Karlsruhe Institute of Technology

KASTEL

Dynamic Extraction of Multi-Round Knowledge Argument Systems

Master's Thesis of

Konstantin Fickel

at the Department of Informatics
Institute of Theoretical Informatics

Reviewer:	Prof. Dr. Jörn Müller-Quade
Second reviewer:	Prof. Dr. Thorsten Strufe
Advisor:	M.Sc. Michael Klooß

01. August 2021 – 28. February 2022

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, February 25th 2022

Konstantin Fickel

(Konstantin Fickel)

Abstract

Bulletproofs [5] are a popular proof system in the discrete logarithm setting. They can be used to enable confidential transactions on blockchains. The central component of Bulletproofs is the Inner Product Argument. It proves that two vectors, which have been committed to, result in a certain inner product, using a logarithmic amount of communication rounds in the size of the vectors.

To prove knowledge soundness of those protocols, the emulator first builds up a tree of transcripts and then uses this tree to extract the witness. Since the proof system is based on the discrete logarithm assumption, extraction has to either yield a discrete logarithm relation or vectors the commitments can be opened to.

Hoffmann, Klooß, and Rupp [8] noticed that this extraction process never uses the whole tree of transcripts. In the case of a successful vector extraction, only a linear amount of transcripts is needed; in the case of a discrete logarithm relation being found the amount is quasi-linear in the size of the proof. This is an improvement compared to the quadratic amount required in [5]. To make use of this observation, dynamic access to the tree is required with new transcripts generated on demand. This has been left as an open problem for further research.

This thesis outlines how such a dynamic access to the tree of transcripts may look like: First an abstraction for the sampling behaviour of such an extractor is formalized, based on which it is explained why the commonly used proof technique does not work in this case. Then, the Inner Product Argument and its extraction are described and proven. Finally, a formal framework describing dynamic extraction is specified and important properties are proven.

Zusammenfassung

Bulletproofs [5] sind ein beliebtes auf der diskreten Logarithmus-Annahme basierendes Beweissystem. Dieses kann unter anderem verwendet werden, um vertrauliche Transaktionen auf Blockchains zu ermöglichen. Die Hauptkomponente davon ist das sogenannte Inner Product Argument, bei welchem für zwei Vektoren, auf welche Commitments bekannt sind, bewiesen wird, dass diese ein bestimmtes Skalarprodukt besitzen. Dabei wird nur eine logarithmische Rundenanzahl in der Länge der Vektoren benötigt mit einer geringen, konstanten Anzahl an zu sendenden Elementen.

Um Knowledge Soundness für diese Protokolle zu beweisen, muss ein Emulator angegeben werden. Dieser baut gewöhnlicherweise zuerst einen Transkriptbaum auf und nutzt diesen dann, um die Vektoren als Zeugen zu extrahieren. Da das Beweissystem auf der diskreten Logarithmus-Annahme basiert, liefert die Extraktion also entweder einen diskreten Logarithmus oder Vektoren, die die Commitments öffnen.

Hoffmann, Klooß und Rupp [8] haben bemerkt, dass beim Extraktionsprozess nie der gesamte Transkriptbaum benötigt wird. Im Falle einer erfolgreichen Vektorextraktion werden nur eine lineare Anzahl an Transkripten in der Größe des Zeugen verwendet; wenn ein diskreter Logarithmus gefunden wurde, benötigt man eine quasi-lineare Anzahl. Dies ist deutlich besser als die quadratische Anzahl an Transkripten aus dem Bulletproof-Papier [5]. Um diese Beobachtung nutzen zu können, wird ein dynamischer Zugriff auf den Transkriptbaum benötigt. Dieses Problem haben die Autoren für weitere Forschung offen gelassen.

Diese Masterarbeit beschäftigt sich damit, wie ein solcher dynamischer Zugriff auf den Transkriptbaum aussehen könnte: Zuerst wird eine Abstraktion des Transkriptanfrageverhaltens des Extraktors formuliert, anhand derer dann die Grenzen der bisherigen Beweisstrategie aufgezeigt werden. Außerdem werden das Inner Product Argument und dessen Extraktion konkretisiert und bewiesen. Zum Abschluss wird ein formelles Framework für die Beschreibung der dynamischen Extraktion und des dynamischen Transkriptbaumaufbaus aufgezeigt und wichtige Eigenschaften desselben bewiesen.

