by Claim 30.4.9}) \\ &\leq p_0 \cdot \frac{(a_{i+1} - 1) \cdot 2^{r_{i+1}}}{\ell} . \end{aligned}$$

Similarly, if $\ell \geq a_{i+1}$ then

$$\frac{(a_{i+1} - 1) \cdot 2^{r_{i+1}}}{\ell} - (a_{i+1} - 1) = \frac{(a_{i+1} - 1) \cdot (2^{r_{i+1}} - \ell)}{\ell}$$

$$\begin{aligned}
&= \frac{2^{r_{i+1}} - \ell}{2^{r_{i+1}}} \cdot \frac{(a_{i+1} - 1) \cdot 2^{r_{i+1}}}{\ell} \\
&= p_0 \cdot \frac{(a_{i+1} - 1) \cdot 2^{r_{i+1}}}{\ell} \quad (\text{by Claim 30.4.9}).
\end{aligned}$$

Either way, we obtain the claimed bound on A_2 .

Recalling the fact that $p_1 = \frac{\ell}{2^{r_{i+1}}}$ (Claim 30.4.9), we deduce that $p_1 \cdot A$ is the sum of two terms:

- $(a_{i+1} - 1) \cdot p_1 \cdot \mathbb{E}_\zeta [B_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} \mid Y_{i+1, \rho_i, \zeta} = \ell \wedge Z_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} = 1];$
- $(a_{i+1} - 1) \cdot p_0 \cdot \mathbb{E}_\zeta [B_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} \mid Y_{i+1, \rho_i, \zeta} = \ell \wedge Z_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} = 0].$

The rest of the proof consists of manipulations on the above expression:

$$\begin{aligned}
&= \mathbb{E}_\zeta [B_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} \mid Y_{i+1, \rho_i, \zeta} = \ell] + (a_{i+1} - 1) \\
&\quad \cdot \left(p_1 \cdot \mathbb{E}_\zeta [B_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} \mid Y_{i+1, \rho_i, \zeta} = \ell \wedge Z_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} = 1] \right. \\
&\quad \left. + p_0 \cdot \mathbb{E}_\zeta [B_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} \mid Y_{i+1, \rho_i, \zeta} = \ell \wedge Z_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} = 0] \right) \\
&\quad + \tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(i+1)} + \text{poly}(\text{len}(\mathbf{x})) \\
&= \mathbb{E}_\zeta [B_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} \mid Y_{i+1, \rho_i, \zeta} = \ell] + (a_{i+1} - 1) \cdot \mathbb{E}_\zeta [B_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} \mid Y_{i+1, \rho_i, \zeta} = \ell] \\
&\quad + \tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(i+1)} + \text{poly}(\text{len}(\mathbf{x})) \\
&= a_{i+1} \cdot \mathbb{E}_\zeta [B_{i+1, \rho_i \| \rho_{i+1}(\zeta), \zeta} \mid Y_{i+1, \rho_i, \zeta} = \ell] + \tau_{\tilde{\mathbf{P}}_{\text{IP}}}^{(i+1)} + \text{poly}(\text{len}(\mathbf{x})).
\end{aligned}$$

□

Claim 30.4.18. TreeFinder runs in expected time $(\prod_{i \in [k]} a_i - 1) \cdot (\tau_{\tilde{\mathbf{P}}_{\text{IP}}} + \text{poly}(\text{len}(\mathbf{x})))$.

Proof. Claim 30.4.17 for $i = 0$ tells us that $\text{TreeFinder}_0(\mathbf{x}, \emptyset, \tilde{\mathbf{P}}_{\text{IP}})$ runs in expected time $(\prod_{i \in [k]} a_i) \cdot (\tau_{\tilde{\mathbf{P}}_{\text{IP}}} + \text{poly}(\text{len}(\mathbf{x})))$. By Definition 30.4.6, we have that $\text{TreeFinder}_0(\mathbf{x}, \emptyset, \tilde{\mathbf{P}}_{\text{IP}}) = \text{TreeFinder}_0(0, \mathbf{x}, \emptyset, \emptyset, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\text{IP}})$. Claim 30.4.5 tells us that $\text{TreeFinder}_0(0, \mathbf{x}, \emptyset, \emptyset, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\text{IP}})$ is the same random variable as invoking $\text{TreeFinder}_0(1, \mathbf{x}, \emptyset, \emptyset, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \tilde{\mathbf{P}}_{\text{IP}})$ where $((\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ is a fresh IP interaction transcript, which is used in each recursive call in the left-most branch of the recursion tree. By definition of TreeFinder (see Construction 30.4.3), we conclude that the expected time of TreeFinder equals the expected time of $\text{TreeFinder}_0(\mathbf{x}, \emptyset, \tilde{\mathbf{P}}_{\text{IP}})$ minus the time to generate one IP interaction transcript (i.e., minus $\tau_{\tilde{\mathbf{P}}_{\text{IP}}}$), which is $(\prod_{i \in [k]} a_i - 1) \cdot (\tau_{\tilde{\mathbf{P}}_{\text{IP}}} + \text{poly}(\text{len}(\mathbf{x})))$. □

30.5 State-restoration knowledge soundness for IPs

We prove that if a public-coin IP has special soundness then the IP has rewinding *state-restoration* knowledge soundness. The rewinding state-restoration knowledge soundness error and extraction time increase as the arity of the special soundness increases; neither depends on the failure probability of the state-restoration IP prover.

Theorem 30.5.1. *Let $\mathbf{IP} = (\mathbf{P}_{\mathbf{IP}}, \mathbf{V}_{\mathbf{IP}})$ be a public-coin IP with round complexity k . If \mathbf{IP} has special soundness with arity (a_1, \dots, a_k) with extraction time $\mathbf{et}_{\mathbf{IP}}^{\text{ss}}$ (see Definition 30.1.5) then \mathbf{IP} has rewinding state-restoration knowledge soundness error $\kappa_{\mathbf{IP}}^{\text{sr}}$ with extraction time $\mathbf{et}_{\mathbf{IP}}^{\text{sr}}$ (see Definition 13.2.5) such that*

$$\begin{aligned}\kappa_{\mathbf{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}}) &\leq (t+1) \cdot \left(1 - \prod_{i \in [k]} \left(1 - \frac{a_i - 1}{2^{r_i}}\right)\right) \leq (t+1) \cdot \sum_{i \in [k]} \frac{a_i - 1}{2^{r_i}}, \\ \mathbf{et}_{\mathbf{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}}, \tau_{\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}}) &\leq (t+1) \cdot \left(\prod_{i \in [k]} a_i - 1\right) \cdot \left(\tau_{\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}} + \text{poly}(\text{len}(\mathbf{x}))\right) + \mathbf{et}_{\mathbf{IP}}^{\text{ss}}(\mathbf{x}).\end{aligned}$$

In particular, if $r = r_1 = \dots = r_k$ and $a = a_1 = \dots = a_k$ then the knowledge soundness error and expected extraction time are

$$\begin{aligned}\kappa_{\mathbf{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}}) &\leq (t+1) \cdot \frac{k \cdot (a-1)}{2^r}, \\ \mathbf{et}_{\mathbf{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}}, \tau_{\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}}) &\leq (t+1) \cdot (a^k - 1) \cdot \left(\tau_{\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}} + \text{poly}(\text{len}(\mathbf{x}))\right) + \mathbf{et}_{\mathbf{IP}}^{\text{ss}}(\mathbf{x}).\end{aligned}$$

The proof of this theorem is similar to the proof of Theorem 30.4.1: rewind the given IP state-restoration prover $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$ several times with correlated inputs in order to obtain an (a_1, \dots, a_k) -tree T for \mathbf{x} ; this suffices because the knowledge extractor $\mathbf{E}_{\mathbf{IP}}^{\text{ss}}$ for special soundness can find a witness w from \mathbf{x} and T . However, there are notable differences that make the proof more technical. First, in rewinding state-restoration knowledge soundness (Definition 13.2.5), the IP state-restoration prover $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$ chooses the instance \mathbf{x} in the IP state-restoration game; in contrast, in rewinding knowledge soundness (Definition 13.1.5) the instance \mathbf{x} is fixed. Moreover, the IP state-restoration prover $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$ can make multiple attempts at convincing the IP verifier, not only with different instances but also with different IP partial transcripts; this complicates the rewinding strategy.

The lemma below provides a probabilistic procedure SRTreeFinder that outputs an (a_1, \dots, a_k) -tree T , which is analogous to Lemma 30.4.2. The procedure SRTreeFinder receives as input the output $(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ of the IP state-restoration game played by $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$, the list of move-response pairs tr^{sr} of $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$, and black-box access to $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$. The main challenge is that each time SRTreeFinder reruns $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$ we get a new output whose instance may not agree with those received as input, which would not contribute progress towards the desired (a_1, \dots, a_k) -tree for \mathbf{x} .³ Informally, SRTreeFinder identifies the first move (if any) corresponding to $(\mathbf{x}, \alpha_1, \sigma)$, and then reruns $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$ with freshly sampled answers (consistently answering other moves as needed) until enough accepting IP interaction transcripts with the same instance and IP prover message are found. Intuitively, this takes a multiplicative factor $(t+1)$ more reruns compared to Lemma 30.4.2, as $\tilde{\mathbf{P}}_{\mathbf{IP}}^{\text{sr}}$ makes at most t moves and can also output something corresponding to no move. Analyzing the success probability and expected running time of SRTreeFinder is technical, and the simpler analysis for Lemma 30.4.2 serves as a useful warmup.

³This is not an issue in the proof of Lemma 30.4.2 because there TreeFinder can rerun only the part of the IP prover $\tilde{\mathbf{P}}_{\mathbf{IP}}$ corresponding to the second round.

Lemma 30.5.2. *The algorithm SRTreeFinder in Construction 30.5.3 satisfies the following property. For every IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ that makes at most t moves,*

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge T \text{ is not an } (a_1, \dots, a_k)\text{-tree for } \mathbf{x} \\ \wedge \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 1 \end{array} \middle| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \\ \text{Game}_{\text{IP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \\ T \leftarrow \text{SRTreeFinder} \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \mathbf{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \end{array} \right] \\ \leq (t+1) \cdot \left(1 - \prod_{i \in [k]} \left(1 - \frac{a_i - 1}{2^{r_i}} \right) \right). \end{math>$$

Moreover, SRTreeFinder runs in expected time $(t+1) \cdot (\prod_{i \in [k]} a_i - 1) \cdot (\tau_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}} + \text{poly}(\text{len}(\mathbf{x})))$.

The algorithm SRTreeFinder invokes the base case SRTreeFinder_0 of a recursively defined helper function SRTreeFinder_j . The helper function SRTreeFinder_j receives black-box access to $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ and (lazily sampled) randomness $\mathbf{rnd} \in \mathcal{U}((r_i)_{i \in [k]})$, and its goal is to find a (a_{j+1}, \dots, a_k) -subtree of (accepting) transcripts for the instance \mathbf{x}' and partial transcript $((\alpha'_i)_{i=1}^j, (\rho'_i)_{i=1}^j)$, determined by the output $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]})$ of $\text{Game}_{\text{IP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$. Therefore, to ensure it obtains a tree of (accepting) transcripts for the correct instance, SRTreeFinder invokes SRTreeFinder_0 on a choice of randomness \mathbf{rnd} that is conditioned so that the output of $\text{Game}_{\text{IP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$ equals the input $(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ received by SRTreeFinder .

In more detail, SRTreeFinder_j invokes $\text{SRTreeFinder}_{j+1}$ multiple times, each time reprogramming the answer of a certain entry of \mathbf{rnd} to be a different IP verifier message ρ'_{j+1} , until TreeFinder_j obtains a_{j+1} subtrees for different IP verifier messages (or else SRTreeFinder_j outputs the error message `fail`). Then SRTreeFinder_j combines these subtrees to obtain the desired (a_{j+1}, \dots, a_k) -subtree. Several technical details complicate the strategy.

- *Early abort.* If the first invocation of $\text{SRTreeFinder}_{j+1}$ by SRTreeFinder_j results in a failure message then SRTreeFinder_j immediately halts and outputs a failure message. (Without making any further attempts.) This *early abort* condition ensures, as will be shown in the analysis, that the *expected* running time of SRTreeFinder_j is suitably bounded.
- *Consistency.* The first invocation of $\text{SRTreeFinder}_{j+1}$ by SRTreeFinder_j is distinguished from other invocations in another way: $\text{SRTreeFinder}_{j+1}$ receives as input a flag \mathbf{c} , which SRTreeFinder_j sets to 1 for the first invocation of $\text{SRTreeFinder}_{j+1}$ and sets to 0 for the other invocations (if there is no early abort).

The other invocations of $\text{SRTreeFinder}_{j+1}$ (i.e., with $\mathbf{c} = 0$) additionally receive as input the target instance \mathbf{x}' , target IP prover messages $(\alpha'_i)_{i=1}^{j+1}$, and target salt strings $(\sigma'_i)_{i=1}^{j+1}$, and the goal of each such invocation is to find another (a_{j+2}, \dots, a_k) -tree of accepting transcripts for the instance \mathbf{x} and partial transcript $((\alpha'_i)_{i=1}^{j+1}, (\rho'_i)_{i=1}^{j+1})$. This is achieved by re-programming the randomness \mathbf{rnd} : until it has found enough subtrees, SRTreeFinder_j samples without replacement a fresh answer ρ'_{j+1} , and invokes $\text{SRTreeFinder}_{j+1}$ with randomness $\mathbf{rnd}[\mu]$ where μ specifies to answer the query $x_{j+1} := (\mathbf{x}', (\alpha'_1, \dots, \alpha'_{j+1}), (\sigma'_1, \dots, \sigma'_{j+1}))$ with the randomness ρ'_{j+1} . (It can also be that SRTreeFinder_j tries all possible answers ρ'_{j+1} without finding enough subtrees, in which case it returns a failure message.)

The above summary is made precise in the construction below.

Construction 30.5.3. The algorithm **SRTreeFinder**, given as input a game output of the IP state-restoration game $(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$, a move-response trace tr^{sr} , and black-box access to an IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$, works as follows.

SRTreeFinder($\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \text{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$):

1. For every $i \in [k]$, if the move-response pair $((\mathbf{x}, (\alpha_1, \dots, \alpha_i), (\sigma_1, \dots, \sigma_i)), \rho_i)$ is not in tr^{sr} then append it to tr^{sr} .
2. Lazily sample an oracle $\text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]})$ consistent with the move-response trace tr^{sr} .
3. Compute $\text{out} \leftarrow \text{SRTreeFinder}_0(1, \emptyset, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$. (SRTreeFinder_j is defined below.)
4. If $\text{out} = \text{fail}$ then output fail .
5. Parse out as $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T)$.
6. Output T .

SRTreeFinder $_j(\mathbf{c}, \mathbf{x}, (\alpha_i)_{i=1}^j, (\sigma_i)_{i=1}^j, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$:

1. If $j = k$ (the base case):
 - a) Run the IP state-restoration game:
 $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}) \leftarrow \text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$.
 - b) Compute the decision bit of the IP verifier: $b := \mathbf{V}_{\text{IP}}(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]})$.
 - c) If $|\mathbf{x}'| > n \vee b = 0$, output fail .
 - d) Let T be the (trivial) tree that consists of a single (root) vertex with no labels.
 - e) If $\mathbf{c} = 1$ then output $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T)$.
 - f) If $\mathbf{c} = 0$:
 - i. If $\mathbf{x}' \neq \mathbf{x} \vee (\alpha'_i)_{i \in [k]} \neq (\alpha_i)_{i \in [k]} \vee (\sigma'_i)_{i \in [k]} \neq (\sigma_i)_{i \in [k]}$ then output fail .
 - ii. Else output $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T)$.
2. Compute $\text{out} \leftarrow \text{SRTreeFinder}_{j+1}(1, \emptyset, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$, with the following modification if $\mathbf{c} = 0$. (No modifications if $\mathbf{c} = 1$.) The first execution of the IP state-restoration game, at the bottom of the recursion, is $\text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$, and let $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]})$ be its output. If $\mathbf{x}' \neq \mathbf{x} \vee (\alpha'_i)_{i=1}^j \neq (\alpha_i)_{i=1}^j \vee (\sigma'_i)_{i=1}^j \neq (\sigma_i)_{i=1}^j$ then immediately terminate the execution of $\text{SRTreeFinder}_{j+1}$ and output fail .
3. If $\text{out} = \text{fail}$ then output fail .
4. Parse out as $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T_{j+1})$.
5. Set $x_{j+1} := (\mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1})$.
6. Set $S := \{0, 1\}^{r_{j+1}} \setminus \{\rho'_{j+1}\}$ and $B := \{(\rho'_{j+1}, T_{j+1})\}$. (Note that $\rho'_{j+1} = \text{rnd}(x_{j+1})$.)
7. While S is non-empty and $|B| < a_{j+1}$:
 - a) Sample a random string $\rho'_{j+1} \in S$, and remove ρ'_{j+1} from S .
 - b) Set $\mu_{j+1} := \{(x_{j+1}, \rho'_{j+1})\}$.
 - c) Compute $\text{out} \leftarrow \text{SRTreeFinder}_{j+1}(0, \mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}[\mu_{j+1}])$.
 - d) If $\text{out} \neq \text{fail}$ then parse out as $(\mathbf{x}'', (\alpha''_i)_{i \in [k]}, (\sigma''_i)_{i \in [k]}, (\rho''_i)_{i \in [k]}, T_{j+1})$ and add (ρ'_{j+1}, T_{j+1}) to B .
8. If $|B| < a_{j+1}$ then output fail .
9. Let T be the tree obtained as follows: (i) the root is labeled by α'_{j+1} ; (ii) for every $(\rho'_{j+1}, T_{\rho'_{j+1}}) \in B$, connect the root to the root of $T_{\rho'_{j+1}}$ and label the edge by ρ'_{j+1} . Note that the root of T has a_{j+1} children.
10. Output $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T)$.

Proof of Lemma 30.5.2. We prove a lower bound on the probability of the event negation, namely, we prove that the probability that T is an (a_1, \dots, a_k) -tree for \mathbf{x} or $|\mathbf{x}| > n$ or $\mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 0$ is at least $1 - (t+1) \cdot \left(1 - \prod_{i \in [k]} \left(1 - \frac{a_i-1}{2^{r_i}}\right)\right)$.

If $|\mathbf{x}| > n$ or $\mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 0$ then SRTreeFinder outputs the error message `fail` (in particular, T is not a valid tree for \mathbf{x}), so the above two conditions are disjoint. We deduce that

$$\begin{aligned} & \Pr \left[\begin{array}{l} T \text{ is an } (a_1, \dots, a_k)\text{-tree for } \mathbf{x} \\ \vee (|\mathbf{x}| > n \vee \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 0) \end{array} \right] \\ &= \Pr[T \text{ is an } (a_1, \dots, a_k)\text{-tree for } \mathbf{x}] + \Pr[|\mathbf{x}| > n \vee \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 0] \\ &\geq \left(\left(1 - (t+1) \cdot \left(1 - \prod_{i \in [k]} \left(1 - \frac{a_i-1}{2^{r_i}}\right)\right)\right) - \Pr[|\mathbf{x}| > n \vee \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 0] \right) \\ &\quad + \Pr[|\mathbf{x}| > n \vee \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 0] \\ &= 1 - (t+1) \cdot \left(1 - \prod_{i \in [k]} \left(1 - \frac{a_i-1}{2^{r_i}}\right)\right). \end{aligned}$$

In Claim 30.5.17 we prove the inequality above. Moreover in Claim 30.5.29 we prove that SRTreeFinder runs in expected time $(t+1) \cdot (\prod_{i \in [k]} a_i - 1) \cdot (\tau_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}} + \text{poly}(\text{len}(\mathbf{x})))$. Establishing both claims is technical, as it involves a number of intermediate claims and computations; the details are in Section 30.5.1. \square

Construction 30.5.4. Let $\mathbf{E}_{\text{IP}}^{\text{ss}}$ be the special soundness extractor for IP , and let SRTreeFinder be the algorithm in Construction 30.5.3. The rewinding knowledge extractor $\mathbf{E}_{\text{IP}}^{\text{sr}}$ receives as input the IP state-restoration game output $(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]})$, the move-response trace \mathbf{tr}^{sr} , and black-box access to an IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$, and works as follows.

$\mathbf{E}_{\text{IP}}^{\text{sr}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \mathbf{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$:

1. Run $T \leftarrow \text{SRTreeFinder}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \mathbf{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$; if SRTreeFinder aborts then abort.
2. Run $\mathbf{w} \leftarrow \mathbf{E}_{\text{IP}}^{\text{ss}}(\mathbf{x}, T)$.
3. Output \mathbf{w} .

Proof of Theorem 30.5.1. The knowledge extractor $\mathbf{E}_{\text{IP}}^{\text{sr}}$ succeeds if the algorithm SRTreeFinder succeeds, so the knowledge soundness error of $\mathbf{E}_{\text{IP}}^{\text{sr}}$ is at most the failure probability of SRTreeFinder . The latter, by Lemma 30.5.2, is at most $(t+1) \cdot \left(1 - \prod_{i \in [k]} \left(1 - \frac{a_i-1}{2^{r_i}}\right)\right)$. Hence the knowledge soundness error of $\mathbf{E}_{\text{IP}}^{\text{sr}}$ is as claimed in Theorem 30.5.1.

Moreover, the expected running time of $\mathbf{E}_{\text{IP}}^{\text{sr}}$ is the expected running time of SRTreeFinder plus the running time of $\mathbf{E}_{\text{IP}}^{\text{ss}}$. By Lemma 30.5.2, the expected running time of SRTreeFinder is at most $(t+1) \cdot (\prod_{i \in [k]} a_i - 1) \cdot (\tau_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}} + \text{poly}(\text{len}(\mathbf{x})))$. By definition, the running time of $\mathbf{E}_{\text{IP}}^{\text{ss}}$ is $\text{et}_{\text{IP}}^{\text{ss}}(\mathbf{x})$. We conclude that the expected running time of $\mathbf{E}_{\text{IP}}^{\text{sr}}$ is as claimed in Theorem 30.5.1. \square

30.5.1 Technical details

In the proof of Lemma 30.5.2 we relied on two main claims: Claim 30.5.17 (bound on error probability) and Claim 30.5.29 (bound on expected running time). The goal of this section

is to prove these claims; in each case, this requires introducing more notation and proving intermediate claims.

Below, for every $j \in \{0, \dots, k\}$, we let \mathcal{Z}_j denote the set of random strings for SRTreeFinder_j .

Error probability Here we focus on proving Claim 30.5.17. The algorithm SRTreeFinder outputs an (a_1, \dots, a_k) -tree T for \mathbf{x} if and only if the tree T output by SRTreeFinder_0 (invoked in Item 3 of SRTreeFinder) is an (a_1, \dots, a_k) -tree for \mathbf{x} . The analysis focuses on upper bounding the error probability of SRTreeFinder_0 ; more generally, the analysis focuses on SRTreeFinder_j for $j \in \{0, 1, \dots, k\}$.

The algorithm SRTreeFinder_j , when invoked with flag $\mathbf{c} = 1$, receives the IP state-restoration prover $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ and (lazily sampled) random oracle, and no other inputs (they are set to \emptyset instead). We introduce a short-hand notation for this case.

Definition 30.5.5. For every $j \in \{0, 1, \dots, k\}$ and $\mathbf{rnd} \in \mathcal{U}((r_i)_{i \in [k]})$,

$$\text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \mathbf{rnd}) := \text{SRTreeFinder}_j(1, \emptyset, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \mathbf{rnd}).$$

We define a set for quantifying over all possible moves for a given round in the IP state-restoration game.

Definition 30.5.6. For every round index $j \in [k]$, define T_j to be the set all of all triples $(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j)$ where \mathbf{x} is an instance of size at most n , $\boldsymbol{\alpha}_j$ is a list of IP prover messages $(\alpha_1, \dots, \alpha_j)$, and $\boldsymbol{\sigma}_j$ is a list of salt strings $(\sigma_1, \dots, \sigma_j)$:

$$T_j := \{(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) : |\mathbf{x}| \leq n, \boldsymbol{\alpha}_j \in \{0, 1\}^{p_{v_1} + \dots + p_{v_j}}, \boldsymbol{\sigma}_j \in \{0, 1\}^{j \cdot s}\}.$$

The three claims below directly follow from the definitions of SRTreeFinder and SRTreeFinder_j (see Construction 30.5.3).

Claim 30.5.7. The following two distributions are identical:

$$\left\{ T \mid \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \text{Game}_{\text{IP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \\ T \leftarrow \text{SRTreeFinder}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \text{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \end{array} \right\}$$

and

$$\left\{ \mathbf{out}[5] \mid \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ \mathbf{out} \leftarrow \text{SRTreeFinder}_0(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \mathbf{rnd}) \end{array} \right\},$$

where, above, if $\mathbf{out} = \text{fail}$ then $\mathbf{out}[5] := \text{fail}$ and, instead, if $\mathbf{out} = (\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T)$ then $\mathbf{out}[5] := T$ (the 5-th entry of \mathbf{out}).

Claim 30.5.8. For every round index $j \in [k]$, $\mathbf{rnd} \in \mathcal{U}((r_i)_{i \in [k]})$, and SRTreeFinder_j randomness $\zeta \in \mathcal{Z}_j$, define $\mathbf{out} := \text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \mathbf{rnd}; \zeta)$. If $\mathbf{out} \neq \text{fail}$ then: (i) \mathbf{out} can be parsed as $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T)$ where $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]})$ is the output of $\text{Game}_{\text{IP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$; and (ii) $\mathbf{out} = \text{SRTreeFinder}_j(0, \mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \mathbf{rnd}; \zeta)$.

Claim 30.5.9. For every round index $j \in [k]$, triple $(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \in T_j$, $\mathbf{rnd} \in \mathcal{U}((r_i)_{i \in [k]})$, and SRTreeFinder_j randomness $\zeta \in \mathcal{Z}_j$, define $\mathbf{out} := \text{SRTreeFinder}_j(0, \mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \mathbf{rnd}; \zeta)$. If $\mathbf{out} \neq \text{fail}$ then \mathbf{out} can be parsed as $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T)$ and, moreover, it holds that:

- $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]})$ is the output of $\text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$; and
- $\mathbf{x}' = \mathbf{x} \wedge (\alpha'_i)_{i=1}^j = \boldsymbol{\alpha}_j \wedge (\sigma'_i)_{i=1}^j = \boldsymbol{\sigma}_j$.

The output of SRTreeFinder does not depend on ζ (only its running time is affected by the internal randomness ζ). We capture this property in the following claim.

Claim 30.5.10. *For every round index $j \in \{0, 1, \dots, k\}$, $\text{rnd} \in \mathcal{U}((r_i)_{i \in [k]})$, and internal randomness $\zeta, \zeta' \in \mathcal{Z}_j$,*

$$\text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}; \zeta) = \text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}; \zeta') .$$

Moreover, for every $(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \in T_j$,

$$\text{SRTreeFinder}_j(0, \mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}; \zeta) = \text{SRTreeFinder}_j(0, \mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}; \zeta') .$$

Henceforth, we omit the internal randomness if we are only interested in the output of SRTreeFinder_j .

The analysis of the success probability of SRTreeFinder_j is in terms of some random variables. Note that, these random variables involve the output of SRTreeFinder_j , so, by Claim 30.5.10, do not depend on the internal randomness of SRTreeFinder_j .

Definition 30.5.11. *For every $j \in \{0, 1, \dots, k\}$ and $\text{rnd} \in \mathcal{U}((r_i)_{i \in [k]})$, define $Z_{j, \text{rnd}}$ to be the indicator for the event*

$$\text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}) \neq \text{fail} .$$

Note that if $j = k$ then $Z_{k, \text{rnd}}$ corresponds to the following event:

$$\left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 1 \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \end{array} \right] .$$

Definition 30.5.12. *For every round index $j \in [k]$, triple $(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \in T_j$, and $\text{rnd} \in \mathcal{U}((r_i)_{i \in [k]})$, define $Y_{j, \text{rnd}, \mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j}$ to be the number of $\rho_j \in \{0, 1\}^{r_j}$ such that, letting $\mu_j := \{((\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j), \rho_j)\}$,*

$$\text{SRTreeFinder}_j(0, \mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}[\mu_j]) \neq \text{fail} .$$

Note that if $Z_{j, \text{rnd}} = 1$ then $Y_{j, \text{rnd}, \mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j} > 0$ where $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T)$ is the output of $\text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$.

Claim 30.5.13. *For every round index $j \in [k]$, $(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \in T_j$, and $\ell \in \{0, 1, \dots, 2^{r_j}\}$,*

$$\Pr \left[\begin{array}{l} Z_{j, \text{rnd}} = 1 \\ \wedge (\mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j) = (\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \\ \text{conditioned on} \\ Y_{j, \text{rnd}, \mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j} = \ell \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ \text{out} \leftarrow \text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}) \\ \text{parse out as } (\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T) \end{array} \right] = \frac{\ell}{2^{r_j}} .$$

Proof. First suppose that $\ell = 0$: if $Y_{j,\text{rnd},\mathbf{x}',(\alpha'_i)_{i=1}^j,(\sigma'_i)_{i=1}^j} = 0$ then $Z_{j,\text{rnd}} = 0$, so the probability equals 0 in this case. Next, if $\ell \in \{1, \dots, 2^{r_j}\}$, we argue as follows:

$$\begin{aligned} & \Pr \left[\begin{array}{l} Z_{j,\text{rnd}} = 1 \\ \wedge (\mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j) = (\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \\ \text{conditioned on} \\ Y_{j,\text{rnd},\mathbf{x}',(\alpha'_i)_{i=1}^j,(\sigma'_i)_{i=1}^j} = \ell \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ \text{out} \leftarrow \text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}) \\ \text{parse out as } (\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T) \end{array} \right] \\ &= \Pr \left[\begin{array}{l} Z_{j,\text{rnd}} = 1 \\ \wedge (\mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j) = (\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \\ \text{conditioned on} \\ Y_{j,\text{rnd},\mathbf{x},\boldsymbol{\alpha}_j,\boldsymbol{\sigma}_j} = \ell \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ \text{out} \leftarrow \text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}) \\ \text{parse out as } (\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T) \end{array} \right] \\ &= \Pr \left[\begin{array}{l} Z_{j,\text{rnd}} = 1 \\ \text{conditioned on} \\ Y_{j,\text{rnd},\mathbf{x},\boldsymbol{\alpha}_j,\boldsymbol{\sigma}_j} = \ell \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ \text{out} \leftarrow \text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}) \\ \text{parse out as } (\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T) \end{array} \right] \\ &= \frac{\ell}{2^{r_j}}. \end{aligned}$$

Above, the second equality holds because the condition $Y_{j,\text{rnd},\mathbf{x},\boldsymbol{\alpha}_j,\boldsymbol{\sigma}_j} = \ell > 0$ implies that $(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j)$ is part of the output of the state-restoration game and thus $(\mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j) = (\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j)$. \square

Claim 30.5.14. *For every round index $j \in [k]$,*

$$\sum_{(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \in T_j} \Pr \left[Y_{j,\text{rnd},\mathbf{x},\boldsymbol{\alpha}_j,\boldsymbol{\sigma}_j} > 0 \mid \text{rnd} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \right] \leq t + 1.$$

Moreover,

$$\sum_{(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \in T_j} \Pr \left[\begin{array}{l} Y_{j,\text{rnd},\mathbf{x},\boldsymbol{\alpha}_j,\boldsymbol{\sigma}_j} > 0 \\ \wedge (\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \\ \neq (\mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j) \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ \text{out} \leftarrow \text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}) \\ \text{parse out as} \\ (\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T) \end{array} \right] \leq t.$$

Proof. First we prove the first part of the claim. Define $S(j, \text{rnd})$ to be the set of unique moves (i.e., no duplicates) performed by $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ in the IP state-restoration game $\text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$; the set $S(j, \text{rnd})$ also includes the move $(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j)$ contained in the output of the game in case $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ did not make this move. Note that $|S(j, \text{rnd})| \leq t + 1$. Moreover, for every $(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \notin S(j, \text{rnd})$ and $\rho_j \in \{0, 1\}^{r_j}$, programming rnd with $\mu_j = \{((\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j), \rho_j)\}$ does not change the output of the IP state-restoration game, i.e., the output of $\text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}[\mu_j], \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$ equals that of $\text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$. We argue that if $(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \notin S(j, \text{rnd})$ then $Y_{j,\text{rnd},\mathbf{x},\boldsymbol{\alpha}_j,\boldsymbol{\sigma}_j} = 0$. Indeed, suppose by way of contradiction that $Y_{j,\text{rnd},\mathbf{x},\boldsymbol{\alpha}_j,\boldsymbol{\sigma}_j} > 0$, which means that there exists an IP verifier message $\rho_j \in \{0, 1\}^{r_j}$ such that, for $\mu_j := \{((\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j), \rho_j)\}$,

$$\text{SRTreeFinder}_j(0, \mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}[\mu_j]) \neq \text{fail},$$

which in turn, by Claim 30.5.9, means that the output of $\text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}[\mu_j], \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$ contains $(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j)$; since $(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \notin S(j, \text{rnd})$, the output of $\text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}[\mu_j], \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$ equals that of $\text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$, which lets us conclude that $(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \in S(j, \text{rnd})$, a contradiction.

We conclude the proof as follows:

$$\begin{aligned}
& \sum_{(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \in T_j} \Pr [Y_{j, \text{rnd}, \mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j} > 0 \mid \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]})] \\
&= \sum_{(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \in T_j} \sum_{\text{rnd}' \in \mathcal{U}((r_i)_{i \in [k]})} \Pr [Y_{j, \text{rnd}', \mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j} > 0] \cdot \Pr [\text{rnd} = \text{rnd}' \mid \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]})] \\
&= \sum_{\text{rnd}' \in \mathcal{U}((r_i)_{i \in [k]})} \Pr [\text{rnd} = \text{rnd}' \mid \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]})] \sum_{(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \in T_j} \Pr [Y_{j, \text{rnd}', \mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j} > 0] \\
&= \sum_{\text{rnd}' \in \mathcal{U}((r_i)_{i \in [k]})} \Pr [\text{rnd} = \text{rnd}' \mid \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]})] \sum_{(\mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j) \in S(j, \text{rnd}')} \Pr [Y_{j, \text{rnd}', \mathbf{x}, \boldsymbol{\alpha}_j, \boldsymbol{\sigma}_j} > 0] \\
&\leq \sum_{\text{rnd}' \in \mathcal{U}((r_i)_{i \in [k]})} \Pr [\text{rnd} = \text{rnd}' \mid \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]})] \cdot (t + 1) \\
&= (t + 1) \cdot \sum_{\text{rnd}' \in \mathcal{U}((r_i)_{i \in [k]})} \Pr [\text{rnd} = \text{rnd}' \mid \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]})] \\
&= (t + 1).
\end{aligned}$$

The proof of the second part of the claim is analogous. The only difference is that the additional condition enables excludes the final output of the IP state-restoration game. Hence it suffices to define the set $S(j, \text{rnd})$ to be the set of unique moves performed by $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ in the IP state-restoration game $\text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$, without the move contained in the output of the game. Therefore, in this case we get $|S(j, \text{rnd})| \leq t$, and the bound follows. \square

Claim 30.5.15. *For every $j \in \{0, \dots, k - 1\}$,*

$$\begin{aligned}
& \Pr [Z_{j, \text{rnd}} = 1 \mid \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}), \text{out} \leftarrow \text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})] \\
&\geq \frac{2^{r_{j+1}}}{2^{r_{j+1}} - a_{j+1} + 1} \\
&\quad \cdot \left(\Pr [Z_{j+1, \text{rnd}} = 1 \mid \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}), \text{out} \leftarrow \text{SRTreeFinder}_{j+1}(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})] - (t + 1) \cdot \frac{a_{j+1} - 1}{2^{r_{j+1}}} \right).
\end{aligned}$$

Proof. For every $\text{rnd} \in \mathcal{U}((r_i)_{i \in [k]})$, $\text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$ does not fail if and only if the following two conditions hold.

- The invocation $\text{SRTreeFinder}_{j+1}(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$ does not fail (see Item 3 in Construction 30.5.3), in which case $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T_{j+1})$ denotes its output (see Item 4 in Construction 30.5.3). Define $x_{j+1} := (\mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1})$.
- There are at least $a_{j+1} - 1$ choices of $\rho_{j+1} \in \{0, 1\}^{r_{j+1}} \setminus \{\rho'_{j+1}\}$ such that, for $\mu_{j+1} := \{(x_{j+1}, \rho_{j+1})\}$,

$$\text{SRTreeFinder}_{j+1}(0, \mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}[\mu_{j+1}]) \neq \text{fail}.$$

Indeed, the while loop (see Item 7 in Construction 30.5.3) succeeds precisely when $a_{j+1} - 1$ choices of $\rho_{j+1} \in \{0, 1\}^{r_{j+1}}$ satisfying the above condition are found.

Note that $\rho'_{j+1} := \text{rnd}_{j+1}(x_{j+1})$ counts as the a_{j+1} -th choice because $\text{SRTreeFinder}_{j+1}(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$ does not fail (see first point above) and, moreover,

$$\begin{aligned} & \text{SRTreeFinder}_{j+1}(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}) \\ &= \text{SRTreeFinder}_{j+1}(0, \mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}) \quad (\text{by Claim 30.5.8}) \\ &= \text{SRTreeFinder}_{j+1}(0, \mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}[\mu_{j+1}]) \quad (\text{since } \rho'_{j+1} = \text{rnd}_{j+1}(x_{j+1})). \end{aligned}$$

The above considerations, along with Definitions 30.5.11 and 30.5.12, directly imply that

$$\begin{aligned} & \Pr \left[Z_{j, \text{rnd}} = 1 \mid \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ \text{out} \leftarrow \text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}) \end{array} \right] \\ &= \Pr \left[\begin{array}{l} Z_{j+1, \text{rnd}} = 1 \\ \wedge Y_{j+1, \text{rnd}, \mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1}} \geq a_{j+1} \end{array} \mid \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ \text{out} \leftarrow \text{SRTreeFinder}_{j+1}(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}) \\ \text{parse out as } (\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T) \end{array} \right]. \end{aligned}$$

For notational simplicity, in the rest of this proof we fix the following experiment for all probability statements whose experiment is implicit:

$$\left[\begin{array}{l} \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ \text{out} \leftarrow \text{SRTreeFinder}_{j+1}(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}) \\ \text{parse out as } (\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}, T) \end{array} \right].$$

By total probability, we can write

$$\begin{aligned} & \Pr \left[Z_{j, \text{rnd}} = 1 \mid \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ \text{out} \leftarrow \text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}) \end{array} \right] \\ &= \sum_{(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \Pr \left[\begin{array}{l} Z_{j+1, \text{rnd}} = 1 \\ \wedge (\mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j) = (\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \\ \wedge Y_{j+1, \text{rnd}, \mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1}} \geq a_{j+1} \end{array} \right] \\ &= \sum_{(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \sum_{\ell=a_{j+1}}^{2^{r_{j+1}}} \Pr \left[\begin{array}{l} Z_{j+1, \text{rnd}} = 1 \\ \wedge (\mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j) = (\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \\ \wedge Y_{j+1, \text{rnd}, \mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1}} = \ell \end{array} \right] \\ &= \sum_{(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \sum_{\ell=a_{j+1}}^{2^{r_{j+1}}} \Pr \left[\begin{array}{l} Z_{j+1, \text{rnd}} = 1 \\ \wedge (\mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j) = (\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \\ \wedge Y_{j+1, \text{rnd}, \mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}} = \ell \end{array} \right] \\ &= \sum_{(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \sum_{\ell=a_{j+1}}^{2^{r_{j+1}}} \frac{\ell}{2^{r_{j+1}}} \cdot \Pr [Y_{j+1, \text{rnd}, \mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}} = \ell] \quad (\text{by Claim 30.5.13}). \end{aligned}$$

Next, since $0 \leq \ell \leq 2^{r_{j+1}}$,

$$\begin{aligned} \frac{\ell}{2^{r_{j+1}}} &= 1 - \left(1 - \frac{\ell}{2^{r_{j+1}}} \right) \\ &\geq 1 - \frac{2^{r_{j+1}}}{2^{r_{j+1}} - a_{j+1} + 1} \cdot \left(1 - \frac{\ell}{2^{r_{j+1}}} \right) \end{aligned}$$

$$= \frac{2^{r_{j+1}}}{2^{r_{j+1}} - a_{j+1} + 1} \cdot \left(\frac{\ell}{2^{r_{j+1}}} - \frac{a_{j+1} - 1}{2^{r_{j+1}}} \right).$$

This lets us continue the derivation by writing:

$$\begin{aligned} &\geq \sum_{(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \sum_{\ell=a_{j+1}}^{2^{r_{j+1}}} \left(\frac{2^{r_{j+1}}}{2^{r_{j+1}} - a_{j+1} + 1} \cdot \left(\frac{\ell}{2^{r_{j+1}}} - \frac{a_{j+1} - 1}{2^{r_{j+1}}} \right) \right) \\ &\quad \cdot \Pr [Y_{j+1, \text{rnd}, \mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}} = \ell] \\ &\geq \sum_{(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \sum_{\ell=1}^{2^{r_{j+1}}} \left(\frac{2^{r_{j+1}}}{2^{r_{j+1}} - a_{j+1} + 1} \cdot \left(\frac{\ell}{2^{r_{j+1}}} - \frac{a_{j+1} - 1}{2^{r_{j+1}}} \right) \right) \\ &\quad \cdot \Pr [Y_{j+1, \text{rnd}, \mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}} = \ell] \\ &= \frac{2^{r_{j+1}}}{2^{r_{j+1}} - a_{j+1} + 1} \sum_{(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \sum_{\ell=1}^{2^{r_{j+1}}} \left(\frac{\ell}{2^{r_{j+1}}} - \frac{a_{j+1} - 1}{2^{r_{j+1}}} \right) \\ &\quad \cdot \Pr [Y_{j+1, \text{rnd}, \mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}} = \ell] \\ &= \frac{2^{r_{j+1}}}{2^{r_{j+1}} - a_{j+1} + 1} \cdot \left(\sum_{(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \sum_{\ell=1}^{2^{r_{j+1}}} \frac{\ell}{2^{r_{j+1}}} \cdot \Pr [Y_{j+1, \text{rnd}, \mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}} = \ell] \right. \\ &\quad \left. - \sum_{(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \sum_{\ell=1}^{2^{r_{j+1}}} \frac{a_{j+1} - 1}{2^{r_{j+1}}} \cdot \Pr [Y_{j+1, \text{rnd}, \mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}} = \ell] \right) \\ &= \frac{2^{r_{j+1}}}{2^{r_{j+1}} - a_{j+1} + 1} \cdot \left(\sum_{(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \sum_{\ell=1}^{2^{r_{j+1}}} \frac{\ell}{2^{r_{j+1}}} \cdot \Pr [Y_{j+1, \text{rnd}, \mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}} = \ell] \right. \\ &\quad \left. - \sum_{(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \Pr [Y_{j+1, \text{rnd}, \mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}} > 0] \cdot \frac{a_{j+1} - 1}{2^{r_{j+1}}} \right) \\ &\geq \frac{2^{r_{j+1}}}{2^{r_{j+1}} - a_{j+1} + 1} \cdot \left(\Pr [Z_{j+1, \text{rnd}} = 1] - (t+1) \cdot \frac{a_{j+1} - 1}{2^{r_{j+1}}} \right). \end{aligned}$$