Contents

List of Figures

List of Tables

1. Introduction

Bulletproofs [5] are a proof system based on the discrete logarithm assumption. One important application are range proofs. Those proofs can be used to convince other parties that a value hidden inside a commitment is within a certain range without revealing the value. No additional information except for the value being within the range can be extracted from the proof, so that they are zero knowledge. On blockchains they can be used to enable confidential transactions.

The central component of Bulletproofs is their Inner Product Argument. This thesis will be about this part of the proof system.

Inner Product Argument The statement to be proven in the Inner Product Argument is that a Pedersen multi-commitment $[c]$ can be opened to two vectors $\mathbf{w}'$ and $\mathbf{w}''$ and their inner product $t = \langle \mathbf{w}', \mathbf{w}'' \rangle$. The latter inner product t is also known as a part of the statement. The Inner Product Argument itself is not zero knowledge. This is not needed for the embedding protocols to be zero knowledge. Hoffmann, Klooß, and Rupp [8] described a way to make the component itself zero knowledge.

An argument of knowledge like the Bulletproof Inner Product Argument is a protocol between two programs, the prover P and the verifier V. The objective of the prover P is to convince the verifier V that a certain statement holds and that P has knowledge of a witness for this statement. The conversation between the two parties consists of multiple messaging rounds, where the verifier V sends random challenges and the prover P responds. At the end of the interaction the verifier V decides whether they believe that the statement holds. The security of the Inner Product Argument is based on the discrete logarithm assumption.

The idea of the Inner Product Argument is to recursively downsize the statement: In every step, the verifier sends a random challenge. The prover P then splits the witness and the commitment key in half and folds them together weighted by the challenge. Then the information required to calculate the new downsized statement is sent to the verifier V. Now both parties have agreed on a statement of half the size for which this protocol can be repeated. After a logarithmic amount of rounds we have a witness of constant size, which can be sent directly.

We want an argument of knowledge to have two security properties: When a protocol is perfectly complete, the verifier is always convinced by an honest prover. In this thesis we are more interested in the second property: For knowledge soundness, intuitively, every prover able to convince the verifier has to have knowledge about the witness.

Knowledge Soundness To formalize this notion, we consider a possibly dishonest prover P^*. We construct an emulator E able to rewind P^*, which outputs transcripts of the protocol between V from the protocol and P^* that are indistinguishable from real transcripts. If the generated transcript is accepting, this extractor also outputs additional witness for the relation with a high probability.

The construction of such an emulator for the Bulletproof Inner Product Argument is split into two steps: Using rewind access to the prover P^*, first a tree of transcripts is generated.

The nodes in the tree correspond to messages by the prover P^*; the edges correspond to the challenges sent by the verifier V. In the tree a path from the root to a leaf corresponds to an accepting transcript. On a layer of the tree every node has the same branching factor. The different children correspond to different continuations of the protocol with different challenges by the verifier.

In a second step an extractor receives this tree of transcript as a static input. The extractor works its way up from the leaves recovering the witness for the statement at every round, until it reaches the root. Afterwards the calculated witness is output. Protocols for which we know an extractor converting a static tree of transcript into a witness are called special sound.

Forking Lemma The Forking Lemma [4] states that knowledge soundness follows from special soundness. In the proof a tree builder is constructed and analyzed, which combined with the extractor gives us an emulator as needed for knowledge soundness.

It is important for the theoretically sound choice of parameters how efficient extraction can be done, i.e. how many transcripts and how much runtime of the emulator are needed. Intuitively, this efficiency is a measure of how well the information about the witness can be hidden in a possibly dishonest prover. With the usual approach, the extractor requires an quadratic amount of transcript in the size of the witness.

Extraction for the Inner Product Argument has to either yield the vectors $\mathbf{w}'$ and $\mathbf{w}''$ behind the commitment or a discrete logarithm relation. Hoffmann, Klooß, and Rupp [8] observed that fewer transcripts have to be used in case no discrete logarithm relation is found. If regular extraction fails, more different challenges are required to find a discrete logarithm relation, but then extraction can be stopped after the current subtree. The problem here is that both outcomes require different parts of the tree, and a lot of the transcripts remain unused for extraction. Because we don't know before extraction which of the two outcomes occurs, we can not take advantage of this observation if we use the static tree finder.

This thesis formalizes a dynamic tree finder and therefore a dynamic version of the Forking Lemma, which makes it possible to construct more efficient extractors for protocols like the Inner Product Argument.

1.1. Related Work

The following section will be separated into different parts, based on the topic the authors made progress in:

1.1.1. Protocols

The Bulletproof Inner Product Argument was first introduced by Bootle et al. [4]. Using this protocol the inner product relation discussed previously can be proven in an logarithmic amount of communication rounds. Compared to later iterations of the protocol, it featured two distinct Pedersen multi-commitments to the secret vectors $\mathbf{w}'$ and $\mathbf{w}''$, making the protocol less communication efficient. Additionally, a protocol for range proofs and one for proving satisfiability of arithmetic circuits using the Inner Product Argument was given.

The Inner Product Argument in the Bulletproof paper [5] followed the same objective, but merged the two commitments into one decreasing the size of the proof. Additionally, two

protocols for range proofs and arithmetic circuit satisfiability built upon the Inner Product Argument were given.

Hoffmann, Klooß, and Rupp [8] made minor changes to the Inner Product Argument and observed design patterns used in those protocols. Additionally, they discovered a way to extract the witness using fewer transcripts, which will be discussed in more details in Subsection 1.1.3.

1.1.2. Forking Lemma and Knowledge Error

While the Forking Lemma from [4] has a relatively large knowledge error of $O(\text{poly}(\kappa)/\sqrt[3]{p})$ with p being the size of the challenge space, Jaeger and Tessaro [9] improved the analysis of the extractor to $\text{poly}(\kappa)/\sqrt[2]{p}$. Instead of only analyzing asymptotic behaviour, [9] provided a concrete analysis of the security, which allows for a theoretically sound choice of parameters. We will adapt this extractor to the dynamic setting and modify their analysis accordingly.

By choosing the challenges without replacement instead of bounding the probability of collisions, Attema, Cramer, and Kohl [1] achieved an optimal knowledge error of $\text{poly}(\kappa)/p$ by using a modified extractor.

Along with their improved analysis of the extractor, Jaeger and Tessaro [9] provided a generalization of witness-extended emulation to predicate extended emulation: Instead of requiring the extractor to find a witness fulfilling a certain relation, now predicates are realized on auxiliary output at different steps of the extraction process. Those predicates check if the auxiliary output is a witness to the given statement as before, or verify properties of intermediary outputs like the tree of transcripts. This framework allows for describing the algorithm and its properties in a more precise way and specifying the capabilities of an extractor by stating the predicates required on the input and those promised on the output. Those concepts and frameworks will be extended in this thesis.

When converting public-coin interactive proofs like Bulletproofs into non-interactive ones using the Fiat-Shamir transform, the security loss for $2\mu + 1$-round protocols for an attacker with Q random oracle queries is generally Q^μ. Attema, Fehr, and Klooß [2] showed by a more careful analysis that for $(k_1, \ldots, k_\mu)$-special sound protocols the knowledge error degrades linearly in Q.

1.1.3. Size of the Tree of Transcripts

The extractors described so far [4, 5, 1] all require a tree of transcripts of quadratic size, making the loss in security of the protocol quadratic. Hoffmann, Klooß, and Rupp [8] showed that either a linear- or a quasi-linear sub-tree is enough if the transcripts are generated on demand. This thesis will build upon this observation and attempt to achieve linear extraction.

Ghoshal and Tessaro [7] achieved a linear amount of required transcripts along with a knowledge error of $\text{poly}(\kappa)/p$ by using the Algebraic Group Model. In this model, adversaries like the malicious prover have to provide a representation using previously known group elements for every new group element sent. Instead of showing extraction for the interactive version of the protocol, extraction was shown for the non-interactive proof obtained by using the Fiat-Shamir Transformation.

1.2. Contribution

This thesis paves the way towards a linear extraction of the Bulletproof Inner Product Argument: A game specifying the sampling behaviour required for quick extraction is defined, which makes it simpler to prove a bound on the expected amount of transcripts required when sampling. A conjecture for such a bound is given, along with some observations on problems and a possible structure for its proof.

The Bulletproof Inner Product Argument and its extraction are described in detail. Here, a focus is set on the faster extraction technique. The idea of dynamic requests for transcripts is formalized, and a notational framework for dynamic extractors is defined. Using this framework, a dynamic version of the Forking Lemma is proven.