The last inequality follows from two observations:

- By Claim 30.5.14, $\sum_{(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \Pr [Y_{j+1, \text{rnd}, \mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}} > 0] \leq t+1$.
- By total probability:

$$\begin{aligned} &\Pr [Z_{j+1, \text{rnd}} = 1] \\ &= \sum_{(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \sum_{\ell=1}^{2^{r_{j+1}}} \Pr \left[\begin{array}{l} Z_{j+1, \text{rnd}} = 1 \\ \wedge (\mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j) = (\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \\ \wedge Y_{j+1, \text{rnd}, \mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1}} = \ell \end{array} \right] \\ &= \sum_{(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \sum_{\ell=1}^{2^{r_{j+1}}} \Pr \left[\begin{array}{l} Z_{j+1, \text{rnd}} = 1 \\ \wedge (\mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j) = (\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \\ \wedge Y_{j+1, \text{rnd}, \mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}} = \ell \end{array} \right] \end{aligned}$$

$$= \sum_{(\mathbf{x}, \alpha_{j+1}, \sigma_{j+1}) \in T_{j+1}} \sum_{\ell=1}^{2^{r_{j+1}}} \frac{\ell}{2^{r_{j+1}}} \cdot \Pr[Y_{j+1, \mathbf{rnd}, \mathbf{x}, \alpha_{j+1}, \sigma_{j+1}} = \ell] \quad (\text{by Claim 30.5.13}).$$

□

Claim 30.5.16. *For every $i \in \{0, \dots, k\}$, define*

$$w_i := \Pr[Z_{i, \mathbf{rnd}} = 1 \mid \mathbf{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]})]. \quad (30.8)$$

Then, for every $i \in \{0, \dots, k-1\}$,

$$w_i \geq \left(\prod_{j=i+1}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot (w_k - (t+1) \cdot \kappa_i).$$

Proof. The proof is by (reverse) induction on i . First consider the base case $i = k-1$:

$$\begin{aligned} w_{k-1} &\geq \frac{2^{r_k}}{2^{r_k} - a_k + 1} \cdot \left(w_k - (t+1) \cdot \frac{a_k - 1}{2^{r_k}} \right) \quad (\text{by Claim 30.5.15}) \\ &= \frac{2^{r_k}}{2^{r_k} - a_k + 1} \cdot (w_k - (t+1) \cdot \kappa_{k-1}) \quad (\text{by Definition 30.4.12 with } i = k-1) \\ &= \left(\prod_{j=(k-1)+1}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot (w_k - (t+1) \cdot \kappa_{k-1}). \end{aligned}$$

Next consider the recursive case, where we assume the inequality for $i+1 \in \{1, \dots, k-1\}$ and prove it for i . We write:

$$\begin{aligned} w_i &\geq \frac{2^{r_{i+1}}}{2^{r_{i+1}} - a_{i+1} + 1} \cdot \left(w_{i+1} - (t+1) \cdot \frac{a_{i+1} - 1}{2^{r_{i+1}}} \right) \quad (\text{by Claim 30.5.15}) \\ &\geq \frac{2^{r_{i+1}}}{2^{r_{i+1}} - a_{i+1} + 1} \cdot \left(\left(\prod_{j=i+2}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot (w_k - (t+1) \cdot \kappa_{i+1}) - (t+1) \cdot \frac{a_{i+1} - 1}{2^{r_{i+1}}} \right) \\ &\quad (\text{by induction}) \\ &= \left(\prod_{j=i+1}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot \left(w_k - (t+1) \cdot \left(\kappa_{i+1} + \left(\prod_{j=i+2}^k \frac{2^{r_j} - a_j + 1}{2^{r_j}} \right) \cdot \frac{a_{i+1} - 1}{2^{r_{i+1}}} \right) \right) \\ &= \left(\prod_{j=i+1}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot (w_k - (t+1) \cdot \kappa_i) \quad (\text{by Claim 30.4.13}). \end{aligned}$$

□

Claim 30.5.17. *The following holds:*

$$\Pr \left[T \text{ is an } (a_1, \dots, a_k)\text{-tree for } \mathbf{x} \mid \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \mathbf{Game}_{\text{IP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \\ T \leftarrow \text{SRTreeFinder}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \text{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \end{array} \right]$$

$$\geq \left(1 - (t+1) \cdot \left(1 - \prod_{i \in [k]} \left(1 - \frac{a_i - 1}{2^{r_i}} \right) \right) \right) - \Pr[\bar{E}],$$

where the event E is defined as

$$E := \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge \mathbf{V}_{\text{IP}}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) = 1 \end{array} \middle| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \leftarrow \\ \text{Game}_{\text{IP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \end{array} \right].$$

Proof. We use Claim 30.5.7 to rewrite the probability statement in the claim:

$$\begin{aligned} & \Pr \left[T \text{ is an } (a_1, \dots, a_k)\text{-tree for } \mathbf{x} \middle| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \text{Game}_{\text{IP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \\ T \leftarrow \text{SRTreeFinder}(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}, \text{tr}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \end{array} \right] \\ &= \Pr \left[\text{out} \neq \text{fail} \middle| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ \text{out} \leftarrow \text{SRTreeFinder}_0(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \mathbf{rnd}) \end{array} \right]. \end{aligned}$$

We are left to argue the inequality:

$$\begin{aligned} & \Pr \left[\text{out} \neq \text{fail} \middle| \begin{array}{l} \mathbf{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ \text{out} \leftarrow \text{SRTreeFinder}_0(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \mathbf{rnd}) \end{array} \right] \\ &= w_0 \quad (\text{by Equation 30.8 with } i = 0) \\ &\geq \left(\prod_{j=1}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot (w_k - (t+1) \cdot \kappa_0) \quad (\text{by Claim 30.5.16 with } i = 0) \\ &= \left(\prod_{j=1}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot (\Pr[E] - (t+1) \cdot \kappa_0) \quad (\text{by Definition 30.5.11}) \\ &= \left(\prod_{j=1}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot \left(\Pr[E] - (t+1) \cdot \left(1 - \prod_{i \in [k]} \left(1 - \frac{a_i - 1}{2^{r_i}} \right) \right) \right) \quad (\text{by Definition 30.4.12}) \\ &= \left(\prod_{j=1}^k \frac{2^{r_j}}{2^{r_j} - a_j + 1} \right) \cdot \left(1 - \Pr[\bar{E}] - (t+1) \cdot \left(1 - \prod_{i \in [k]} \left(1 - \frac{a_i - 1}{2^{r_i}} \right) \right) \right) \quad (\Pr[\bar{E}] + \Pr[E] = 1) \\ &\geq \left(1 - (t+1) \cdot \left(1 - \prod_{i \in [k]} \left(1 - \frac{a_i - 1}{2^{r_i}} \right) \right) \right) - \Pr[\bar{E}] \quad (\text{multiplicative term is at least 1}). \end{aligned}$$

□

Running time Here we focus on proving Claim 30.5.29. The expected running time of SRTreeFinder is dominated by the running time of SRTreeFinder_0 . The analysis focuses on upper bounding the expected running time of SRTreeFinder_0 ; more generally, the analysis focuses on SRTreeFinder_j for $j \in \{0, 1, \dots, k\}$.

Definition 30.5.18. For every $j \in \{0, 1, \dots, k\}$, define B_j to be the running time of

$$\text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \mathbf{rnd}; \zeta) = \text{SRTreeFinder}_j(1, \emptyset, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \mathbf{rnd}; \zeta),$$

for a random choice of $\mathbf{rnd} \in \mathcal{U}((r_i)_{i \in [k]})$ and SRTreeFinder_j randomness ζ .

Definition 30.5.19. For every $j \in \{0, 1, \dots, k - 1\}$ and $\text{rnd} \in \mathcal{U}((r_i)_{i \in [k]})$, the pseudo-output of $\text{SRTreeFinder}_{j+1}(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$ is the pair

$$((\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}), b)$$

where $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}) := \text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$ and $b := \mathbf{V}_{\text{IP}}(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]})$. By Claim 30.5.8, if $\text{SRTreeFinder}_{j+1}(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$ does not return fail then its output can be parsed as $((\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}), T_{j+1})$. Similarly, for every $(\mathbf{x}, \alpha_j, \sigma_j) \in T_j$, the pseudo-output of $\text{SRTreeFinder}_j(0, \mathbf{x}, \alpha_j, \sigma_j, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$ also equals the aforementioned pair.

Definition 30.5.20. For every $j \in \{0, 1, \dots, k - 1\}$, we define several random variables over a random choice of $\text{rnd} \in \mathcal{U}((r_i)_{i \in [k]})$ and internal randomness of SRTreeFinder_j . Below we denote by $((\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}), b)$ the pseudo-output of $\text{SRTreeFinder}_{j+1}(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$ (as invoked by SRTreeFinder_j in its Item 2).

- D_j is the decision bit in the pseudo-output of $\text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$.
- $B_{j,1}$ is the total time spent by $\text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$ on invocations of $\text{SRTreeFinder}_{j+1}$ whose pseudo-output equals $((\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}), 1)$; moreover, $M_{j,1}$ is the total number of such invocations.
- $B_{j,0}$ is the total time spent by $\text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$ on invocations of $\text{SRTreeFinder}_{j+1}$ whose pseudo-output equals $((\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}), 0)$; moreover, $M_{j,0}$ is the total number of such invocations.
- $B_{j,\neq}$ is the total time spent by $\text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$ on invocations of $\text{SRTreeFinder}_{j+1}$ whose pseudo-output equals $((\mathbf{x}'', (\alpha''_i)_{i \in [k]}, (\sigma''_i)_{i \in [k]}, (\rho''_i)_{i \in [k]}), b)$ with the condition that

$$(\mathbf{x}'', (\alpha''_i)_{i \in [k]}, (\sigma''_i)_{i \in [k]}, (\rho''_i)_{i \in [k]}) \neq (\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]})$$

(and any decision bit b); moreover, $M_{j,\neq}$ is the total number of such invocations.

Recalling that B_j denotes the running time of $\text{SRTreeFinder}_j(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$ (see Definition 30.5.18), note that:

$$B_j = B_{j,1} + B_{j,0} + B_{j,\neq} + \text{poly}(\text{len}(\mathbf{x})). \quad (30.9)$$

Definition 30.5.21. For every output size $\sigma \in \mathbb{N}$ and query $x \in \{0, 1\}^*$, we denote by $U_{\sigma,x}$ the set of all functions of the form $g: \{0, 1\}^* \setminus \{x\} \rightarrow \{0, 1\}^\sigma$ (the function is defined everywhere except at x). For every $j \in [k]$, $x_j = (\mathbf{x}, \alpha_j, \sigma_j) \in T_j$, and $g \in U_{r_j, x_j}$, we define $E_{j, x_j, g}$ to be the following event:

$$\left[\begin{array}{l} x_j = (\mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j) \\ \wedge g =_{x_j} \text{rnd}_j \end{array} \middle| \begin{array}{l} \text{rnd} \leftarrow \mathcal{U}((r_i)_{i \in [k]}) \\ (\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}) \xleftarrow{\text{tr}_{\text{sr}}} \text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \end{array} \right],$$

where $\text{rnd}_j =_{x_j} g$ denotes that g equals rnd_j everywhere but at x_j (where g is not defined).

Claim 30.5.22. For every $j \in [k]$, $x_j = (\mathbf{x}, \alpha_j, \sigma_j) \in T_j$, and $g \in U_{r_j, x_j}$, define

$$u_{x_j, g} := 2^{r_j} \cdot \Pr \left[\begin{array}{l} x_j = (\mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j) \\ \wedge \mathbf{V}_{\text{IP}}(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}) = 1 \\ \text{conditioned on} \\ g =_{x_j} \text{rnd}_j \end{array} \right],$$

$$v_{x_j,g} := 2^{r_j} \cdot \Pr \left[\begin{array}{l} x_j = (\mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j) \\ \wedge \mathbf{V}_{\text{IP}}(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}) = 0 \\ \text{conditioned on} \\ g =_{x_j} \mathsf{rnd}_j \end{array} \right],$$

where both probabilities are for the following experiment

$$\left[\begin{array}{l} \mathsf{rnd} \leftarrow \mathcal{U}((\mathsf{r}_i)_{i \in [k]}) \\ (\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \mathbf{Game}_{\text{IP}}^{\text{sr}}(s, \mathsf{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}) \end{array} \right].$$

The following equations directly follow from the two definitions:

$$\Pr \left[x_j = (\mathbf{x}', (\alpha'_i)_{i=1}^j, (\sigma'_i)_{i=1}^j) \mid g =_{x_j} \mathsf{rnd}_j \right] = \frac{u_{x_j,g} + v_{x_j,g}}{2^{r_j}}, \quad (30.10)$$

$$\Pr \left[D_j = 1 \mid E_{j,x_j,g} \right] = \frac{u_{x_j,g}}{u_{x_j,g} + v_{x_j,g}}, \quad (30.11)$$

$$\Pr \left[D_j = 0 \mid E_{j,x_j,g} \right] = \frac{v_{x_j,g}}{u_{x_j,g} + v_{x_j,g}}. \quad (30.12)$$

Claim 30.5.23. For every $j \in \{0, 1, \dots, k-1\}$, $x_{j+1} = (\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}$, and $g \in U_{r_{j+1}, x_{j+1}}$ such that $\Pr[E_{j+1, x_{j+1}, g}] > 0$,

$$\begin{aligned} \mathbb{E} [B_{j,1} \mid E_{j+1, x_{j+1}, g}] &= \mathbb{E} [B_{j+1} \mid D_{j+1} = 1 \wedge E_{j+1, x_{j+1}, g}] \cdot \mathbb{E} [M_{j,1} \mid E_{j+1, x_{j+1}, g}], \\ \mathbb{E} [B_{j,0} \mid E_{j+1, x_{j+1}, g}] &= \mathbb{E} [B_{j+1} \mid D_{j+1} = 0 \wedge E_{j+1, x_{j+1}, g}] \cdot \mathbb{E} [M_{j,0} \mid E_{j+1, x_{j+1}, g}]. \end{aligned}$$

Proof. We prove the first equality; the second equality is proved similarly.

Let $B_{j,1}^i$ denote the time spent on the i -th invocation of $\text{SRTreeFinder}_{j+1}$ whose pseudo-output equals $((\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}), 1)$. If $M_{j,1} = \ell$ then $B_{j,1} = \sum_{i \in [\ell]} B_{j,1}^i$ (see Definition 30.5.20). For every $\ell \in \{0, 1, \dots, 2^{r_{j+1}}\}$,

$$\mathbb{E} [B_{j,1} \mid E_{j+1, x_{j+1}, g} \wedge M_{j,1} = \ell] = \sum_{i=1}^{\ell} \mathbb{E} [B_{j,1}^i \mid E_{j+1, x_{j+1}, g} \wedge M_{j,1} = \ell] \quad (30.13)$$

$$= \sum_{i=1}^{\ell} \mathbb{E} [B_{j,1}^1 \mid E_{j+1, x_{j+1}, g} \wedge M_{j,1} = \ell] \quad (30.14)$$

$$= \sum_{i=1}^{\ell} \mathbb{E} [B_{j+1} \mid E_{j+1, x_{j+1}, g} \wedge D_{j+1} = 1] \quad (30.15)$$

$$= \mathbb{E} [B_{j+1} \mid D_{j+1} = 1 \wedge E_{j+1, x_{j+1}, g}] \cdot \ell. \quad (30.16)$$

We explain the equalities above.

1. Equation 30.13 follows from $B_{j,1} = \sum_{i \in [\ell]} B_{j,1}^i$ and the linearity of expectation.
2. Equation 30.14 follows from the fact that all invocations of $\text{SRTreeFinder}_{j+1}$ have (individually) the same expected running time, so, in particular, we can consider the expected running time of the first invocation. This merits an argument, so let us assume that there is more than one invocation ($\ell > 1$).

The first invocation receives rnd , while other invocations each receive rnd with rnd_{j+1} programmed at the query x_{j+1} with a different answer ρ'_{j+1} sampled from the set S (see

Item 7 in Construction 30.5.3), which is initially set to $S := \{0, 1\}^{r_{j+1}} \setminus \{\text{rnd}_{j+1}(x_{j+1})\}$ (see Item 6 in Construction 30.5.3). Each value in S has the same probability of being used for each of the invocations. Thus, the marginal distribution of $\text{rnd}[\mu_{j+1}]$ is identical to a fresh sample of rnd (with no programming).

Therefore, each invocation after the first one is individually distributed as

$$\text{SRTreeFinder}_{j+1}(0, \mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$$

where $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]}) := \text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$. We are left to compare this to the first invocation, which is $\text{SRTreeFinder}_{j+1}(1, \emptyset, \emptyset, \emptyset, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd})$.

The two types of executions are, in fact, identically distributed both in output distribution and running time. Indeed, if $j+1 = k$ then receiving $\mathbf{c} = 1$ and $(\emptyset, \emptyset, \emptyset)$ or $\mathbf{c} = 0$ and $(\mathbf{x}', (\alpha'_i)_{i=1}^k, (\sigma'_i)_{i=1}^k)$ does not change the execution. Moreover, if $j+1 < k$, the only difference between the two executions comes from the check in Item 2 of $\text{SRTreeFinder}_{j+1}$, which will pass because the $(\mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1})$ is contained in the output of $\text{Game}_{\text{IP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}})$. Overall, the expected running time remains the same.

3. Equation 30.15 follows by definition. The condition $M_{j,1} = \ell$ is no longer relevant because we are considering only the first invocation, so we can replace it with $D_{j+1} = 1$. In this case, the random variable $B_{j,1}^1$ corresponds to B_{j+1} .

Using the above, we deduce that

$$\begin{aligned} & \mathbb{E}[B_{j,1} \mid E_{j+1,x_{j+1},g}] \\ &= \sum_{\ell=0}^{2^{r_{j+1}}} \Pr[M_{j,1} = \ell \mid E_{j+1,x_{j+1},g}] \cdot \mathbb{E}[B_{j,1} \mid E_{j+1,x_{j+1},g} \wedge M_{j,1} = \ell] \quad (\text{by total probability}) \\ &= \sum_{\ell=0}^{2^{r_{j+1}}} \Pr[M_{j,1} = \ell \mid E_{j+1,x_{j+1},g}] \cdot (\mathbb{E}[B_{j+1} \mid D_{j+1} = 1 \wedge E_{j+1,x_{j+1},g}] \cdot \ell) \quad (\text{by Equation 30.16}) \\ &= \mathbb{E}[B_{j+1} \mid D_{j+1} = 1 \wedge E_{j+1,x_{j+1},g}] \cdot \sum_{\ell=0}^{2^{r_{j+1}}} \Pr[M_{j,1} = \ell \mid E_{j+1,x_{j+1},g}] \cdot \ell \\ &= \mathbb{E}[B_{j+1} \mid D_{j+1} = 1 \wedge E_{j+1,x_{j+1},g}] \cdot \mathbb{E}[M_{j,1} \mid E_{j+1,x_{j+1},g}]. \end{aligned}$$

□

Claim 30.5.24. *For every $j \in \{0, 1, \dots, k-1\}$, $x_{j+1} = (\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}$, and $g \in U_{r_{j+1},x_{j+1}}$ such that $\Pr[E_{j+1,x_{j+1},g}] > 0$,*

$$\begin{aligned} \mathbb{E}[M_{j,1} \mid E_{j+1,x_{j+1},g}] &\leq a_{j+1} \cdot \Pr[D_{j+1} = 1 \mid E_{j+1,x_{j+1},g}], \\ \mathbb{E}[M_{j,0} \mid E_{j+1,x_{j+1},g}] &\leq a_{j+1} \cdot \Pr[D_{j+1} = 0 \mid E_{j+1,x_{j+1},g}]. \end{aligned}$$

Proof. We prove the first inequality. If $D_{j+1} = 0$ then $M_{j,1} = 0$, because $D_{j+1} = 0$ implies that the first invocation of $\text{SRTreeFinder}_{j+1}$ fails, so SRTreeFinder_j fails as well (see Item 3 in Construction 30.5.3); hence there are no invocations of $\text{SRTreeFinder}_{j+1}$ whose pseudo-output contains a decision bit 1, which means that $M_{j,1} = 0$. Conversely, if $D_{j+1} = 1$ then $M_{j,1} \leq a_{j+1}$, because SRTreeFinder_j looks for at most a_{j+1} executions of $\text{SRTreeFinder}_{j+1}$ that do not fail.