Finally, a theorem is proven, that combines the dynamic extraction framework, its Forking Lemma, the extractor and possible observations about the sampling behaviour to demonstrate linear extraction in the size of the witness.

2. Preliminaries

The definitions in the following chapter follow the ones by [9, Section 6], [1, Section 2] and [8, Section 2].

We call a function $f\colon \mathbb{N} \to \mathbb{R}_{>0}$ *negligible* if for all $c \in \mathbb{N}$, there exists a $\kappa_0 \in \mathbb{N}$ such that $f(\kappa) \leq 1/\kappa^c$ for all $\kappa \geq \kappa_0$. In short, we write $\mathrm{negl}(\kappa)$ to denote a negligible function $\mathrm{negl}\colon \mathbb{N} \to \mathbb{R}_{>0}$. *Polynomially bounded* functions in the security parameter κ are denoted by $\mathrm{poly}(\kappa)$, i.e. if there exist $c, \kappa_0 \in \mathbb{N}$ such that $\mathrm{poly}(\kappa) \leq \kappa^c$ for all $\kappa \geq \kappa_0$.

We will denote the input of algorithms by small letters like x, and sometimes instead of explicitly parametrizing the security parameter κ we implicitly parameterize it by input length $|x|$.

If S is a set, $x \leftarrow\!\!\$\, S$ stands for sampling an uniformly random element. For a probabilistic algorithm Alg, $x \leftarrow\!\!\$\, \mathsf{Alg}(\cdots)$ assigns the result of the execution to the variable x.

2.1. Mathematical Foundations

The integer $p \in \mathbb{N}$ will always denote a prime number in this thesis. It is used to parameterize the field $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$. The group $\mathbb{G}$ is a cyclic abelian group of order p, for which we will be using additive implicit notation. Elements of this group are written within brackets, for example $[Q] \in \mathbb{G}$. $[1] \in \mathbb{G}$ denotes the generator of the group, which together with the group order p is implicitly given along with the description of the group $\mathbb{G}$.

In mathematical contexts, we will write matrices and vectors in bold font, for example $\mathbf{w} \in \mathbb{F}_p$, and single elements like $[Q] \in \mathbb{G}$ in regular font. Elements of a matrix can be indexed by a subscript letter like $w_i \in \mathbb{F}_p$ for the ith element of $\mathbf{w} \in \mathbb{F}_p^n$ (where w_1 is the first) if they are just regular single elements, they will be denoted in regular font.

2.1.1. Discrete Logarithm Assumption

The Bulletproof proof system is built upon the discrete logarithm assumption, for which the following definition is from [8, Defintion 2.1] and [5, Defintion 1]:

Definition 1 (Discrete Logarithm Relation / Assumption)**.** For $\mathbf{g} \in \mathbb{G}^n$, a vector $\mathbf{v} \in \mathbb{F}_p^n$ is a discrete logarithm relation if $\mathbf{v} \in \mathrm{Ker}([\mathbf{g}])$, i.e. $[\mathbf{g}]\,\mathbf{v} = 0$. If additionally $\mathbf{v} \neq 0$, we call $\mathbf{v}$ a non-trivial discrete logarithm relation.

Let $\mathsf{GrpGen}(1^\kappa)$ be a group generator. The discrete logarithm assumption holds for $\mathbb{G}$ if

$$\mathbf{Pr}\left[\mathbb{G} \leftarrow\!\!\$\, \mathsf{GrpGen}(1^\kappa);\, [\mathbf{g}] \leftarrow\!\!\$\, \mathbb{G}^m;\, \mathbf{v} \leftarrow\!\!\$\, \mathcal{A}(1^\kappa, \mathbb{G}, [\mathbf{g}]) : [\mathbf{g}]\,\mathbf{v} = 0 \wedge \mathbf{v} \neq 0\right] \leq \mathrm{negl}(\kappa)$$

Note that finding a non-trivial discrete logarithm relation generalizes the discrete logarithm assumption: A non-trivial kernel element for $[\,h\;1\,] \in \mathbb{G}^2$ directly yields the discrete logarithm of $[h]$.

2.1.2. Pedersen Commitments

We will use Pedersen-Commitments [13] to "hide" the vectors we want to prove statements about. For a given commitment key $\mathsf{ck} = [\mathbf{g}] \in \mathbb{G}^n$, we can commit to a vector $\mathbf{w} \in \mathbb{F}_p^n$ by calculating and sending $\mathsf{Com}_{\mathsf{ck}}(\mathrm{w}) = [c] = [\mathbf{g}]\,\mathbf{w} \in \mathbb{G}$. The commitment can then be unveiled by publishing $\mathbf{w}$.

This form of the Pedersen-Commitment is binding, because from two different openings a discrete logarithm relation can easily be calculated. To make the commitment hiding, it would have to be extended by an additional blinding factor $r_{\mathbf{w}} \leftarrow\!\!\$\, \mathbb{F}_p$ so that $[c] = [\mathbf{g}]\,\mathbf{w} + [h]\,r_{\mathbf{w}}$ with $[h] \in \mathbb{G}$; when unveiling, the $r_{\mathbf{w}}$ would have to be revealed along with $\mathbf{w}$.

2.2. Interactive Proofs

The following definitions of proof systems and their extraction follow [2] and are adapted to match the ones by [9, Section 6].

Definition 2 (Interactive Proof System)**.** A μ-round Interactive Proof System PS is a quintuple $\mathsf{PS} = (\mathsf{S}, \mathcal{R}, \mathsf{P}, \mathsf{V}, \mu)$ consisting of the following:

- A setup algorithm S generating the public parameters pp.
- A relation $\mathcal{R}$, for which membership and knowledge of a witness is proven.
- An interactive, polynomially bounded Prover Turing-machine P and
- an interactive, polynomially bounded Verifier Turing-machine V,
- as well as the number of communication rounds $\mu \in \mathbb{N}$.

At the start of the execution of the protocol $\langle \mathsf{P}_{pp}(u, w), \mathsf{V}_{pp}(u) \rangle$, which can found written down as pseudo-code in Figure 5.2, both verifier V and prover P receive the statement u and the prover is additionally given the secret witness w. After the prover has sent its first message m_0, there are μ rounds of communication with the messages m_{2i-1} from the verifier and m_{2i} from the prover for $i \in \{1, \ldots, \mu\}$. The messages are collected in the transcript $\mathrm{tr} = (m_0, m_1 \ldots, m_{2\mu-1}, m_{2\mu})$. As the output d of the protocol, V either accepts ($d = 1$) or rejects ($d = 0$) the prover's claim of knowing a w such that $(u; w) \in \mathcal{R}_{pp}$. If the verifier V accepts after the protocol, the corresponding transcript tr is called an accepting transcript. Otherwise, it is called an rejecting transcript.

When considering prover and verifier as algorithms or Turing-machines, we denote them as P and V if they are seen as parties of the mathematical description of the protocol, they are denoted by $\mathscr{P}$ and $\mathscr{V}$.

2.2.1. Properties of Interactive Proofs Systems

Two properties are required for a proof system to be useful: An honest prover has to be able to convince the verifier that it has indeed knowledge of the witness, which is called *completeness* of the protocol, and a prover P^* that manages to convince the verifier V from the protocol, has knowledge of the witness — which is called *knowledge soundness.* Defining completeness is simple:

Definition 3 (Perfect Completeness)**.** Perfect Completeness requires that for all pp and $(u, w) \in \mathcal{R}_{pp}$:

$$\mathbf{Pr}\left[d = 1 \,\middle|\, (\cdot, d) \leftarrow\!\$ \left\langle \mathsf{P}_{pp}(u, w), \mathsf{V}_{pp}(u) \right\rangle\right] = 1.$$

The other property required is *knowledge soundness*, which was initially introduced by Bellare and Goldreich [3] as *validity*. The updated definition by Attema, Cramer, and Kohl [1] is presented in the following.

Definition 4 (Knowledge Soundness)**.** Let $\mathsf{PS} = (\mathsf{S}, \mathcal{R}, \mathsf{P}, \mathsf{V}, \mu)$ be an interactive proof system for relation $\mathcal{R}$. Let $\kappa \colon \mathbb{N} \to [0, 1)$ be a function.