Therefore,

$$\mathbb{E}[M_{j,1} \mid E_{j+1,x_{j+1},g}]$$

$$\begin{aligned}
&= \Pr[D_{j+1} = 0 \mid E_{j+1,x_{j+1},g}] \cdot \mathbb{E}[M_{j,1} \mid E_{j+1,x_{j+1},g} \wedge D_{j+1} = 0] \\
&\quad + \Pr[D_{j+1} = 1 \mid E_{j+1,x_{j+1},g}] \cdot \mathbb{E}[M_{j,1} \mid E_{j+1,x_{j+1},g} \wedge D_{j+1} = 1] \\
&\leq \Pr[D_{j+1} = 0 \mid E_{j+1,x_{j+1},g}] \cdot 0 + \Pr[D_{j+1} = 1 \mid E_{j+1,x_{j+1},g}] \cdot a_{j+1} \\
&= a_{j+1} \cdot \Pr[D_{j+1} = 1 \mid E_{j+1,x_{j+1},g}].
\end{aligned}$$

Next, we prove the second inequality. Observe that:

- $\mathbb{E}[M_{j,0} \mid E_{j+1,x_{j+1},g} \wedge D_{j+1} = 1] \leq (a_{j+1} - 1) \cdot \frac{v_{x_{j+1},g}}{u_{x_{j+1},g}}$. Conditioned on $E_{j+1,x_{j+1},g} \wedge D_{j+1} = 1$, the random variable $M_{j,0}$ follows the random variable Y_{NHC} defined for the negative hypergeometric distribution (see Definition 30.1.7) with a set size $u_{x_{j+1},g} + v_{x_{j+1},g} - 1$, a total of $u_{x_{j+1},g} - 1$ blue elements, and a target of $a_{j+1} - 1$ blue elements; therefore the expectation (with conditioning) is at most $(a_{j+1} - 1) \cdot \frac{(u_{x_{j+1},g} + v_{x_{j+1},g} - 1) - (u_{x_{j+1},g} - 1)}{(u_{x_{j+1},g} - 1) + 1} = (a_{j+1} - 1) \cdot \frac{v_{x_{j+1},g}}{u_{x_{j+1},g}}$.
- $\mathbb{E}[M_{j,0} \mid E_{j+1,x_{j+1},g} \wedge D_{j+1} = 0] = 1$. If $D_{j+1} = 0$ then $\text{SRTreeFinder}_{j+1}(1, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}; \zeta)$ (the first invocation of $\text{SRTreeFinder}_{j+1}$ by SRTreeFinder_j) fails, so SRTreeFinder_j aborts early, in which case $M_{j,0} = 1$. (There was a single, and failed, invocation of $\text{SRTreeFinder}_{j+1}$.)

We conclude as follows:

$$\begin{aligned}
&\mathbb{E}[M_{j,0} \mid E_{j+1,x_{j+1},g}] \\
&= \Pr[D_{j+1} = 0 \mid E_{j+1,x_{j+1},g}] \cdot \mathbb{E}[M_{j,0} \mid E_{j+1,x_{j+1},g} \wedge D_{j+1} = 0] \\
&\quad + \Pr[D_{j+1} = 1 \mid E_{j+1,x_{j+1},g}] \cdot \mathbb{E}[M_{j,0} \mid E_{j+1,x_{j+1},g} \wedge D_{j+1} = 1] \\
&\leq \Pr[D_{j+1} = 0 \mid E_{j+1,x_{j+1},g}] \cdot 1 + \Pr[D_{j+1} = 1 \mid E_{j+1,x_{j+1},g}] \cdot (a_{j+1} - 1) \cdot \frac{v_{x_{j+1},g}}{u_{x_{j+1},g}} \\
&= \frac{v_{x_{j+1},g}}{u_{x_{j+1},g} + v_{x_{j+1},g}} + \frac{u_{x_{j+1},g}}{u_{x_{j+1},g} + v_{x_{j+1},g}} \cdot (a_{j+1} - 1) \cdot \frac{v_{x_{j+1},g}}{u_{x_{j+1},g}} \quad (\text{by Equations 30.11 and 30.12}) \\
&= a_{j+1} \cdot \frac{v_{x_{j+1},g}}{u_{x_{j+1},g} + v_{x_{j+1},g}} \\
&= a_{j+1} \cdot \Pr[D_{j+1} = 0 \mid E_{j+1,x_{j+1},g}].
\end{aligned}$$

□

Claim 30.5.25. *For every $j \in \{0, 1, \dots, k - 1\}$,*

$$\mathbb{E}[B_{j,1} + B_{j,0}] \leq a_{j+1} \cdot \mathbb{E}[B_{j+1}].$$

Proof. It suffices to argue that, for every $x_{j+1} = (\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}$ and $g \in U_{r_{j+1},x_{j+1}}$ such that $\Pr[E_{j+1,x_{j+1},g}] > 0$,

$$\mathbb{E}[B_{j,1} + B_{j,0} \mid E_{j+1,x_{j+1},g}] \leq a_{j+1} \cdot \mathbb{E}[B_{j+1} \mid E_{j+1,x_{j+1},g}].$$

Indeed, by combining Claim 30.5.23 and Claim 30.5.24, we obtain

$$\begin{aligned}
\mathbb{E}[B_{j,1} \mid E_{j+1,x_{j+1},g}] &\leq \mathbb{E}[B_{j+1} \mid D_{j+1} = 1 \wedge E_{j+1,x_{j+1},g}] \cdot (a_{j+1} \cdot \Pr[D_{j+1} = 1 \mid E_{j+1,x_{j+1},g}]), \\
\mathbb{E}[B_{j,0} \mid E_{j+1,x_{j+1},g}] &\leq \mathbb{E}[B_{j+1} \mid D_{j+1} = 0 \wedge E_{j+1,x_{j+1},g}] \cdot (a_{j+1} \cdot \Pr[D_{j+1} = 0 \mid E_{j+1,x_{j+1},g}]).
\end{aligned}$$

Adding the two inequalities we obtain the desired upper bound:

$$\begin{aligned} & \mathbb{E}[B_{j,1} + B_{j,0} \mid E_{j+1,x_{j+1},g}] \\ & \leq a_{j+1} \cdot (\mathbb{E}[B_{j+1} \mid D_{j+1} = 1 \wedge E_{j+1,x_{j+1},g}] \cdot \Pr[D_{j+1} = 1 \mid E_{j+1,x_{j+1},g}] \\ & \quad + \mathbb{E}[B_{j+1} \mid D_{j+1} = 0 \wedge E_{j+1,x_{j+1},g}] \cdot \Pr[D_{j+1} = 0 \mid E_{j+1,x_{j+1},g}]) \\ & = a_{j+1} \cdot \mathbb{E}[B_{j+1} \mid E_{j+1,x_{j+1},g}] . \end{aligned}$$

□

Claim 30.5.26. *For every $j \in \{0, 1, \dots, k - 1\}$,*

$$\mathbb{E}[B_{j,\neq}] \leq (a_{j+1} - 1) \cdot t \cdot \left(\tau_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}} + \text{poly}(\text{len}(\mathbf{x})) \right) .$$

Proof. First we argue that $\mathbb{E}[M_{j,\neq}] \leq (a_{j+1} - 1) \cdot t$. For every $x_{j+1} = (\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}$ and $g \in U_{r_{j+1},x_{j+1}}$ such that $\Pr[E_{j+1,x_{j+1},g}] > 0$, the following equations hold:

$$\mathbb{E}[M_{j,\neq} \mid E_{j+1,x_{j+1},g} \wedge D_{j+1} = 0] = 0 , \tag{30.17}$$

$$\mathbb{E}[M_{j,\neq} \mid E_{j+1,x_{j+1},g} \wedge D_{j+1} = 1] \leq (a_{j+1} - 1) \cdot \frac{2^{r_{j+1}} - u_{x_{j+1},g} - v_{x_{j+1},g}}{u_{x_{j+1},g}} , \tag{30.18}$$

$$\mathbb{E}[M_{j,\neq} \mid E_{j+1,x_{j+1},g}] \leq \Pr[D_{j+1} = 1 \mid E_{j+1,x_{j+1},g}] \cdot \left((a_{j+1} - 1) \cdot \frac{2^{r_{j+1}} - u_{x_{j+1},g} - v_{x_{j+1},g}}{u_{x_{j+1},g}} \right) . \tag{30.19}$$

Equation 30.17 follows from the fact that $E_{j+1,x_{j+1},g}$ implies that $x_{j+1} = (\mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1})$, and thus $M_{j,\neq} = 0$. Equation 30.18 follows from the fact that, conditioned on $E_{j+1,x_{j+1},g} \wedge D_{j+1} = 1$, $M_{j,\neq}$ follows the random variable Y_{NHG} defined for the negative hypergeometric distribution (see Definition 30.1.7) from a set with $2^{r_{j+1}} - v_{x_{j+1},g} - 1$ elements, a total of $u_{x_{j+1},g} - 1$ elements, and a target of $a_{j+1} - 1$ blue elements. Equation 30.19 is derived as follows:

$$\begin{aligned} & \mathbb{E}[M_{j,\neq} \mid E_{j+1,x_{j+1},g}] \\ & = \Pr[D_{j+1} = 0 \mid E_{j+1,x_{j+1},g}] \cdot \mathbb{E}[M_{j,\neq} \mid E_{j+1,x_{j+1},g} \wedge D_{j+1} = 0] \\ & \quad + \Pr[D_{j+1} = 1 \mid E_{j+1,x_{j+1},g}] \cdot \mathbb{E}[M_{j,\neq} \mid E_{j+1,x_{j+1},g} \wedge D_{j+1} = 1] \quad (\text{by total probability}) \\ & = \Pr[D_{j+1} = 0 \mid E_{j+1,x_{j+1},g}] \cdot 0 \\ & \quad + \Pr[D_{j+1} = 1 \mid E_{j+1,x_{j+1},g}] \cdot \mathbb{E}[M_{j,\neq} \mid E_{j+1,x_{j+1},g} \wedge D_{j+1} = 1] \quad (\text{by Equation 30.17}) \\ & \leq \Pr[D_{j+1} = 1 \mid E_{j+1,x_{j+1},g}] \cdot \left((a_{j+1} - 1) \cdot \frac{2^{r_{j+1}} - u_{x_{j+1},g} - v_{x_{j+1},g}}{u_{x_{j+1},g}} \right) \quad (\text{by Equation 30.18}) . \end{aligned}$$

(The above assumes that $\Pr[D_{j+1} = 0 \wedge E_{j+1,x_{j+1},g}] > 0$ and $\Pr[D_{j+1} = 1 \wedge E_{j+1,x_{j+1},g}] > 0$, and therefore the conditioned random variables are well-defined. Otherwise, if any of these probabilities are 0, then we can replace the corresponding terms with 0 and the inequality still holds.)

In light of the above, we write:

$$\mathbb{E}[M_{j,\neq} \mid E_{j+1,x_{j+1},g}]$$

$$\begin{aligned}
&\leq \frac{u_{x_{j+1},g}}{u_{x_{j+1},g} + v_{x_{j+1},g}} \cdot \left((a_{j+1} - 1) \cdot \frac{2^{r_{j+1}} - u_{x_{j+1},g} - v_{x_{j+1},g}}{u_{x_{j+1},g}} \right) \quad (\text{by Equation 30.11}) \\
&= (a_{j+1} - 1) \cdot \frac{2^{r_{j+1}} - u_{x_{j+1},g} - v_{x_{j+1},g}}{u_{x_{j+1},g} + v_{x_{j+1},g}} \\
&= (a_{j+1} - 1) \cdot \left(\frac{2^{r_{j+1}}}{u_{x_{j+1},g} + v_{x_{j+1},g}} - 1 \right) \\
&= (a_{j+1} - 1) \cdot \left(\frac{1}{\Pr[x_{j+1} = (\mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1}) \mid g =_{x_{j+1}} \text{rnd}_{j+1}]} - 1 \right) \quad (\text{by Equation 30.10}) \\
&= (a_{j+1} - 1) \cdot \left(\frac{\Pr[g =_{x_{j+1}} \text{rnd}_{j+1}]}{\Pr[E_{j+1,x_{j+1},g}]} - 1 \right) \quad (\text{by Definition 30.5.21}) \\
&= (a_{j+1} - 1) \cdot \left(\frac{\Pr[g =_{x_{j+1}} \text{rnd}_{j+1}] - \Pr[E_{j+1,x_{j+1},g}]}{\Pr[E_{j+1,x_{j+1},g}]} \right) \\
&= (a_{j+1} - 1) \cdot \left(\frac{\Pr[x_{j+1} \neq (\mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1}) \wedge g =_{x_{j+1}} \text{rnd}_{j+1}]}{\Pr[E_{j+1,x_{j+1},g}]} \right).
\end{aligned}$$

Define $U^* := \{g \in U_{r_{j+1},x_{j+1}} : u_{x_j,g} > 0\}$. For every $g \notin U^*$, $E_{j+1,x_{j+1},g}$ implies that $D_{j+1} = 0$, and in turn that $\mathbb{E}[M_{j,\neq} \mid E_{j+1,x_{j+1},g}] = 0$. Therefore,

$$\begin{aligned}
&\mathbb{E}[M_{j,\neq}] \\
&= \sum_{x_{j+1}=(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \sum_{g \in U^*} \Pr[E_{j+1,x_{j+1},g}] \cdot \mathbb{E}[M_{j,\neq} \mid E_{j+1,x_{j+1},g}] \\
&\leq (a_{j+1} - 1) \cdot \sum_{x_{j+1}=(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \sum_{g \in U^*} \Pr[x_{j+1} \neq (\mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1}) \wedge g =_{x_{j+1}} \text{rnd}_{j+1}] \\
&= (a_{j+1} - 1) \cdot \sum_{x_{j+1}=(\mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}) \in T_{j+1}} \Pr \left[\begin{array}{l} Y_{j+1, \text{rnd}, \mathbf{x}, \boldsymbol{\alpha}_{j+1}, \boldsymbol{\sigma}_{j+1}} > 0 \\ \wedge x_{j+1} \neq (\mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1}) \end{array} \right] \\
&\leq (a_{j+1} - 1) \cdot t \quad (\text{by Claim 30.5.14}).
\end{aligned}$$

Next, we conclude the claim by arguing that $B_{j,\neq} \leq M_{j,\neq} \cdot (\tau_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}} + \text{poly}(\text{len}(\mathbf{x})))$. This directly follows from the fact that the running time of an invocation of $\text{SRTreeFinder}_{j+1}$ whose pseudo-output is $((\mathbf{x}'', (\alpha''_i)_{i \in [k]}, (\sigma''_i)_{i \in [k]}, (\rho''_i)_{i \in [k]}), b)$ with the condition that $(\mathbf{x}'', (\alpha''_i)_{i \in [k]}, (\sigma''_i)_{i \in [k]}, (\rho''_i)_{i \in [k]}) \neq (\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]})$ (and any decision bit b) is $\tau_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}} + \text{poly}(\text{len}(\mathbf{x}))$. Note that an invocation of $\text{SRTreeFinder}_{j+1}$ by SRTreeFinder_j that satisfies the above property can only happen within the while loop (see Item 7c in Construction 30.5.3), which receives inputs as follows: $\text{SRTreeFinder}_{j+1}(0, \mathbf{x}', (\alpha'_i)_{i=1}^{j+1}, (\sigma'_i)_{i=1}^{j+1}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \text{rnd}[\mu_{j+1}])$.

In the (base) case $j = k - 1$, SRTreeFinder_k runs the IP state-restoration game with $\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}$ and performs other checks (which, e.g., involve computing the decision bit b); this takes time $\tau_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}} + \text{poly}(\text{len}(\mathbf{x}))$. In the (recursive) case $j < k - 1$, SRTreeFinder_j runs $\text{SRTreeFinder}_{j+1}$ in Item 2 and possibly more times in Item 7c in the while loop. The first such invocation may satisfy the above condition only if $c = 0$, in which case SRTreeFinder_j terminates the execution of $\text{SRTreeFinder}_{j+1}$ as soon as the first run of the IP state-restoration game reveals that the pseudo-output does not match with $(\mathbf{x}', (\alpha'_i)_{i \in [k]}, (\sigma'_i)_{i \in [k]}, (\rho'_i)_{i \in [k]})$; this takes time

$\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x}))$. As for other executions in the while loop (if any), if they satisfy the above condition then, by induction, run in time $\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x}))$. \square

Claim 30.5.27. *For every $j \in \{0, 1, \dots, k - 1\}$,*

$$\mathbb{E}[B_j] \leq a_{j+1} \cdot \mathbb{E}[B_{j+1}] + (a_{j+1} - 1) \cdot t \cdot \left(\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x})) \right).$$

Proof. This directly follows from Equation 30.9, Claim 30.5.25, and Claim 30.5.26. \square

Claim 30.5.28. *For every $i \in \{0, 1, \dots, k - 1\}$,*

$$\mathbb{E}_{rnd,\zeta}[B_{i,rnd,\zeta}] \leq (t + 1) \cdot \prod_{j=i+1}^k a_j \cdot \left(\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x})) \right).$$

Moreover, for every rnd and ζ , $B_{k,rnd,\zeta} = \tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x}))$.

Proof. The algorithm $SRTreeFinder_k$ (see Item 1) runs the IP state-restoration game with $\tilde{\mathbf{P}}_{IP}^{sr}$ and performs some $\text{poly}(\text{len}(\mathbf{x}))$ -time operations, so $B_{k,rnd,\zeta} = \tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x}))$. Next, by Claim 30.5.27, for every $i \in \{0, 1, \dots, k - 1\}$,

$$\mathbb{E}_{rnd,\zeta}[B_{i,rnd,\zeta}] \leq a_{i+1} \cdot \mathbb{E}_{rnd,\zeta}[B_{i+1,rnd,\zeta}] + (a_{i+1} - 1) \cdot t \cdot \left(\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x})) \right). \quad (30.20)$$

We explain how Equation 30.20 implies the claim. Define for notational convenience $a_{k+1} := 1$. We prove by induction on $i \in \{0, 1, \dots, k\}$ that

$$\mathbb{E}_{rnd,\zeta}[B_{i,rnd,\zeta}] \leq \left((t + 1) \cdot \prod_{j=i+1}^{k+1} a_j - t \right) \cdot \left(\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x})) \right),$$

which directly implies the claim. For the base case, we already proved that $B_{k,rnd,\zeta} = \tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x}))$, which is at most $((t + 1) \cdot a_{k+1} - t) \cdot (\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x}))) = ((t + 1) \cdot 1 - t) \cdot (\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x}))) = (\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x})))$. For the inductive case, we assume the inequality for $i + 1 \leq k$, and prove it for i :

$$\begin{aligned} & \mathbb{E}_{rnd,\zeta}[B_{i,rnd,\zeta}] \\ & \leq a_{i+1} \cdot \mathbb{E}_{rnd,\zeta}[B_{i+1,rnd,\zeta}] + (a_{i+1} - 1) \cdot t \cdot \left(\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x})) \right) \quad (\text{by Equation 30.20}) \\ \\ & \leq a_{i+1} \cdot \left((t + 1) \cdot \prod_{j=i+2}^{k+1} a_j - t \right) \cdot \left(\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x})) \right) + (a_{i+1} - 1) \cdot t \cdot \left(\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x})) \right) \\ & \quad (\text{by inductive hypothesis}) \\ & = \left((t + 1) \cdot \prod_{j=i+1}^{k+1} a_j - a_{i+1} \cdot t + a_{i+1} \cdot t - t \right) \cdot \left(\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x})) \right) \\ & = \left((t + 1) \cdot \prod_{j=i+1}^{k+1} a_j - t \right) \cdot \left(\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x})) \right). \end{aligned}$$

\square

Claim 30.5.29. SRTreeFinder runs in expected time $(t+1) \cdot (\prod_{i \in [k]} a_i - 1) \cdot (\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x})))$.

Proof. The running time of SRTreeFinder is dominated by the running time of SRTreeFinder_0 (invoked in Item 3). Claim 30.5.28 for $i = 0$ tells us that $\text{SRTreeFinder}_0(1, \tilde{\mathbf{P}}_{IP}^{sr}, \text{rnd})$ runs in expected time $(t+1) \cdot (\prod_{i \in [k]} a_i) \cdot (\tau_{\tilde{\mathbf{P}}_{IP}^{sr}} + \text{poly}(\text{len}(\mathbf{x})))$.