A proof system PS is knowledge sound with knowledge error κ if there exists a polynomial $q \colon \mathbb{N} \to \mathbb{N}$ and an algorithm E called knowledge extractor with the following properties: E given input u and rewindable oracle access to a (potentially dishonest) prover P^*, runs in an expected polynomial number of steps in the size of the statement and, whenever $\langle \mathsf{P}^*_{pp}(u, s), \mathsf{V}_{pp}(u) \rangle$ outputs an accepting transcript with probability $\epsilon(u) \geq \kappa(|u|)$, it successfully outputs a witness w with $(u; w) \in \mathcal{R}$ with probability at least ${}^{(\epsilon(u) - \kappa(|u|))}/_{q(|u|)}$.

A proof system with both of those properties is called a proof or argument of knowledge.

Definition 5 (Proof of Knowledge (or Argument of Knowledge))**.** A proof system PS, which is perfectly complete and knowledge sound with knowledge error κ is called a proof of knowledge.

An argument of knowledge identical to the one of a proof of knowledge, but knowledge soundness only holds under certain computational assumptions.

If the challenges sent are just the randomness drawn by the verifier, the protocol is called public coin:

Definition 6 (Public Coin Protocol)**.** A proof system PS is public coin if the messages by the verifier m_{2i-1} are set equal to V's random coins.

In this case, we let $\mathsf{V}_{pp}(u, \mathrm{tr}) \in \{0, 1\}$ denote V's decision on whether tr is accepting.

2.2.2. Special Soundness

Extraction for protocols like the Bulletproof inner product argument [5] is separated into two steps: First, the possibly cheating prover P^* is run and rewound to build up the tree of transcripts. Later, this tree of transcripts is used to calculate a witness. The following definition of a tree of transcripts is taken from [1, Definition 9], which is not to be mistaken for the abstraction of a tree of 1-entries introduced in Section 3.1.

Definition 7 (Tree of Transcripts)**.** Let $(k_1, \ldots, k_\mu) \in \mathbb{N}^\mu$. A $(k_1, \ldots, k_\mu)$-tree of transcripts for a $(2\mu + 1)$-move public coin protocol is a set of $K = \prod_{i=1}^{\mu} k_i$ transcripts arranged in the following tree structure: The nodes in this tree correspond to the prover's messages and the edges correspond to the verfier's challenges. Every node at depth i has precisely k_i children corresponding to the verifier's challenges. Every transcript corresponds to exactly one path from the root node to a leaf node.

An extractor using an $\mathbf{k} = (k_1, \ldots, k_\mu)$-tree of transcripts to find a witness makes a proof system $\mathbf{k}$-special sound:

Definition 8 ($(k_1, \ldots, k_\mu)$-Special Soundness)**.** Let $(k_1, \ldots, k_\mu) \in \mathbb{N}^\mu$. A $(2\mu + 1)$-move public coin protocol is $(k_1, \ldots, k_\mu)$-special sound if there exists an efficient algorithm that on input a $(k_1, \ldots, k_\mu)$-tree of accepting transcripts outputs a witness w such that $(u; w) \in \mathcal{R}$.

As shown by Bünz et al. [5], Attema, Cramer, and Kohl [1], and Jaeger and Tessaro [9] in their forking lemma, from special soundness of a proof system PS we can infer knowledge soundness by combining the tree extractor with a suitable tree finder.

2.3. Quick- & Short-Circuit-Extraction

For extraction of the Bulletproof protocol, not all $K = \prod_{i=1}^{\mu} k_i$ transcripts are required. To analyze protocols with those properties, Hoffmann, Klooß, and Rupp [8] introduced short-circuit- and quick-extraction: Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be two relations, where a statement u belongs to the relation if there exits a witness w such that $(u; w) \in \mathcal{R}_i$ for $i = 1, 2$. We compose those two relations into $\mathcal{R} = \mathcal{OR}(\mathcal{R}_1, \mathcal{R}_2)$, so that a combined statement (u_1, u_2) belongs to the relation $\mathcal{R}$ if a witness for one of the two statements can be found, i.e.

$$\{((u_1, u_2), (w_1, w_2)) \,|(u_i, w_i) \in \mathcal{R}_i \text{ for } i = 1 \text{ or } i = 2\}.$$

Let $(k_1, \ldots, k_\mu) \in \mathbb{N}^\mu$ and $(\hat{k}_1, \ldots, \hat{k}_\mu) \in \mathbb{N}^\mu$, such that $k_i \leq \hat{k}_i$. If there exists quick- and short-circuit extraction of a protocol, two different cases can be distinguished: In the first, a smaller $(k_1, \ldots, k_\mu)$-subtree suffices for extraction, and a witness for the first relation $\mathcal{R}_1$ is found. We call this *quick-extraction.* If on a layer i of the tree quick-extraction fails, the extractor requests $\hat{k}_i$ children. After having explored all of those children, a witness for relation $\mathcal{R}_2$ is found and extraction for the rest of the tree is aborted. This behaviour is called *short-circuit extraction.*

In the following Chapter 3.1, the abstract $(\hat{k}_1, \ldots, \hat{k}_\mu)$-burst $(k_1, \ldots, k_\mu)$-sampling game is considered.

3. Burst Extraction

In the following sections, we will define the $(\hat{k}_1, \ldots, \hat{k}_\mu)$-burst $(k_1, \ldots, k_\mu)$-collision game. This game is an abstraction of the sampling process used by our extractor in the following chapters, and will allow for a simpler analysis of the extraction process.

3.1. Burst Collision Game

Instead of explicitly considering running the malicious prover P^*, we will assume that we have all outcomes for all possible challenge choices $c_1, \ldots, c_\mu \in \{1, \ldots, N\}$ by V and all possible prover-randomnesses $a \in \{1, \ldots, R\}$ encoded in a $(\mu+1)$-dimensional matrix $H \in \{0,1\}^{R \times N \times \cdots \times N}$. We can access elements of the matrix by $H\left[a, c_1, \ldots, c_\mu\right]$, and $H\left[a, c_1, \ldots, c_\mu\right] = 1$ signifies that the corresponding transcript of the corresponding run of the protocol is accepted by the verifier.

A sub-matrix $H\,(a, c_1, \ldots, c_m) \in \{0,1\}^{\mu-m}$ for $m \in \{1, \ldots \mu\}$, $a \in \{1, \ldots, R\}$ and $c_1, \ldots, c_m \in \{1, \ldots, N\}$ is the $(\mu - m)$-dimensional sub-matrix, which contains all entries whose first $(m+1)$-dimensions are equal to $(a, c_1, \ldots, c_m)$.

$(k_1, \ldots, k_\mu)$-trees As in [1] we define $(k_1, \ldots, k_\mu)$-trees of 1-entries in H with $\mathbf{k} = (k_1, \ldots, k_\mu) \in \mathbb{N}^\mu$ recursively as follows: For $\mu = 0$, a tree of 1-entries is simply a 1-entry in H. For arbitrary μ, a $(k_1, \ldots, k_\mu)$-tree of 1-entries is the union of k_1 $(k_2, \ldots, k_\mu)$-trees of 1-entries in $H\,(a, c_1)$ up to $H\left(a, c_{k_1}\right)$ respectively with fixed a and pairwise distinct c_i. In Figure 3.1 one can find a sketch of such a tree.

As a result, by this definition we have that a $(k_1, \ldots, k_\mu)$-tree of 1-entries in matrix H is a tuple of $K = \prod_{i=1}^{\mu} k_i$ 1-entries identified by a tuple of challenges $\{1, \ldots, R\} \times \{1, \ldots N\}^\mu$ that are in a $(k_1, \ldots, k_\mu)$-tree structure. A $(k_{M+1}, \ldots, k_\mu)$-subtree of a $(k_1, \ldots, k_\mu)$-tree of 1-entries is a $(1, \ldots, 1, k_{M+1}, \ldots, k_\mu)$-tree of 1-entries in H, which corresponds to a $(k_{M+1}, \ldots, k_\mu)$ in $H\,(a, c_1, \ldots, c_M)$ with additional entries in the tuples for $(a, c_1, \ldots c_M)$.

$(\hat{k}_1, \ldots, \hat{k}_\mu)$-burst $(k_1, \ldots, k_\mu)$-collision game The $(\hat{k}_1, \ldots, \hat{k}_\mu)$-burst $(k_1, \ldots, k_\mu)$-collision game is a model for the quick- and short-circuit extraction process suggested by Hoffmann, Klooß, and Rupp [8, Definition 2.19]: Extraction of a witness on layer m for the corresponding subtree is usually possible with k_{m+1} successful extractions on the layer below, which they call quick-extraction. If quick-extraction fails, we know that we will find a break of the discrete logarithm assumption if we manage to find $\hat{k}_i \geq k_i$ successfully extracted children of the node – after which we can then stop expansion for the entire tree.