The “ -1 ” in the statement of the claim denotes the fact that SRTreeFinder receives as input the output $(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$ of $\text{Game}_{IP}^{sr}(s, \text{rnd}, \tilde{\mathbf{P}}_{IP}^{sr})$, so SRTreeFinder_0 can be modified to avoid computing the left-most branch of the recursion tree (which corresponds to the output $(\mathbf{x}, (\alpha_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]})$). \square

31 Round-by-round soundness

Round-by-round soundness (and knowledge soundness) is a property that implies state-restoration soundness (and knowledge soundness), and is satisfied by public-coin IPs and public-coin IOPs of practical interest (see Chapter 27). In particular, a public-coin IP with round-by-round soundness is suitable for use by the Fiat–Shamir transformation (Construction 14.1.1 or Construction 15.1.1), and a public-coin IOP with round-by-round soundness is suitable for use by the BCS transformation (Construction 25.1.1). Round-by-round soundness (and knowledge soundness) are attractive notions because they are relatively straightforward to define, and it can be easier to establish that a given protocol satisfies them, in comparison to establishing state-restoration soundness (and knowledge soundness).

The material in this chapter is written for public-coin IOPs, and corresponding statements hold for public-coin IPs (which are the special case where the verifier reads the prover’s messages in their entirety).

Organization In Section 31.1 we define the notions of round-by-round soundness and round-by-round knowledge soundness. In Section 31.2 we show how round-by-round soundness implies state-restoration soundness. In Section 31.3 we show how round-by-round knowledge soundness implies state-restoration knowledge soundness.

31.1 Definition

An IOP (in particular, IP) with round-by-round soundness is such that we can assign a bad event to each round of the protocol, and when that bad event happens then the interaction is “doomed” in that the malicious prover may have a way to win. Each round’s bad event is upper bounded by a corresponding error.

State function The definition of round-by-round soundness relies on the notion of a *state function* for a (possibly partial) transcript of interaction. Informally, the state function receives as input an instance \mathbf{x} and a (possibly partial) transcript trc , and outputs a bit denoting whether the transcript is doomed. An empty transcript is not doomed; IOP prover messages cannot make a transcript doomed; and a full transcript that is doomed is rejected by the IOP verifier.

Definition 31.1.1. A state function for an IOP $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ is a deterministic (possibly inefficient) function RBRState that receives as input an instance \mathbf{x} and an interaction transcript trc and outputs a bit for which the following holds.

- Empty transcript: if $\text{trc} = \emptyset$ is the empty transcript then $\text{RBRState}(\mathbf{x}, \text{trc}) = 0$.
- Prover moves: if $\text{trc} = (\Pi_1, \rho_1, \dots, \Pi_i, \rho_i)$ is a transcript for i rounds (for $i < k$) with $\text{RBRState}(\mathbf{x}, \text{trc}) = 0$ then, for every IOP string $\Pi_{i+1} \in \Sigma^{l_{i+1}}$, $\text{RBRState}(\mathbf{x}, \text{trc} \parallel \Pi_{i+1}) = 0$.
- Full transcript: if $\text{trc} = (\Pi_1, \rho_1, \dots, \Pi_k, \rho_k)$ is a full transcript and $\text{RBRState}(\mathbf{x}, \text{trc}) = 0$ then $\mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 0$.

RBR soundness Round-by-round soundness states that for, every partial transcript where the IOP verifier is about to move (to send randomness), if the instance is not in the language and the transcript is not doomed, then the probability that the next random message of the IOP verifier makes the transcript doomed is upper bounded by an error. Different rounds may have different errors bounding this event, or one can consider a single error bounding all of them.

Definition 31.1.2. An IOP $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ for a relation \mathcal{R} has **round-by-round soundness errors** $(\epsilon_{\text{IOP},i}^{\text{rbr}})_{i \in [k]}$ if there exists a state function RBRState such that for every instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$, round index $i \in [k]$, and malicious prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}$ the following holds:

$$\Pr \left[\begin{array}{l} \text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_i)) = 0 \\ \text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_i, \rho_i)) = 1 \end{array} \middle| \begin{array}{l} ((\Pi_j)_{j \in [i]}, (\rho_j)_{j \in [i-1]}) \leftarrow \tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}} \\ \rho_i \leftarrow \{0, 1\}^{r_i} \end{array} \right] \leq \epsilon_{\text{IOP},i}^{\text{rbr}}(\mathbf{x}).$$

The IOP has **round-by-round soundness error** $\epsilon_{\text{IOP}}^{\text{rbr}}$ if for every $i \in [k]$ it holds that $\epsilon_{\text{IOP},i}^{\text{rbr}} \leq \epsilon_{\text{IOP}}^{\text{rbr}}$.

We additionally define $\epsilon_{\text{IOP},i}^{\text{rbr}}(n) := \max \{ \epsilon_{\text{IOP},i}^{\text{rbr}}(\mathbf{x}) \mid \mathbf{x} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \text{ and } \mathbf{x} \notin \mathcal{L}(\mathcal{R}) \}$.

In the special case where $k = 1$, the above definition corresponds to the (standard) soundness definition for PCPs (see Definition 19.1.2): a 1-round IOP with round-by-round soundness error $\epsilon_{\text{IOP}}^{\text{rbr}}$ is a PCP with soundness error $\epsilon_{\text{PCP}} := \epsilon_{\text{IOP}}^{\text{rbr}}$.

More generally, below we prove that the (standard) soundness of an IOP is at most the sum of its round-by-round soundness errors $(\epsilon_{\text{IOP},i}^{\text{rbr}})_{i \in [k]}$ (itself at most k times a single round-by-round soundness error $\epsilon_{\text{IOP}}^{\text{rbr}}$ that bounds all of them). In fact, later in Section 31.2, we prove that round-by-round soundness implies state-restoration soundness.

Claim 31.1.3. Let $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ be a public-coin IOP for a relation \mathcal{R} . If IOP has round-by-round soundness errors $(\epsilon_{\text{IOP},i}^{\text{rbr}})_{i \in [k]}$ then IOP has (standard) soundness error ϵ_{IOP} (see Definition 23.1.2) such that for every instance $\mathbf{x} \notin \mathcal{L}(\mathcal{R})$

$$\epsilon_{\text{IOP}}(\mathbf{x}) \leq \sum_{i \in [k]} \epsilon_{\text{IOP},i}^{\text{rbr}}(\mathbf{x}) \leq k \cdot \epsilon_{\text{IOP}}^{\text{rbr}}(\mathbf{x}).$$

Proof. Fix a malicious IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$. We wish to upper bound the following probability:

$$\Pr[\langle \tilde{\mathbf{P}}_{\text{IOP}}, \mathbf{V}_{\text{IOP}}(\mathbf{x}) \rangle_{\text{IOP}} = 1].$$

Let $\text{trc} = (\Pi_1, \rho_1, \dots, \Pi_k, \rho_k)$ be the random variable denoting the interaction transcript between $\tilde{\mathbf{P}}_{\text{IOP}}$ and $\mathbf{V}_{\text{IOP}}(\mathbf{x})$. For every $i \in [k]$, let X_i be an indicator random variable for the event that $\text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_i, \rho_i)) = 1$. Observe that if $\mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1$ then it holds that $\text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_k, \rho_k)) = 1$. Hence it suffices to upper bound $\Pr[X_k = 1]$.

Let $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i)$ be a round-by-round prover that outputs an interaction transcript between $\tilde{\mathbf{P}}_{\text{IOP}}$ and $\mathbf{V}_{\text{IOP}}(\mathbf{x})$ up to and including the i -th IOP string of $\tilde{\mathbf{P}}_{\text{IOP}}$. Define $X_0 := 0$. For every $i \in [k]$, by round-by-round soundness,

$$\begin{aligned} & \Pr[X_i = 1 \mid X_{i-1} = 0] \\ &= \Pr \left[\begin{array}{l} \text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_i, \rho_i)) = 1 \\ \text{conditioned on} \\ X_{i-1} = 0 \end{array} \middle| \begin{array}{l} ((\Pi_j)_{j \in [i]}, (\rho_j)_{j \in [i-1]}) \leftarrow \tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i) \\ \rho_i \leftarrow \{0, 1\}^{r_i} \end{array} \right] \end{aligned}$$

$$\leq \epsilon_{\text{IOP},i}^{\text{rbr}}(\mathbf{x}).$$

Then, for every $i \in [k]$ we can write

$$\begin{aligned} & \Pr[X_i = 1] \\ &= \Pr[X_i = 1 \mid X_{i-1} = 0] \cdot \Pr[X_{i-1} = 0] + \Pr[X_i = 1 \mid X_{i-1} = 1] \cdot \Pr[X_{i-1} = 1] \\ &\leq \Pr[X_i = 1 \mid X_{i-1} = 0] + \Pr[X_{i-1} = 1]. \end{aligned}$$

Applying the above for every $i \in [k]$ we get

$$\begin{aligned} & \Pr[X_k = 1] \\ &\leq \Pr[X_k = 1 \mid X_{k-1} = 0] + \Pr[X_{k-1} = 1] \\ &\leq \dots \\ &\leq \sum_{i \in [k]} \Pr[X_i = 1 \mid X_{i-1} = 0] \\ &\leq \sum_{i \in [k]} \epsilon_{\text{IOP},i}^{\text{rbr}}(\mathbf{x}) \\ &\leq k \cdot \epsilon_{\text{IOP}}^{\text{rbr}}(\mathbf{x}). \end{aligned}$$

□

RBR knowledge soundness Round-by-round knowledge soundness strengthens round-by-round soundness to require that if, at some round, the probability that a given partial transcript that is not doomed becomes doomed with probability above a certain threshold then an extractor can efficiently find the witness from this transcript (and, indeed, any continuation of it). More precisely, the extractor only needs the IOP strings sent by the prover in the transcript.

Definition 31.1.4. An IOP $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ for a relation \mathcal{R} has **round-by-round knowledge errors** $(\kappa_{\text{IOP},i}^{\text{rbr}})_{i \in [k]}$ if there exists a polynomial-time extractor $\mathbf{E}_{\text{IOP}}^{\text{rbr}}$ and state function RBRState such that the following holds. For every instance \mathbf{x} , round index $i \in [k]$, and malicious prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}$,

$$\Pr \left[\begin{array}{l} \text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_i)) = 0 \\ \text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_i, \rho_i)) = 1 \\ (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \end{array} \middle| \begin{array}{l} ((\Pi_i)_{i \in [k]}, (\rho_j)_{j \in [i-1]}) \leftarrow \tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}} \\ \rho_i \leftarrow \{0, 1\}^{r_i} \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IOP}}^{\text{rbr}}(\mathbf{x}, (\Pi_i)_{i \in [k]}) \end{array} \right] \leq \kappa_{\text{IOP},i}^{\text{rbr}}(\mathbf{x}).$$

The IOP has **round-by-round knowledge soundness error** $\kappa_{\text{IOP}}^{\text{rbr}}$ if for every $i \in [k]$ it holds that $\kappa_{\text{IOP},i}^{\text{rbr}} \leq \kappa_{\text{IOP}}^{\text{rbr}}$.

We additionally define $\kappa_{\text{IOP},i}^{\text{rbr}}(n) := \max \{ \kappa_{\text{IOP},i}^{\text{rbr}}(\mathbf{x}) \mid \mathbf{x} \in \{0, 1\}^* \text{ with } |\mathbf{x}| \leq n \}$.

In the special case where $k = 1$, the above definition corresponds to the (standard) knowledge soundness definition for PCPs (see Definition 19.1.3): a 1-round IOP with round-by-round knowledge soundness error $\kappa_{\text{IOP}}^{\text{rbr}}$ is a PCP with knowledge soundness error $\kappa_{\text{PCP}} := \kappa_{\text{IOP}}^{\text{rbr}}$.

Similarly to the case of soundness, the (standard) knowledge soundness of an IOP is at most the sum of its round-by-round knowledge soundness errors. Moreover, later in Section 31.3, we prove that round-by-round knowledge soundness implies state-restoration knowledge soundness.

Claim 31.1.5. Let $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ be a public-coin IOP for a relation \mathcal{R} . If IOP has round-by-round knowledge soundness errors $(\kappa_{\text{IOP},i}^{\text{rbr}})_{i \in [k]}$ then IOP has (standard) straightline knowledge soundness error κ_{IOP} (see Definition 23.1.6) such that for every instance \mathbf{x}

$$\kappa_{\text{IOP}}(\mathbf{x}) \leq \sum_{i \in [k]} \kappa_{\text{IOP},i}^{\text{rbr}}(\mathbf{x}) \leq k \cdot \kappa_{\text{IOP}}^{\text{rbr}}(\mathbf{x}).$$

Construction 31.1.6. Let $\mathbf{E}_{\text{IOP}}^{\text{rbr}}$ be the round-by-round extractor for IOP . We construct an extractor \mathbf{E}_{IOP} that, given as input an instance \mathbf{x} and interaction transcript trc , works as follows.

$\mathbf{E}_{\text{IOP}}(\mathbf{x}, \text{trc})$:

1. Parse trc as a tuple $(\Pi_1, \rho_1, \dots, \Pi_k, \rho_k)$.
2. Compute $\mathbf{w} := \mathbf{E}_{\text{IOP}}^{\text{rbr}}(\mathbf{x}, (\Pi_i)_{i \in [k]})$.
3. Output \mathbf{w} .

Proof. Fix a malicious IOP prover $\tilde{\mathbf{P}}_{\text{IOP}}$. We wish to upper bound the following probability

$$\Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} b \xleftarrow{\text{trc}} \langle \tilde{\mathbf{P}}_{\text{IOP}}, \mathbf{V}_{\text{IOP}}(\mathbf{x}) \rangle_{\text{IOP}} \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IOP}}(\mathbf{x}, \text{trc}) \end{array} \right].$$

By construction of \mathbf{E}_{IOP} , the above is equivalent to the following

$$\Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge b = 1 \end{array} \middle| \begin{array}{l} b \xleftarrow{(\Pi_1, \rho_1, \dots, \Pi_k, \rho_k)} \langle \tilde{\mathbf{P}}_{\text{IOP}}, \mathbf{V}_{\text{IOP}}(\mathbf{x}) \rangle_{\text{IOP}} \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IOP}}^{\text{rbr}}(\mathbf{x}, (\Pi_i)_{i \in [k]}) \end{array} \right].$$

For every $i \in [k]$, define X_i to be the indicator random variable for the event

$$\text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_i, \rho_i)) = 1.$$

If $\mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1$ then $\text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_k, \rho_k)) = 1$; hence it suffices to upper bound the probability $\Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_k = 1]$.

Let $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i)$ be a round-by-round prover that outputs an interaction transcript between $\tilde{\mathbf{P}}_{\text{IOP}}$ and $\mathbf{V}_{\text{IOP}}(\mathbf{x})$ up to and including the i -th IOP string of $\tilde{\mathbf{P}}_{\text{IOP}}$. Define $X_0 := 0$. For every $i \in [k]$, by round-by-round knowledge soundness,

$$\begin{aligned} & \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_i = 1 \mid X_{i-1} = 0] \\ &= \Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_i, \rho_i)) = 1 \\ \text{conditioned on} \\ X_{i-1} = 0 \end{array} \middle| \begin{array}{l} ((\Pi_j)_{j \in [i]}, (\rho_j)_{j \in [i-1]}) \leftarrow \tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i) \\ \rho_i \leftarrow \{0, 1\}^{r_i} \end{array} \right] \\ &\leq \kappa_{\text{IOP},i}^{\text{rbr}}(\mathbf{x}). \end{aligned}$$

Then, for every $i \in [k]$ we can write

$$\begin{aligned} & \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_i = 1] \\ &= \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_i = 1 \mid X_{i-1} = 0] \cdot \Pr[X_{i-1} = 0] \\ &\quad + \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_i = 1 \mid X_{i-1} = 1] \cdot \Pr[X_{i-1} = 1] \\ &\leq \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_i = 1 \mid X_{i-1} = 0] + \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \mid X_{i-1} = 1] \cdot \Pr[X_{i-1} = 1] \end{aligned}$$

$$= \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_i = 1 \mid X_{i-1} = 0] + \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_{i-1} = 1].$$

Applying the above for every $i \in [k]$ we get

$$\begin{aligned} & \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_k = 1] \\ & \leq \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_k = 1 \mid X_{k-1} = 0] + \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_{k-1} = 1] \\ & \leq \dots \\ & \leq \sum_{i \in [k]} \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_i = 1 \mid X_{i-1} = 0] \\ & \leq \sum_{i \in [k]} \kappa_{\text{IOP}, i}^{\text{rbr}}(\mathbf{x}) \\ & \leq k \cdot \kappa_{\text{IOP}}^{\text{rbr}}(\mathbf{x}). \end{aligned}$$

□

31.2 RBR soundness implies SR soundness

The theorem below states that the state-restoration soundness error of a public-coin IOP with round complexity k is at most a multiplicative factor of $t + k$ larger than its round-by-round soundness error.

Theorem 31.2.1. *Let $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ be a public-coin IOP. If IOP has round-by-round soundness error $\epsilon_{\text{IOP}}^{\text{rbr}}$ then IOP has state-restoration soundness error $\epsilon_{\text{IOP}}^{\text{sr}}$ (see Definition 23.2.2) such that*

$$\epsilon_{\text{IOP}}^{\text{sr}}(s, t, n) \leq (t + k) \cdot \epsilon_{\text{IOP}}^{\text{rbr}}(n).$$

Proof. Fix a salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$, and instance size bound $n \in \mathbb{N}$. We wish to upper bound the following probability:

$$\Pr \left[\mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \mid \begin{array}{l} \mathsf{rnd} = (\mathsf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathsf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\mathsf{tr}^{\text{sr}}} \\ \mathsf{Game}_{\text{IOP}}^{\text{sr}}(s, \mathsf{rnd}, \mathbf{P}_{\text{IOP}}^{\text{sr}}) \end{array} \right].$$

The final output of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ is a move of the form $(\mathbf{x}, (\Pi_1, \dots, \Pi_k), (\sigma_1, \dots, \sigma_k))$, which may or may not appear as a move in tr^{sr} . Define $\mathsf{tr}_{\text{full}}^{\text{sr}}$ to be the concatenation of the trace tr^{sr} and the move-response pairs

$$\left((\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i)), \mathsf{rnd}_i(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i)) \right)_{i \in [k]},$$

namely the k moves obtained from every prefix of $(\mathbf{x}, (\Pi_1, \dots, \Pi_k), (\sigma_1, \dots, \sigma_k))$ along with their answers according to $\mathsf{rnd} = (\mathsf{rnd}_i)_{i \in [k]}$. The trace $\mathsf{tr}_{\text{full}}^{\text{sr}}$ contains at most $t + k$ move-response pairs.

We say that $\mathsf{tr}_{\text{full}}^{\text{sr}}$ *fully contains* a move $(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i))$ if for every $j \in [i]$ the move $(\mathbf{x}, (\Pi_1, \dots, \Pi_j), (\sigma_1, \dots, \sigma_j))$ is contained in $\mathsf{tr}_{\text{full}}^{\text{sr}}$. Let I_i be the set of round- i moves that are fully contained in $\mathsf{tr}_{\text{full}}^{\text{sr}}$. Note that $\sum_{i \in [k]} |I_i| \leq t + k$, since $(I_i)_{i \in [k]}$ are pairwise disjoint and are in $\mathsf{tr}_{\text{full}}^{\text{sr}}$.

For every $i \in [k]$, let X_i be the indicator random variable for the event that $\text{tr}_{\text{full}}^{\text{sr}}$ contains a move $(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i)) \in I_i$ such that

$$\text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_i, \rho_i)) = 1,$$

where $\rho_j := \text{rnd}_j(\mathbf{x}, (\Pi_1, \dots, \Pi_j), (\sigma_1, \dots, \sigma_j))$ for every $j \in [i]$.

If the IOP state-restoration prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ wins the state-restoration game then $X_k = 1$, and thus it suffices to upper bound the probability that $X_k = 1$. Indeed, if $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ wins then the IOP verifier accepts the transcript induced by the final output of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$, which in turn implies that the state function outputs 1 when given transcript (which is fully contained in $\text{tr}_{\text{full}}^{\text{sr}}$).

Define $X_0 := 0$. For every $i \in [k]$, we upper bound $\Pr[X_i = 1 | X_{i-1} = 0]$. Fix a move $(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i)) \in I_i$, which implies that $(\mathbf{x}, (\Pi_1, \dots, \Pi_{i-1}), (\sigma_1, \dots, \sigma_{i-1})) \in I_{i-1}$. Since additionally we assumed that $X_{i-1} = 0$ we get that $\text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_{i-1}, \rho_{i-1})) = 0$ where $\rho_j := \text{rnd}_j(\mathbf{x}, (\Pi_1, \dots, \Pi_j), (\sigma_1, \dots, \sigma_j))$ for every $j \in [i-1]$.

Assume, without loss of generality, that $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ does not make duplicate moves. For every $a \in [t]$, consider the round-by-round prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i, a, \text{rnd})$ that outputs the partial transcript induced by the a -th move among all the moves of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ for round i .

$\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i, a, \text{rnd})$:

1. Set $c := 0$.
2. Simulate the execution of $\text{Game}_{\text{IOP}}^{\text{sr}}(s, \text{rnd}, \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}})$. Whenever $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ outputs a move of the form $(\mathbf{x}, (\Pi_1, \dots, \Pi_{i'}), (\sigma_1, \dots, \sigma_{i'}))$ for a round $i' \in [k]$:
 - a) If $i' = i$ and $(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i)) \in I_i$ then set $c := c + 1$.
 - b) If $c = a$ then:
 - i. For every $j \in [i-1]$, set $\rho_j := \text{rnd}_j(\mathbf{x}, (\Pi_1, \dots, \Pi_j), (\sigma_1, \dots, \sigma_j))$.
 - ii. Output $((\Pi_j)_{j \in [i]}, (\rho_j)_{j \in [i-1]})$ (and halt).
 - c) If $c < a$ then answer with $\rho_{i'} := \text{rnd}_{i'}(\mathbf{x}, (\Pi_1, \dots, \Pi_{i'}), (\sigma_1, \dots, \sigma_{i'}))$.
3. Output \perp .

Since $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ does not make duplicate moves, if $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i, a, \text{rnd})$ outputs $((\Pi_j)_{j \in [i]}, (\rho_j)_{j \in [i-1]})$ (that does not equal \perp) then $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i, a, \text{rnd})$ has not answered the move $(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i))$, which means that $((\Pi_j)_{j \in [i]}, (\rho_j)_{j \in [i-1]})$ is independent of $\rho_i := \text{rnd}_i(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i))$. Hence, by round-by-round soundness of the IOP,

$$\Pr \left[\begin{array}{l} \text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_i, \rho_i)) = 1 \\ \text{conditioned on} \\ X_{i-1} = 0 \end{array} \middle| \begin{array}{l} \text{rnd} = (\text{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ ((\Pi_j)_{j \in [i]}, (\rho_j)_{j \in [i-1]}) \leftarrow \tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i, a, \text{rnd}) \\ \rho_i \leftarrow \{0, 1\}^{r_i} \end{array} \right] \leq \epsilon_{\text{IOP}}^{\text{rbr}}(n).$$

Moreover, if $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i, a, \text{rnd})$ outputs \perp then the above probability is 0. If $X_i = 1$ then for some $a \in [t]$ it must hold that the output of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i, a, \text{rnd})$ satisfies the event in the probability statement above. Thus, taking a union bound over all moves in I_i , we get that

$$\Pr[X_i = 1 | X_{i-1} = 0] \leq |I_i| \cdot \epsilon_{\text{IOP}}^{\text{rbr}}(n).$$

Then for every $i \in [k]$ we can write

$$\Pr[X_i = 1]$$

$$\begin{aligned}
&= \Pr[X_i = 1 \mid X_{i-1} = 0] \cdot \Pr[X_{i-1} = 0] + \Pr[X_i = 1 \mid X_{i-1} = 1] \cdot \Pr[X_{i-1} = 1] \\
&\leq \Pr[X_i = 1 \mid X_{i-1} = 0] + \Pr[X_{i-1} = 1].
\end{aligned}$$

Applying the above for every $i \in [k]$ we get

$$\begin{aligned}
&\Pr[X_k = 1] \\
&\leq \Pr[X_k = 1 \mid X_{k-1} = 0] + \Pr[X_{k-1} = 1] \\
&\leq \dots \\
&\leq \sum_{i \in [k]} \Pr[X_i = 1 \mid X_{i-1} = 0] \\
&\leq \sum_{i \in [k]} |I_i| \cdot \epsilon_{\text{IOP}}^{\text{rbr}}(n) \\
&\leq (t+k) \cdot \epsilon_{\text{IOP}}^{\text{rbr}}(n).
\end{aligned}$$

□

31.3 RBR knowledge soundness implies SR knowledge soundness

The theorem below about knowledge soundness is analogous to Theorem 31.2.1 (which is about soundness): it states that the state-restoration knowledge soundness error of a public-coin IOP with round complexity k is at most a multiplicative factor of $t+k$ larger than its round-by-round knowledge soundness error.

Theorem 31.3.1. *Let $\text{IOP} = (\mathbf{P}_{\text{IOP}}, \mathbf{V}_{\text{IOP}})$ be a public-coin IP with round complexity k . If IOP has round-by-round knowledge soundness error $\epsilon_{\text{IOP}}^{\text{rbr}}$ then IOP has straightline state-restoration knowledge soundness error $\kappa_{\text{IOP}}^{\text{sr}}$ (see Definition 23.2.3) such that*

$$\kappa_{\text{IOP}}^{\text{sr}}(s, t, n) \leq (t+k) \cdot \epsilon_{\text{IOP}}^{\text{rbr}}(n).$$

Construction 31.3.2. Let $\mathbf{E}_{\text{IOP}}^{\text{rbr}}$ be the round-by-round extractor for IOP . We construct an extractor $\mathbf{E}_{\text{IOP}}^{\text{sr}}$ that, given as input an instance \mathbf{x} , IOP strings $(\Pi_i)_{i \in [k]}$, salts $(\sigma_i)_{i \in [k]}$, and move-response trace tr^{sr} , works as follows.

$\mathbf{E}_{\text{IOP}}^{\text{sr}}(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, \text{tr}^{\text{sr}})$:

1. Compute $\mathbf{w} := \mathbf{E}_{\text{IOP}}^{\text{rbr}}(\mathbf{x}, (\Pi_i)_{i \in [k]})$.
2. Output \mathbf{w} .

Proof. Fix a salt size $s \in \mathbb{N}$, move budget $t \in \mathbb{N}$, t -move malicious prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$, and instance size bound $n \in \mathbb{N}$. We wish to upper bound the following probability:

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \end{array} \middle| \begin{array}{l} \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \\ \text{Game}_{\text{IOP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IOP}}^{\text{sr}}(\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, \text{tr}^{\text{sr}}) \end{array} \right].$$

By construction of $\mathbf{E}_{\text{IOP}}^{\text{sr}}$, the above is equivalent to the following

$$\Pr \left[\begin{array}{l} |\mathbf{x}| \leq n \\ \wedge (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \wedge \mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1 \end{array} \middle| \begin{array}{l} \mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]} \leftarrow \mathcal{U}((\mathbf{r}_i)_{i \in [k]}) \\ (\mathbf{x}, (\Pi_i)_{i \in [k]}, (\sigma_i)_{i \in [k]}, (\rho_i)_{i \in [k]}) \xleftarrow{\text{tr}^{\text{sr}}} \\ \mathbf{Game}_{\text{IOP}}^{\text{sr}}(s, \mathbf{rnd}, \mathbf{P}_{\text{IOP}}^{\text{sr}}) \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IOP}}^{\text{rbr}}(\mathbf{x}, (\Pi_i)_{i \in [k]}) \end{array} \right].$$

Define $\text{tr}_{\text{full}}^{\text{sr}}$ to be the concatenation of the trace tr^{sr} and the move-response pairs

$$\left((\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i)), \mathbf{rnd}_i(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i)) \right)_{i \in [k]},$$

namely the k moves obtained from every prefix of $(\mathbf{x}, (\Pi_1, \dots, \Pi_k), (\sigma_1, \dots, \sigma_k))$ along with their answers according to $\mathbf{rnd} = (\mathbf{rnd}_i)_{i \in [k]}$. The trace $\text{tr}_{\text{full}}^{\text{sr}}$ contains at most $t + k$ move-response pairs.

We say that $\text{tr}_{\text{full}}^{\text{sr}}$ *fully contains* a move $(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i))$ if for every $j \in [i]$ the move $(\mathbf{x}, (\Pi_1, \dots, \Pi_j), (\sigma_1, \dots, \sigma_j))$ is contained in $\text{tr}_{\text{full}}^{\text{sr}}$. Let I_i be the set of round- i moves that are fully contained in $\text{tr}_{\text{full}}^{\text{sr}}$. Note that $\sum_{i \in [k]} |I_i| \leq t + k$, since $(I_i)_{i \in [k]}$ are pairwise disjoint and are in $\text{tr}_{\text{full}}^{\text{sr}}$.

For every $i \in [k]$, let X_i be the indicator random variable for the event that $\text{tr}_{\text{full}}^{\text{sr}}$ contains a move $(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i)) \in I_i$ such that

$$\text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_i, \rho_i)) = 1,$$

where $\rho_j := \mathbf{rnd}_j(\mathbf{x}, (\Pi_1, \dots, \Pi_j), (\sigma_1, \dots, \sigma_j))$ for every $j \in [i]$.

If $\mathbf{V}_{\text{IOP}}^{(\Pi_i)_{i \in [k]}}(\mathbf{x}, (\rho_i)_{i \in [k]}) = 1$ then $\text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_k, \rho_k)) = 1$, which implies that $X_k = 1$ (since $(\Pi_1, \rho_1, \dots, \Pi_k, \rho_k)$ is fully contained in tr^{sr}). Hence it suffices to upper bound the probability $\Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_k = 1]$.

Define $X_0 := 0$. For every $i \in [k]$, we upper bound

$$\Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_i = 1 \mid X_{i-1} = 0].$$

Fix a move $(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i)) \in I_i$, so $(\mathbf{x}, (\Pi_1, \dots, \Pi_{i-1}), (\sigma_1, \dots, \sigma_{i-1})) \in I_{i-1}$. Since additionally we assumed that $X_{i-1} = 0$ we get that $\text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_{i-1}, \rho_{i-1})) = 0$ where $\rho_j := \mathbf{rnd}_j(\mathbf{x}, (\Pi_1, \dots, \Pi_j), (\sigma_1, \dots, \sigma_j))$ for every $j \in [i-1]$.

Assume, without loss of generality, that $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ does not make duplicate moves. For every $a \in [t]$, consider the round-by-round prover $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i, a, \mathbf{rnd})$ that outputs the partial transcript induced by the a -th move among all the moves of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ for round i .

$\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i, a, \mathbf{rnd})$:

1. Set $c := 0$.
2. Simulate the execution of $\mathbf{Game}_{\text{IOP}}^{\text{sr}}(s, \mathbf{rnd}, \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}})$. Whenever $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ outputs a move of the form $(\mathbf{x}, (\Pi_1, \dots, \Pi_{i'}), (\sigma_1, \dots, \sigma_{i'}))$ for a round $i' \in [k]$:
 - a) If $i' = i$ and $(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i)) \in I_i$ then set $c := c + 1$.
 - b) If $c = a$ then:
 - i. For every $j \in [i-1]$, set $\rho_j := \mathbf{rnd}_j(\mathbf{x}, (\Pi_1, \dots, \Pi_j), (\sigma_1, \dots, \sigma_j))$.
 - ii. Output $((\Pi_j)_{j \in [i]}, (\rho_j)_{j \in [i-1]})$ (and halt).
 - c) If $c < a$ then answer with $\rho_{i'} := \mathbf{rnd}_{i'}(\mathbf{x}, (\Pi_1, \dots, \Pi_{i'}), (\sigma_1, \dots, \sigma_{i'}))$.

3. Output \perp .

Since $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ does not make duplicate moves, if $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i, a, \text{rnd})$ outputs $((\Pi_j)_{j \in [i]}, (\rho_j)_{j \in [i-1]})$ (that does not equal \perp) then $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i, a, \text{rnd})$ has not answered the move $(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i))$, which means that $((\Pi_j)_{j \in [i]}, (\rho_j)_{j \in [i-1]})$ is independent of $\rho_i := \text{rnd}_i(\mathbf{x}, (\Pi_1, \dots, \Pi_i), (\sigma_1, \dots, \sigma_i))$. Hence, by round-by-round knowledge soundness of the IOP,

$$\Pr \left[\begin{array}{l} (\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \text{RBRState}(\mathbf{x}, (\Pi_1, \rho_1, \dots, \Pi_i, \rho_i)) = 1 \\ \text{conditioned on} \\ X_{i-1} = 0 \end{array} \middle| \begin{array}{l} \text{rnd} = (\text{rnd}_i)_{i \in [\mathbf{k}]} \leftarrow \mathcal{U}((r_i)_{i \in [\mathbf{k}]}) \\ ((\Pi_j)_{j \in [i]}, (\rho_j)_{j \in [i-1]}) \leftarrow \tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i, a, \text{rnd}) \\ \rho_i \leftarrow \{0, 1\}^{r_i} \\ \mathbf{w} \leftarrow \mathbf{E}_{\text{IOP}}^{\text{sr}}(\mathbf{x}, (\Pi_i)_{i \in [\mathbf{k}]}, (\sigma_i)_{i \in [\mathbf{k}]}, \mathbf{tr}^{\text{sr}}) \end{array} \right] \leq \kappa_{\text{IOP}, i}^{\text{rbr}}(n).$$

Moreover, if $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i, a, \text{rnd})$ outputs \perp then the above probability is 0. If $X_i = 1$ then for some $a \in [t]$ it must hold that the output of $\tilde{\mathbf{P}}_{\text{IOP}}^{\text{rbr}}(i, a, \text{rnd})$ satisfies the event in the probability statement above. Taking a union bound over all moves in I_i , we get that

$$\Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_i = 1 \mid X_{i-1} = 0] \leq |I_i| \cdot \kappa_{\text{IOP}, i}^{\text{rbr}}(n).$$

Then, for every $i \in [\mathbf{k}]$ we can write

$$\begin{aligned} & \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_i = 1] \\ &= \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_i = 1 \mid X_{i-1} = 0] \cdot \Pr[X_{i-1} = 0] \\ &\quad + \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_i = 1 \mid X_{i-1} = 1] \cdot \Pr[X_{i-1} = 1] \\ &\leq \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_i = 1 \mid X_{i-1} = 0] + \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \mid X_{i-1} = 1] \cdot \Pr[X_{i-1} = 1] \\ &= \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_i = 1 \mid X_{i-1} = 0] + \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_{i-1} = 1]. \end{aligned}$$

Applying the above for every $i \in [\mathbf{k}]$ we get

$$\begin{aligned} & \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_{\mathbf{k}} = 1] \\ &\leq \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_{\mathbf{k}} = 1 \mid X_{\mathbf{k}-1} = 0] + \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_{\mathbf{k}-1} = 1] \\ &\leq \dots \\ &\leq \sum_{i \in [\mathbf{k}]} \Pr[(\mathbf{x}, \mathbf{w}) \notin \mathcal{R} \wedge X_i = 1 \mid X_{i-1} = 0] \\ &\leq \sum_{i \in [\mathbf{k}]} |I_i| \cdot \kappa_{\text{IOP}, i}^{\text{rbr}}(n) \\ &\leq (t + \mathbf{k}) \cdot \kappa_{\text{IOP}}^{\text{rbr}}(n). \end{aligned}$$

□

Error bounds

We summarize the main error bounds proved in this book. Table 1 contains error bounds for commitment schemes. Table 2 and Table 3 contain error bounds for interactive arguments and non-interactive arguments respectively. Table 4 and Table 5 contain error bounds obtained from special soundness and round-by-round soundness respectively.

commitment scheme	binding	single-extractability	multi-extractability	hiding
Chapter 17: CM [σ, ℓ, s]	Lemma 17.1.1: $\frac{1}{2} \cdot \frac{t^2}{2^\sigma}$	Lemma 17.2.1: $\frac{1}{2} \cdot \frac{t^2}{2^\sigma}$	Lemma 17.2.2: $\frac{1}{2} \cdot \frac{t^2}{2^\sigma} + \frac{n \cdot t}{2^\sigma}$	Lemma 17.3.1 (for $t < \infty$): $\frac{t}{2^s}$
Chapter 18: MT [σ, Σ, ℓ, s]	Lemma 18.4.1: $\frac{1}{2} \cdot \frac{t^2}{2^\sigma}$	Lemma 18.5.1: $\kappa_{\text{MT}}(\sigma, t, \ell)$ $\leq \frac{1}{2} \cdot \frac{t^2}{2^\sigma}$	Lemma 18.5.6: $\kappa_{\text{MT}}(\sigma, t, \ell, n, k)$ $\leq 2 \cdot \frac{t^2}{2^\sigma}$	Lemma 18.6.1 (for $t < \infty$): $z_{\text{rt}}(\sigma, \ell, s, t) \leq \frac{\ell \cdot t}{2^s} + \frac{\ell \cdot t}{2^{2\sigma}}$ Lemma 18.6.3 (for $t < \infty$) & Remark 18.6.9: $z_{\text{MT}}(\sigma, \ell, s, q, t) \leq \frac{q \cdot \ell \cdot t}{2^s} + \frac{q \cdot \ell \cdot t}{2^{2\sigma}}$

Table 1: Summary of error bounds for commitment schemes.

IARG	$\epsilon_{\text{ARG}}(\lambda, t, n)$
Construction 20.1.1: Kilian [PCP, λ, s]	Theorem 20.3.1: $\epsilon_{\text{PCP}}(n) + \kappa_{\text{MT}}(\lambda, t, l)$
Construction 24.1.1: iBCS [IOP, λ, s]	Theorem 24.3.1: $\epsilon_{\text{IOP}}(n) + \kappa_{\text{MT}}(\lambda, t, l, k, k)$

Table 2: Summary of error bounds for IARGs.

NARG	$\epsilon_{\text{ARG}}(\lambda, t, n)$	$\kappa_{\text{ARG}}(\lambda, t, n)$	$z_{\text{ARG}}(\lambda, t, n)$
Construction 10.1.1: FS _{SP} [SP, λ, s]	Theorem 10.2.1: $(t+1) \cdot \epsilon_{\text{SP}}(n)$ Lemma 12.2.1: $\epsilon_{\text{SP}}^{\text{sr}}(s, t, n)$	Theorem 11.1.1: $(t+1) \cdot \kappa_{\text{SP}}(n)$ Lemma 12.2.2: $\kappa_{\text{SP}}^{\text{sr}}(s, t, n)$	Theorem 11.2.1: $z_{\text{SP}}(n) + \frac{t}{2^s}$
Construction 14.1.1: FS _{IP} [IP, λ, s]	Theorem 14.3.1: $\epsilon_{\text{IP}}^{\text{sr}}(s, t, n)$	Theorem 14.4.1: $\kappa_{\text{IP}}^{\text{sr}}(s, t, n)$	
Construction 15.1.1: FS _{IP} [*] [IP, λ, s]	Theorem 15.2.1: $\epsilon_{\text{IP}}^{\text{sr}}(s, t, n) + \frac{t^2}{2^\lambda}$	Theorem 16.1.1: $\kappa_{\text{IP}}^{\text{sr}}(s, t, n) + \frac{t^2}{2^\lambda}$	Theorem 16.2.1: $z_{\text{IP}}(n) + \frac{t}{2^s}$
Construction 21.1.1: Micali [PCP, λ, s]	Theorem 21.2.1: $(t+1) \cdot \epsilon_{\text{PCP}}(n) + \kappa_{\text{MT}}(\lambda, t, l, t+1, 1)$ Claim 21.2.3: $\epsilon_{\text{PCP}}^{\text{sr}}(\lambda+s, t, n) + \kappa_{\text{MT}}(\lambda, t, l, t+1, 1)$	Theorem 22.1.1: $(t+1) \cdot \kappa_{\text{PCP}}(n) + \kappa_{\text{MT}}(\lambda, t, l, t+1, 1)$ Claim 21.2.3: $\kappa_{\text{PCP}}^{\text{sr}}(\lambda+s, t, n) + \kappa_{\text{MT}}(\lambda, t, l, t+1, 1)$	Theorem 22.2.1: $z_{\text{PCP}}(n) + z_{\text{MT}}(\lambda, l(n), s, q(n), t) + \frac{t}{2^s}$
Construction 25.1.1: BCS [IOP, λ, s]	Theorem 25.2.1: $\epsilon_{\text{IOP}}^{\text{sr}}(\lambda+s, t, n) + \kappa_{\text{MT}}(\lambda, t, l, (t+1)\cdot k, k) + \frac{t^2}{2^\lambda}$	Theorem 26.1.1: $\kappa_{\text{IOP}}^{\text{sr}}(\lambda+s, t, n) + \kappa_{\text{MT}}(\lambda, t, l, (t+1)\cdot k, k) + \frac{t^2}{2^\lambda}$	Theorem 26.2.1: $z_{\text{IOP}}(n) + z_{\text{MT}}(\lambda, l(n), s, q(n), t) + \frac{t}{2^s}$

Table 3: Summary of error bounds for NARGs. For simplicity, for knowledge soundness we display only the error bounds for the case of straightline knowledge soundness. The case of rewinding knowledge soundness can be found in the theorem statements. The gray bounds refer to tighter error bounds in terms of state-restoration soundness, which are upper bounded by the black bound (which is in terms of standard soundness).

	standard	state-restoration
Definition 31.1.2: round-by-round soundness	Claim 31.1.3: $\epsilon_{\text{IOP}}(\mathbf{x}) \leq k \cdot \epsilon_{\text{IOP}}^{\text{rbr}}(\mathbf{x})$	Theorem 31.2.1: $\epsilon_{\text{IOP}}^{\text{sr}}(s, t, n) \leq (t + k) \cdot \epsilon_{\text{IOP}}^{\text{rbr}}(n)$
Definition 31.1.4: round-by-round knowledge soundness	Claim 31.1.5: $\kappa_{\text{IOP}}(\mathbf{x}) \leq k \cdot \kappa_{\text{IOP}}^{\text{rbr}}(\mathbf{x})$	Theorem 31.3.1: $\kappa_{\text{IOP}}^{\text{sr}}(s, t, n) \leq (t + k) \cdot \kappa_{\text{IOP}}^{\text{rbr}}(n)$

Table 4: Summary of error bounds from round-by-round soundness and knowledge soundness.

	standard	state-restoration
Definition 30.1.2: special soundness with arity a	Theorem 30.2.1: $\kappa_{\text{SP}}(\mathbf{x}, \delta_{\tilde{\mathbf{P}}_{\text{SP}}}) \leq \frac{a-1}{2^r}$	Theorem 30.3.1: $\kappa_{\text{SP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}}) \leq (t+1) \cdot \frac{a-1}{2^r}$
Definition 30.1.5: special soundness with arity (a_1, \dots, a_k)	Theorem 30.4.1: $\kappa_{\text{IP}}(\mathbf{x}) \leq 1 - \prod_{i \in [k]} \left(1 - \frac{a_i-1}{2^{r_i}}\right)$	Theorem 30.5.1: $\kappa_{\text{IP}}^{\text{sr}}(s, t, n, \delta_{\tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}}) \leq (t+1) \cdot \left(1 - \prod_{i \in [k]} \left(1 - \frac{a_i-1}{2^{r_i}}\right)\right)$

Table 5: Summary of error bounds from special soundness.

List of figures

1	Parts of this book.	xiv
2	Transformations in this book.	xiv
1.1	Pairwise and three-wise interaction of 3 events.	5
4.1	Diagram of a NARG and an IARG.	37
8.1	Diagram of an SP.	74
9.1	Diagram of Construction 9.1.1.	80
10.1	Diagram of the FS transformation for SPs.	91
13.1	Diagram of an IP.	111
14.1	Diagram of the FS transformation for public-coin IPs.	124
15.1	Diagram of the fast FS transformation for public-coin IPs.	134
17.1	Diagram of the basic commitment scheme CM.	153
18.1	The binary tree graph T_8 .	168
18.2	Computation of MT.Commit for $\ell = 8$.	170
18.3	Collision between two authentication paths.	173
19.1	Diagram of a PCP.	199
20.1	Diagram of the Kilian transformation.	209
21.1	Diagram of the Micali transformation.	218
23.1	Diagram of an IOP.	233
24.1	Diagram of the iBCS transformation.	242
25.1	Diagram of the BCS transformation.	252
28.1	Common error bound types.	282
28.2	Two different error bound functions.	283
28.3	Two definitions for security level.	284
29.1	Trees for message length ℓ from 1 to 8.	295
29.2	Authenticating $I = \{5, 6\}$ in the tree T_3 .	298
29.3	Authenticating $I = \{3, 6\}$ in the tree T_3 .	298
29.4	Authenticating $I = \{1, 3, 5, 6\}$ in the tree T_3 .	298

29.5 Example of path pruning for $\ell = 16$ and $I = \{1, 2, 3, 4\}$	303
29.6 Example of path pruning for $\ell = 16$ and $I = \{2, 7, 12, 14\}$	303
29.7 Graphs reporting Merkle opening proof sizes.	304
30.1 Diagram of an a -fork for an SP.	309
30.2 Diagram of an (a_1, a_2) -tree of transcripts for a public-coin IP.	310

List of tables

1	Error bounds for commitment schemes.	360
2	Error bounds for IARGs.	360
3	Error bounds for NARGs.	361
4	Error bounds from RBR soundness and knowledge soundness.	362
5	Error bounds from special soundness.	362

Glossary

Acronyms

- ROM random oracle model (see Chapter 2)
- NARG non-interactive argument (see Section 4.1)
- IARG interactive argument (see Section 4.2)
- SNARG succinct non-interactive argument (see Section 4.4)
- SP sigma protocol (see Chapter 8)
- IP interactive proof (see Chapter 13)
- PCP probabilistically checkable proof (see Chapter 19)
- IOP interactive oracle proof (see Chapter 23)

Symbols

- X, Y, Z random variables
- $\Delta(X, Y)$ statistical distance between X and Y
- E probability event
- $\mathcal{R}, \mathcal{L}(\mathcal{R})$ relation and its language
- \mathbf{x}, \mathbf{w} instance and witness
- n instance size bound
- s privacy parameter
- τ salt
- aux intermediate state of a stateful algorithm
- $A(\underline{B})$ the algorithm A uses the algorithm B as a black box

Random oracle

- σ output size of a random oracle
- $\mathcal{U}(\sigma)$ uniform distribution over random oracles
- f random oracle
- \dot{f} lazily sampled random oracle
- t bound on the number of queries to the random oracle
- tr query-answer trace

Argument systems (Chapter 4)

- NARG tuple specifying a non-interactive argument
- IARG tuple specifying an interactive argument

- \mathcal{P}, \mathcal{V} argument prover and argument verifier
- $q_{\mathcal{P}}, q_{\mathcal{V}}$ number of queries to the random oracle by \mathcal{P} and \mathcal{V}
- $c_{\mathcal{P}}, c_{\mathcal{V}}, c$ bits sent by \mathcal{P} , sent by \mathcal{V} , and their sum
- π, s argument string and argument string size (for non-interactive arguments)
- ϵ_{ARG} soundness error of an argument
- κ_{ARG} knowledge soundness error of an argument
- z_{ARG} zero-knowledge error of an argument
- \mathcal{E} knowledge extractor of an argument
- \mathcal{S} zero-knowledge simulator of an argument
- et_{ARG} extraction time of an argument extractor

Probabilistic proofs

- $\text{SP}, \text{IP}, \text{PCP}, \text{IOP}$ tuple specifying an SP, IP, PCP, IOP
- $\mathbf{P}_{\text{SP}}, \mathbf{P}_{\text{IP}}, \mathbf{P}_{\text{PCP}}, \mathbf{P}_{\text{IOP}}$ prover of an SP, IP, PCP, IOP
- $\mathbf{V}_{\text{SP}}, \mathbf{V}_{\text{IP}}, \mathbf{V}_{\text{PCP}}, \mathbf{V}_{\text{IOP}}$ verifier of an SP, IP, PCP, IOP
- $\epsilon_{\text{SP}}, \epsilon_{\text{IP}}, \epsilon_{\text{PCP}}, \epsilon_{\text{IOP}}$ soundness error of an SP, IP, PCP, IOP
- $\kappa_{\text{SP}}, \kappa_{\text{IP}}, \kappa_{\text{PCP}}, \kappa_{\text{IOP}}$ knowledge soundness error of an SP, IP, PCP, IOP
- $z_{\text{SP}}, z_{\text{IP}}, z_{\text{PCP}}, z_{\text{IOP}}$ honest-verifier zero-knowledge error of an SP, IP, PCP, IOP
- $\mathbf{E}_{\text{SP}}, \mathbf{E}_{\text{IP}}, \mathbf{E}_{\text{PCP}}, \mathbf{E}_{\text{IOP}}$ knowledge extractor of an SP, IP, PCP, IOP
- $\mathbf{S}_{\text{SP}}, \mathbf{S}_{\text{IP}}, \mathbf{S}_{\text{PCP}}, \mathbf{S}_{\text{IOP}}$ zero-knowledge simulator of an SP, IP, PCP, IOP
- $\text{et}_{\text{SP}}, \text{et}_{\text{IP}}, \text{et}_{\text{IOP}}$ extraction time of an SP, IP, IOP extractor

Complexity measures for probabilistic proofs

- k round complexity
- Σ proof alphabet
- l proof length
- q query complexity
- r randomness complexity
- pt prover time
- vt verifier time
- pv prover-to-verifier communication complexity
- vp verifier-to-prover communication complexity

Basic commitment scheme (Chapter 17)

- $\text{CM} = (\text{CM.Commit}, \text{CM.Check})$ tuple defining the scheme
- m, ℓ message and message length
- τ, s salt and salt size
- cm commitment

Merkle commitment scheme (Chapter 18)

- $\text{MT} = (\text{MT.Commit}, \text{MT.Open}, \text{MT.Check})$ tuple defining the scheme
- \mathbf{m}, Σ, ℓ message vector, message alphabet, and message length
- τ, s salt and salt size
- rt Merkle commitment
- td Merkle opening trapdoor

- pf Merkle opening proof
- $T_\ell = (V_\ell, E_\ell)$ tree graph
- $\text{path}, \text{copath}$ path, copath

State restoration

- $\text{Game}_{\text{SP}}^{\text{sr}}, \text{Game}_{\text{IP}}^{\text{sr}}, \text{Game}_{\text{PCP}}^{\text{sr}}, \text{Game}_{\text{IOP}}^{\text{sr}}$ state-restoration game of an SP, IP, PCP, IOP
- rnd randomness of the state-restoration game
- t move budget in a state-restoration game
- tr^{sr} move-response trace of the state-restoration game
- σ salt string in a state-restoration game
- $\tilde{\mathbf{P}}_{\text{SP}}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IP}}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{PCP}}^{\text{sr}}, \tilde{\mathbf{P}}_{\text{IOP}}^{\text{sr}}$ state-restoration malicious prover for an SP, IP, PCP, IOP
- $\epsilon_{\text{SP}}^{\text{sr}}, \epsilon_{\text{IP}}^{\text{sr}}, \epsilon_{\text{PCP}}^{\text{sr}}, \epsilon_{\text{IOP}}^{\text{sr}}$ state-restoration soundness error of an SP, IP, PCP, IOP
- $\kappa_{\text{SP}}^{\text{sr}}, \kappa_{\text{IP}}^{\text{sr}}, \kappa_{\text{PCP}}^{\text{sr}}, \kappa_{\text{IOP}}^{\text{sr}}$ state-restoration knowledge soundness error of an SP, IP, PCP, IOP
- $\mathbf{E}_{\text{SP}}^{\text{sr}}, \mathbf{E}_{\text{IP}}^{\text{sr}}, \mathbf{E}_{\text{PCP}}^{\text{sr}}, \mathbf{E}_{\text{IOP}}^{\text{sr}}$ state-restoration knowledge extractor of an SP, IP, PCP, IOP
- $\text{et}_{\text{SP}}^{\text{sr}}, \text{et}_{\text{IP}}^{\text{sr}}, \text{et}_{\text{IOP}}^{\text{sr}}$ extraction time of an SP, IP, IOP state-restoration extractor

Bibliographic notes

The material in this book systematizes and extends results in the cryptography literature, and also contributes new results. Below, we summarize the main research papers related to the material in this book. Where relevant, we explain how our exposition and results differ from the literature.

Random oracle model The notion of a random oracle was used as early as [BG81] in complexity theory, in the context of relativized complexity classes. Random oracles were then used in cryptography, e.g., to achieve new constructions [Bra81] and to establish limitations [IR89]. The random oracle model (ROM) was most prominently defined and advocated for cryptography in [BR93]. The fact that security in the ROM does not, in general, imply security when the random oracle is replaced by an efficient hash function was proved in [CGH04].

Probabilistic proofs We consider several types of probabilistic proofs:

- sigma protocols (SPs), 3-message interactive proofs that appeared as early as [Sch89; Cra97];
- interactive proofs (IPs), introduced in [GMR89; Bab85];
- probabilistically checkable proofs (PCPs), introduced in [BFLS91; FGLSS96; AS98; ALMSS98];
- interactive oracle proofs (IOPs), introduced in [BCS16; RRR21].

It is *not* a goal of this book to discuss constructions of probabilistic proofs. The goal of this book is to explain how to use random oracles to transform probabilistic proofs into arguments.

Commitment schemes We study two types of commitment schemes in the ROM.

- The basic commitment scheme in the ROM is a folklore construction, whose extractability and hiding in the random oracle model was studied, e.g., in [Pas03].
- The Merkle commitment scheme was introduced in [Mer89a]. Merkle commitment schemes are typically used in the setting where the hash function is merely collision resistant, whereas the hash function in the setting that we consider is a random oracle. The extractability properties of Merkle commitment schemes in the ROM were studied in [Val08; BCS16], and their hiding properties in the ROM were studied in [BCS16].

Our treatment of commitment schemes in the ROM in this book (see Part IV and Chapter 29) includes new security definitions, analyses, security bounds, and more.

Transformations We study several constructions of arguments in the ROM.

- The SP-to-NARG transformation in Chapter 10 is due to [FS86].
- The IP-to-NARG transformation in Chapter 14 is a folklore generalization of the above transformation. The optimization in Chapter 15 is also folklore and adopts ideas from the Merkle–Damgård construction [Dam89; Mer89b].
- The PCP-to-IARG transformation in Chapter 20 is due to [Kil92].

- The PCP-to-NARG transformation in Chapter 21 is due to [Mic00].
- The IOP-to-NARG transformation in Chapter 25 is due to [BCS16], and the IOP-to-IARG transformation in Chapter 24 is a straightforward simplification of it.

Our presentation of the transformations at times differs in important details from descriptions in the literature. For example, we use multiple random oracles for distinct purposes (this favorably impacts concrete security and exposition) and we add salts to certain queries (this facilitates properties such as adaptive zero knowledge).

Security definitions Most prior works on SNARGs in the ROM ([Mic00; Val08; BCS16]) study non-adaptive notions of security (e.g., non-adaptive soundness). An exception is [COS20], which studies adaptive soundness and knowledge soundness for the IOP-to-NARG transformation of [BCS16]; they use the generic reduction in Section 6.2, incurring a multiplicative security loss.

In contrast, we directly establish adaptive security for all (interactive and non-interactive) argument systems that we consider without incurring any additional losses (compared to non-adaptive security). In particular, the definitions of soundness, knowledge soundness, and zero knowledge that we use are against adaptive adversaries (see Chapters 4 and 5).

State-restoration soundness All constructions of non-interactive arguments in this book require the underlying probabilistic proof to satisfy a strong notion of soundness known as state-restoration soundness, initially introduced in [BCS16] for IOPs. We provide a coherent treatment of this notion for all probabilistic proofs that we study (SPs, IPs, PCPs, IOPs); this leads to simplifications and harmonizations of the definition to work across these settings. State-restoration soundness serves as a necessary and sufficient soundness notion for the security of the non-interactive arguments that we study, and it is implied by notable strong soundness notions (such as special soundness and round-by-round soundness). This differs from many works in the literature that, instead, directly argue the security of a non-interactive argument starting from stronger soundness notions.

Security reductions All constructions of arguments in the ROM in this book are proved secure via security reductions that significantly improve over the relevant prior literature.

- In some cases, this is because no security reductions were available. For example, we contribute the first analyses in the ROM for the PCP-to-IARG transformation in Chapter 20 and the IOP-to-IARG transformation in Chapter 24. As another example, we contribute the first analysis of the optimization in Chapter 15 of the IP-to-NARG transformation in Chapter 14.
- In other cases, this is because our comprehensive treatment of commitment schemes in the ROM offers new useful building blocks. For example, we contribute security analyses for the PCP-to-NARG transformation in Chapter 21 and IOP-to-NARG transformation in Chapter 25 that are more modular and detailed than prior ones in the literature.

In all cases, the fact that we aim for adaptive security notions imposes additional delicate technical considerations throughout security reductions that are not present in prior literature.

Strong soundness notions We discuss notions of soundness for probabilistic proofs that imply state-restoration soundness (which is necessary and sufficient for the security of the non-interactive arguments that we study).

- *Special soundness.* Special soundness for SPs was introduced in [Sch89; Cra97], and special soundness for (public-coin) IPs was introduced in [BCCGP16]; the definitions of special soundness that we use are standard (see Section 30.1).

In Section 30.2 we show that special soundness for an SP implies rewinding knowledge soundness, and in Section 30.4 we do the same for public-coin IPs. The analyses are adapted from [ACK21] to our definitions of knowledge soundness.

In Section 30.3 we show that special soundness of an SP implies rewinding state-restoration knowledge soundness, and in Section 30.5 we do the same for public-coin IPs. This is similar to the fact that special soundness of an SP or public-coin IP suffices for the knowledge soundness of a non-interactive argument obtained from the SP or public-coin IP via the Fiat–Shamir transformation; this was proved by [PS96; BN06] for SPs and by [AFK22] for public-coin IPs.

Our treatment differs somewhat with prior literature due to the structure of our knowledge soundness definitions. However, the approach is essentially the same as in the citations above.

- *Round-by-round soundness.* The notion of round-by-round soundness in Chapter 31 was introduced for IPs in [CCHLRR18], and for IOPs in [CMS19]. The notion of round-by-round knowledge soundness was introduced in [CMS19]. The relationship between round-by-round soundness and special soundness is studied in [BGTZ23].

The definitions in Chapter 31 are a mild strengthening of those in the literature: the knowledge extractor does not receive as input verifier randomness. This choice has two motivations. First, it is consistent with the notion of knowledge soundness for PCPs (Definition 19.1.3, a standard definition), where the knowledge extractor receives as input the instance and PCP string, but not the PCP verifier randomness. Second, this implies a notion of straightline state-restoration knowledge soundness where the extractor does not receive verifier randomness, which in turn implies a notion of straightline knowledge soundness for non-interactive arguments where the extractor does not receive query access to the random oracle.

State-restoration soundness implies, in certain parameter settings, round-by-round soundness [Hol19]. We do not discuss this “reverse” implication.

Post-quantum security While not discussed in this book, many constructions of cryptographic proofs that we studied are known to be secure against quantum adversaries. The model to consider here is the *quantum random oracle model* (QROM), introduced in [BDFLSZ11]; briefly, adversaries are computationally unbounded and make a bounded number of superposition queries to the random oracle. Security in the ROM does not, in general, imply security in the QROM [BDFLSZ11; YZ22]; nevertheless, cryptographic proofs in this book remain secure in the QROM. For the interested reader, we provide some pointers to the research literature: the Fiat–Shamir transformation applied to a sigma protocol that is special sound is secure in the QROM [DFMS19; LZ19]; the Micali transformation is secure in the QROM [CMS19]; and the BCS transformation applied to an IOP that is round-by-round sound is secure in the QROM [CMS19].

Security in the standard model In this book we restrict our attention to cryptographic proofs in the ROM, i.e., cryptographic proofs based on, and only based on, a hash function that is assumed to be ideal. For some constructions in this book, one can perform a (less efficient) security analysis without assuming that the underlying hash function is ideal.

For example, the Kilian transformation (Chapter 20) and the iBCS transformation (Chapter 24) can be proved secure while only assuming that the underlying hash function is collision-resistant (more generally, replacing the Merkle commitment scheme in the ROM with any vector commitment scheme that is position binding) [Kil92; BG08; IMSX15; CDGS23]; this leads to succinct interactive arguments in the standard model (with no oracles) based on standard cryptographic assumptions. However, in this case the security reduction (for soundness or knowledge soundness) is less efficient because it relies on rewinding the adversary, leading to a concrete security loss compared to the analysis in the ROM. One can view the analysis in the ROM as the one a practitioner may rely upon, while the analysis from collision resistant as the one researchers might use for results in theoretical cryptography.

In contrast, for succinct non-interactive arguments (e.g., for the Micali transformation and BCS transformation), similar security reductions to “falsifiable” assumptions (such as the collision resistance of a hash function) are not possible [GW11]. This limitation explains why succinct non-interactive arguments are typically proved secure with the aid of the ROM, or (and possibly in addition to) other types of (non-falsifiable) assumptions.

Bibliography

- [ACFY24] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev
STIR: Reed–Solomon proximity testing with fewer queries
 Cryptology ePrint Archive, Report 2024/390. 2024.
 URL: <https://eprint.iacr.org/2024/390>.
 (Cited on page 278.)
- [ACK21] Thomas Attema, Ronald Cramer, and Lisa Kohl
A compressed Σ -Protocol theory for lattices
 In Proceedings of CRYPTO 2021 (41st International Cryptology Conference), pp. 549–579.
 URL: <https://eprint.iacr.org/2021/307>.
 (Cited on page 371.)
- [AFK22] Thomas Attema, Serge Fehr, and Michael Kloß
Fiat–Shamir transformation of multi-round interactive proofs
 In Proceedings of TCC 2022 (20th Theory of Cryptography Conference), pp. 113–142.
 URL: <https://eprint.iacr.org/2021/1377>.
 (Cited on page 371.)
- [AH87] William Aiello and Johan Håstad
Perfect zero-knowledge languages can be recognized in two rounds
 In Proceedings of FOCS 1987 (28th Symposium on Foundations of Computer Science), pp. 439–448.
 URL: <https://doi.org/10.1109/SFCS.1987.47>.
 (Cited on page 276.)
- [ALMSS98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy
Proof verification and the hardness of approximation problems
 In Journal of the ACM 45.3 (1998), pp. 501–555.
 URL: <https://doi.org/10.1145/278298.278306>.
 A preliminary version of this article appears in FOCS 1992.
 (Cited on pages 277, 369.)
- [AS03] Sanjeev Arora and Madhu Sudan
Improved low-degree testing and its applications
 In Combinatorica 23.3 (2003), pp. 365–426.
 URL: <https://doi.org/10.1007/s00493-003-0025-0>.
 A preliminary version of this article appears in STOC 1997.
 (Cited on page 277.)
- [AS98] Sanjeev Arora and Shmuel Safra
Probabilistic checking of proofs: a new characterization of NP
 In Journal of the ACM 45.1 (1998), pp. 70–122.
 URL: <https://doi.org/10.1145/273865.273901>.
 A preliminary version of this article appears in FOCS 1992.
 (Cited on pages 277, 369.)

- [Bab85] László Babai
Trading group theory for randomness
In Proceedings of STOC 1985 (17th Symposium on Theory of Computing), pp. 421–429.
URL: <https://doi.org/10.1145/22145.22192>.
(Cited on page 369.)
- [BBPW18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell
Bulletproofs: short proofs for confidential transactions and more
In Proceedings of S&P 2018 (39th IEEE Symposium on Security and Privacy), pp. 315–334.
URL: <https://doi.org/10.1109/SP.2018.00020>.
(Cited on page 279.)
- [BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev
Fast Reed–Solomon interactive oracle proofs of proximity
In Proceedings of ICALP 2018 (45th International Colloquium on Automata, Languages and Programming), 14:1–14:17.
URL: <https://doi.org/10.4230/LIPIcs.ICALP.2018.14>.
(Cited on page 278.)
- [BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev
Scalable zero knowledge with no trusted setup
In Proceedings of CRYPTO 2019 (39th International Cryptology Conference), pp. 733–764.
URL: https://doi.org/10.1007/978-3-030-26954-8_23.
(Cited on page 278.)
- [BCCGP16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit
Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting
In Proceedings of EUROCRYPT 2016 (35th International Conference on Theory and Application of Cryptographic Techniques), pp. 327–357.
URL: https://doi.org/10.1007/978-3-662-49896-5_12.
(Cited on page 371.)
- [BCFGRS17] Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner
Zero knowledge protocols from succinct constraint detection
In Proceedings of TCC 2017 (15th Theory of Cryptography Conference), pp. 172–206.
URL: https://doi.org/10.1007/978-3-319-70503-3_6.
(Cited on page 278.)
- [BCG20] Jonathan Bootle, Alessandro Chiesa, and Jens Groth
Linear-Time arguments with sublinear verification from tensor codes
In Proceedings of TCC 2020 (18th Theory of Cryptography Conference), pp. 19–46.
URL: https://doi.org/10.1007/978-3-030-64378-2_2.
(Cited on page 278.)
- [BCGGHJ17] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen
Linear-Time zero-knowledge proofs for arithmetic circuit satisfiability
In Proceedings of ASIACRYPT 2017 (23rd International Conference on the Theory and Applications of Cryptology and Information Security), pp. 336–365.
URL: https://doi.org/10.1007/978-3-319-70700-6_12.
(Cited on page 278.)

- [BCGRS17] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner
Interactive oracle proofs with constant rate and query complexity
In Proceedings of ICALP 2017 (44th International Colloquium on Automata, Languages and Programming), 40:1–40:15.
URL: <https://doi.org/10.4230/LIPIcs.ICALP.2017.40>.
(Cited on page 278.)
- [BCGV16] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, and Madars Virza
Quasilinear-Size zero knowledge from linear-algebraic PCPs
In Proceedings of TCC 2016-A (13th Theory of Cryptography Conference), pp. 33–64.
URL: https://doi.org/10.1007/978-3-662-49099-0_2.
(Cited on page 278.)
- [BCL22] Jonathan Bootle, Alessandro Chiesa, and Siqi Liu
Zero-Knowledge IOPs with linear-time prover and polylogarithmic-time verifier
In Proceedings of EUROCRYPT 2022 (42nd International Conference on Theory and Application of Cryptographic Techniques), pp. 275–304.
URL: https://doi.org/10.1007/978-3-031-07085-3_10.
(Cited on page 278.)
- [BCRSVW19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward
Aurora: transparent succinct arguments for R1CS
In Proceedings of EUROCRYPT 2019 (38th International Conference on the Theory and Applications of Cryptographic Techniques), pp. 103–128.
URL: https://doi.org/10.1007/978-3-030-17653-2_4.
(Cited on page 278.)
- [BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner
Interactive oracle proofs
In Proceedings of TCC 2016-B (14th Theory of Cryptography Conference), pp. 31–60.
URL: https://doi.org/10.1007/978-3-662-53644-5_2.
(Cited on pages 248, 369, 370.)
- [BDFLSZ11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry
Random oracles in a quantum world
In Proceedings of ASIACRYPT 2011 (17th International Conference on the Theory and Application of Cryptology and Information Security), pp. 41–69.
URL: https://doi.org/10.1007/978-3-642-25385-0_3.
(Cited on page 371.)
- [BFL91] László Babai, Lance Fortnow, and Carsten Lund
Non-Deterministic exponential time has two-prover interactive protocols
In Computational Complexity 1 (1991), pp. 3–40.
URL: <https://doi.org/10.1007/BF01200056>.
A preliminary version of this article appears in FOCS 1990.
(Cited on page 276.)
- [BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy
Checking computations in polylogarithmic time
In Proceedings of STOC 1991 (23rd Symposium on Theory of Computing), pp. 21–32.
URL: <https://doi.org/10.1145/103418.103428>.
(Cited on pages 276, 369.)

- [BG08] Boaz Barak and Oded Goldreich
Universal Arguments and their Applications
In SIAM Journal on Computing 38.5 (2008), pp. 1661–1694.
URL: <https://doi.org/10.1137/070709244>.
A preliminary version of this article appears in CCC '02.
(Cited on page 372.)
- [BG81] Charles H. Bennett and John Gill
Relative to a random oracle A, $P^A \neq NP^A \neq coNP^A$ with probability 1
In SIAM Journal on Computing 10.1 (1981), pp. 96–113.
URL: <https://doi.org/10.1137/0210008>.
(Cited on page 369.)
- [BGGHKMR88] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway
Everything provable is provable in zero-knowledge
In Proceedings of CRYPTO 1988 (8th International Cryptology Conference), pp. 37–56.
URL: https://doi.org/10.1007/0-387-34799-2_4.
(Cited on page 276.)
- [BGHSV05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan
Short PCPs verifiable in polylogarithmic time
In Proceedings of CCC 2005 (20th Conference on Computational Complexity), pp. 120–134.
URL: <https://doi.org/10.1109/CCC.2005.27>.
(Cited on page 277.)
- [BGKTTZ23] Alexander R. Block, Albert Garreta, Jonathan Katz, Justin Thaler, Pratyush Ranjan Tiwari, and Michał Zajac
Fiat-Shamir security of FRI and related SNARKs
In Proceedings of the 29th International Conference on the Theory and Application of Cryptology and Information Security, pp. 3–40.
URL: https://doi.org/10.1007/978-981-99-8724-5%5C_1.
(Cited on page 279.)
- [BGTZ23] Alexander R. Block, Albert Garreta, Pratyush Ranjan Tiwari, and Michał Zajac
On soundness notions for interactive oracle proofs
Cryptology ePrint Archive, Paper 2023/1256. 2023.
URL: <https://eprint.iacr.org/2023/1256>.
(Cited on page 371.)
- [BN06] Mihir Bellare and Gregory Neven
Multi-signatures in the plain public-key model and a general forking lemma
In Proceedings of CCS 2006 (13th Conference on Computer and Communications Security), pp. 390–399.
URL: <https://doi.org/10.1145/1180405.1180453>.
(Cited on page 371.)
- [BR93] Mihir Bellare and Phillip Rogaway
Random oracles are practical: a paradigm for designing efficient protocols
In Proceedings of CCS 1993 (1st Conference on Computer and Communications Security), pp. 62–73.
URL: <https://doi.org/10.1145/168588.168596>.
(Cited on page 369.)

- [Bra81] Gilles Brassard
A time-luck tradeoff in relativized cryptography
In Journal of Computer and System Sciences 22.3 (1981), pp. 280–311.
URL: [https://doi.org/10.1016/0022-0000\(81\)90034-9](https://doi.org/10.1016/0022-0000(81)90034-9).
(Cited on page 369.)
- [BS08] Eli Ben-Sasson and Madhu Sudan
Short PCPs with polylog query complexity
In SIAM Journal on Computing 38.2 (2008), pp. 551–607.
URL: <https://doi.org/10.1137/050646445>.
A preliminary version of this article appears in STOC 2005.
(Cited on page 277.)
- [CCHLRR18] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, and Ron D. Rothblum
Fiat-Shamir from simpler assumptions
Cryptology ePrint Archive, Report 2018/1004. 2018.
URL: <https://eprint.iacr.org/2018/1004>.
(Cited on pages 279, 371.)
- [CDGS23] Alessandro Chiesa, Marcel Dall’Agnol, Ziyi Guan, and Nicholas Spooner
On the Security of Succinct Interactive Arguments from Vector Commitments
Cryptology ePrint Archive, Paper 2023/1737. 2023.
URL: <https://eprint.iacr.org/2023/1737>.
(Cited on page 372.)
- [CG21] Alessandro Chiesa and Tom Gur
Foundations and frontiers of probabilistic proofs
2021.
URL: <https://www.slmath.org/workshops/931>.
(Cited on pages xi, 275.)
- [CG22] Ignacio Cascudo and Emanuele Giunta
On interactive oracle proofs for boolean R1CS statements
In Proceedings of FC 2022 (26th International Conference on Financial Cryptography and Data Security), pp. 230–247.
URL: https://doi.org/10.1007/978-3-031-18283-9_11.
(Cited on page 278.)
- [CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi
The random oracle methodology, revisited
In Journal of the ACM 51.4 (2004), pp. 557–594.
URL: <https://doi.org/10.1145/1008731.1008734>.
A preliminary version of this article appears in STOC 1998.
(Cited on page 369.)
- [Chi23] Alessandro Chiesa
Foundations and frontiers of probabilistic proofs
2023.
URL: <https://www.slmath.org/workshops/1037>.
(Cited on pages xi, 275.)
- [CMS19] Alessandro Chiesa, Peter Manohar, and Nicholas Spooner
Succinct arguments in the quantum random oracle model
In Proceedings of TCC 2019 (17th Theory of Cryptography Conference), pp. 1–29.
URL: https://doi.org/10.1007/978-3-030-36033-7_1.
(Cited on page 371.)

- [COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner
Fractal: post-quantum and transparent recursive proofs from holography
In Proceedings of EUROCRYPT 2020 (39th International Conference on the Theory and Applications of Cryptographic Techniques), pp. 769–793.
URL: https://doi.org/10.1007/978-3-030-45721-1_27.
(Cited on pages 278, 370.)
- [Cra97] Ronald Cramer
Modular design of secure yet practical cryptographic protocols
PhD thesis. University of Amsterdam, 1997.
URL: <https://ir.cwi.nl/pub/21438/21438A.pdf>.
(Cited on pages 369, 371.)
- [Dam89] Ivan Damgård
A design principle for hash functions
In Proceedings of CRYPTO 1989 (9th International Cryptology Conference). Vol. 435, pp. 416–427.
URL: https://doi.org/10.1007/0-387-34805-0_39.
(Cited on page 369.)
- [DFKRS11] Irit Dinur, Eldar Fischer, Guy Kindler, Ran Raz, and Shmuel Safra
PCP characterizations of NP: toward a polynomially-small error-probability
In Computational Complexity 20.3 (2011), pp. 413–504.
URL: <https://doi.org/10.1007/s00037-011-0014-4>.
A preliminary version of this article appears in STOC 1999.
(Cited on page 277.)
- [DFMS19] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner
Security of the Fiat–Shamir transformation in the quantum random-oracle model
In Proceedings of CRYPTO ’19 (39th Annual International Cryptology Conference), pp. 356–383.
URL: https://doi.org/10.1007/978-3-030-26951-7_13.
(Cited on page 371.)
- [DHK15] Irit Dinur, Prahladh Harsha, and Guy Kindler
Polynomially low error PCPs with polyloglog n queries via modular composition
In Proceedings of STOC 2015 (47th Symposium on Theory of Computing), pp. 267–276.
URL: <https://doi.org/10.1145/2746539.2746630>.
(Cited on page 277.)
- [Din07] Irit Dinur
The PCP theorem by gap amplification
In Journal of the ACM 54.3 (2007), 12:1–12:44.
URL: <https://doi.org/10.1145/1236457.1236459>.
A preliminary version of this article appears in STOC 2006.
(Cited on page 277.)
- [FGLSS96] Uriel Feige, Shafi Goldwasser, Laszlo Lovász, Shmuel Safra, and Mario Szegedy
Interactive proofs and the hardness of approximating cliques
In Journal of the ACM 43.2 (1996), pp. 268–292.
URL: <https://doi.org/10.1145/226643.226652>.
A preliminary version of this article appears in FOCS 1991.
(Cited on page 369.)

- [For89] Lance Fortnow
The complexity of perfect zero-knowledge
In Advances in Computing Research 5 (1989), pp. 327–343.
URL: <https://doi.org/10.1145/28395.28418>.
A preliminary version of this article appears in STOC 1987.
(Cited on page 276.)
- [FS86] Amos Fiat and Adi Shamir
How to prove yourself: practical solutions to identification and signature problems
In Proceedings of CRYPTO 1986 (6th International Cryptology Conference), pp. 186–194.
URL: https://doi.org/10.1007/3-540-47721-7_12.
(Cited on pages 74, 111, 121, 369.)
- [GH98] Oded Goldreich and Johan Håstad
On the complexity of interactive proofs with bounded communication
In Information Processing Letters 67.4 (1998), pp. 205–214.
URL: [https://doi.org/10.1016/S0020-0190\(98\)00116-1](https://doi.org/10.1016/S0020-0190(98)00116-1).
(Cited on page 276.)
- [GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum
Delegating computation: interactive proofs for Muggles
In Journal of the ACM 62.4 (2015), 27:1–27:64.
URL: <https://doi.org/10.1145/2699436>.
A preliminary version of this article appears in STOC 2008.
(Cited on page 275.)
- [GLSTW23] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby
Brakedown: linear-time and field-agnostic SNARKs for R1CS
In Proceedings of CRYPTO 2023 (43rd Annual International Cryptology Conference).
URL: https://doi.org/10.1007/978-3-031-38545-2_7.
(Cited on page 278.)
- [GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff
The knowledge complexity of interactive proof systems
In SIAM Journal on Computing 18.1 (1989), pp. 186–208.
URL: <https://doi.org/10.1137/0218012>.
A preliminary version of this article appears in STOC 1985.
(Cited on page 369.)
- [GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson
Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems
In Journal of the ACM 38.3 (1991), pp. 691–729.
URL: <https://doi.org/10.1145/116825.116852>.
A preliminary version of this article appears in FOCS 1986.
(Cited on pages 276, 278.)
- [GVW02] Oded Goldreich, Salil Vadhan, and Avi Wigderson
On interactive proofs with a laconic prover
In Computational Complexity 11.1-2 (2002), pp. 1–53.
URL: <https://doi.org/10.1007/s00037-002-0169-0>.
A preliminary version of this article appears in ICALP 2001.
(Cited on page 276.)

- [GW11] Craig Gentry and Daniel Wichs
Separating Succinct Non-Interactive Arguments From All Falsifiable Assumptions
In Proceedings of STOC '11 (43rd Annual ACM Symposium on Theory of Computing), pp. 99–108.
URL: <https://doi.org/10.1145/1993636.1993651>.
(Cited on page 372.)
- [Hol19] Justin Holmgren
On round-by-round soundness and state restoration attacks
Cryptology ePrint Archive, Paper 2019/1261. 2019.
URL: <https://eprint.iacr.org/2019/1261>.
(Cited on page 371.)
- [IMSX15] Yuval Ishai, Mohammad Mahmoody, Amit Sahai, and David Xiao
On Zero-Knowledge PCPs: Limitations, Simplifications, and Applications
2015.
URL: <http://www.cs.virginia.edu/~mohammad/files/papers/ZKPCPs-Full.pdf>.
(Cited on page 372.)
- [IR89] Russell Impagliazzo and Steven Rudich
Limits on the provable consequences of one-way permutations
In Proceedings of STOC 1989 (21st Symposium on Theory of Computing), pp. 44–61.
URL: <https://doi.org/10.1145/73007.73012>.
(Cited on page 369.)
- [Kil92] Joe Kilian
A note on efficient zero-knowledge proofs and arguments
In Proceedings of STOC 1992 (24th Symposium on Theory of Computing), pp. 723–732.
URL: <https://doi.org/10.1145/129712.129782>.
(Cited on pages 207, 369, 372.)
- [LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan
Algebraic methods for interactive proof systems
In Journal of the ACM 39.4 (1992), pp. 859–868.
URL: <https://doi.org/10.1145/146585.146605>.
A preliminary version of this article appears in FOCS 1990.
(Cited on page 275.)
- [LZ19] Qipeng Liu and Mark Zhandry
Revisiting post-quantum Fiat-Shamir
In Proceedings of CRYPTO '19 (39th Annual International Cryptology Conference), pp. 326–355.
URL: https://doi.org/10.1007/978-3-030-26951-7_12.
(Cited on page 371.)
- [Mer89a] Ralph C. Merkle
A certified digital signature
In Proceedings of CRYPTO 1989 (9th International Cryptology Conference), pp. 218–238.
URL: https://doi.org/10.1007/0-387-34805-0_21.
(Cited on pages 166, 369.)
- [Mer89b] Ralph C. Merkle
One way hash functions and DES
In Proceedings of CRYPTO 1989 (9th International Cryptology Conference). Vol. 435, pp. 428–446.

- URL: https://doi.org/10.1007/0-387-34805-0_40.
(Cited on page 369.)
- [Mic00] Silvio Micali
Computationally sound proofs
In SIAM Journal on Computing 30.4 (2000), pp. 1253–1298.
URL: <https://doi.org/10.1137/S0097539795284959>.
A preliminary version of this article appears in FOCS 1994.
(Cited on pages 216, 370.)
- [Mie09] Thilo Mie
Short PCPPs verifiable in polylogarithmic time with $o(1)$ queries
In Annals of Mathematics and Artificial Intelligence 56 (3 2009), pp. 313–338.
URL: <https://doi.org/10.1007/s10472-009-9169-y>.
(Cited on page 277.)
- [Pas03] Rafael Pass
On deniability in the common reference string and random oracle model
In Proceedings of CRYPTO 2003 (23rd International Cryptology Conference), pp. 316–337.
URL: https://doi.org/10.1007/978-3-540-45146-4_19.
(Cited on page 369.)
- [PS96] David Pointcheval and Jacques Stern
Security proofs for signature schemes
In Proceedings of EUROCRYPT 1996 (15th International Conference on Theory and Application of Cryptographic Techniques), pp. 387–398.
URL: https://doi.org/10.1007/3-540-68339-9_33.
(Cited on page 371.)
- [RR20] Noga Ron-Zewi and Ron Rothblum
Local proofs approaching the witness length
In Proceedings of FOCS 2020 (61st Symposium on Foundations of Computer Science), pp. 846–857.
URL: <https://doi.org/10.1109/FOCS46700.2020.00083>.
(Cited on page 278.)
- [RR22] Noga Ron-Zewi and Ron D. Rothblum
Proving as fast as computing: succinct arguments with constant prover overhead
In Proceedings of STOC 2022 (54th Symposium on the Theory of Computing), pp. 1353–1363.
URL: <https://doi.org/10.1145/3519935.3519956>.
(Cited on page 278.)
- [RRR21] Omer Reingold, Ron Rothblum, and Guy Rothblum
Constant-round interactive proofs for delegating computation
In SIAM Journal on Computing 50.3 (2021).
URL: <https://doi.org/10.1137/16M1096773>.
A preliminary version of this article appears in STOC 2016.
(Cited on pages 275, 369.)
- [RS97] Ran Raz and Shmuel Safra
A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP
In Proceedings of STOC 1997 (29th Symposium on Theory of Computing), pp. 475–484.
URL: <https://doi.org/10.1145/258533.258641>.
(Cited on page 277.)

- [Sch89] Claus-Peter Schnorr
Efficient identification and signatures for smart cards
In Proceedings of CRYPTO 1989 (9th International Cryptology Conference), pp. 239–252.
URL: https://doi.org/10.1007/0-387-34805-0_22.
(Cited on pages 279, 369, 371.)
- [Sha92] Adi Shamir
IP = PSPACE
In Journal of the ACM 39.4 (1992), pp. 869–877.
URL: <https://doi.org/10.1145/146585.146609>.
A preliminary version of this article appears in FOCS 1990.
(Cited on page 275.)
- [Tha22] Justin Thaler
Proofs, arguments, and zero-knowledge
2022.
URL: <https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html>.
(Cited on page xii.)
- [Val08] Paul Valiant
Incrementally verifiable computation or proofs of knowledge imply time/space efficiency
In Proceedings of TCC 2008 (5th Theory of Cryptography Conference), pp. 1–18.
URL: https://doi.org/10.1007/978-3-540-78524-8_1.
(Cited on pages 369, 370.)
- [YZ22] Takashi Yamakawa and Mark Zhandry
Verifiable quantum advantage without structure
In Proceedings of FOCS 2022 (63rd Symposium on Foundations of Computer Science), pp. 69–74.
URL: <https://doi.org/10.1109/FOCS54457.2022.00014>.
(Cited on page 371.)
- [ZKP22] ZKProof
ZKProof community reference
2022.
URL: <https://docs.zkproof.org/reference>.
(Cited on page xi.)