We define a predicate β, which is run on a $(k_{m+1}, \ldots, k_\mu)$-subtree for $m \in \{0, \ldots, \mu\}$ once it is fully built up. If β evaluates to 0, we consider extraction successful, and just return the found subtree and continue with the rest of the tree. If β evaluates to 1, we say that this subtree bursts, and attempt to find $\hat{k}_{m+1}$ instead of k_{m+1} children of the root. Instead of continuing

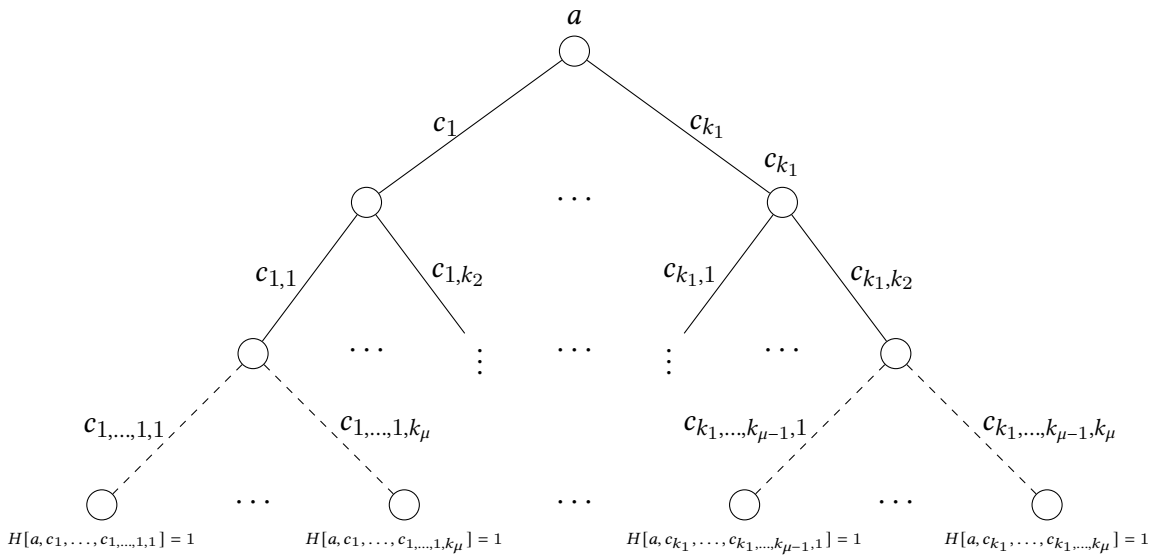


Figure 3.1.: $(k_1, \ldots, k_\mu)$-tree of transcripts of a $(2\mu + 1)$-move public-coin interactive proof following [1]

with the rest of the tree, we then just return the resulting $(\hat{k}_{m+1}, k_{m+2}, \ldots, k_\mu)$-subtree and stop building up the full tree of transcripts.

We require $\hat{k}_1, \ldots, \hat{k}_\mu, k_1, \ldots, k_\mu \in \mathbb{N}^\times$ with $k_i \leq \hat{k}_\mu$ for all $i \in \{1, \ldots, \mu\}$. The predicate β maps for all $m \in \{0, \ldots, \mu - 1\}$ a $(k_{m+1}, \ldots, k_\mu)$-subtree of 1-entries in H to $\{0, 1\}$.

The objective of the $(\hat{k}_1, \ldots, \hat{k}_\mu)$-burst $(k_1, \ldots, k_\mu)$-collision game is to either

- find a $(k_1, \ldots, k_\mu)$-tree of 1-entries in H, of which no subtree is marked by β or
- find a $(\hat{k}_{m+1}, k_{m+2}, \ldots, k_\mu)$-subtree τ of 1-entries in H, of which only the entire tree is marked by β, but none of the subtrees.

Our procedure to play the $(\hat{k}_1, \ldots, \hat{k}_\mu)$-burst $(k_1, \ldots, k_\mu)$-collision game is described in Figure 3.2. Possible traversed trees by BurstTree can be found in Figure 3.3. The subtree returned is painted in black, the rest of the tree is greyed out.

3.2. Random Variables for the Extraction Process

In the remainder of this section, we will analyse the execution of the $\left(\hat{k}_1, \ldots, \hat{k}_\mu\right)$-burst $(k_1, \ldots, k_\mu)$-game. For a formal description, we now introduce random variables that indicate some properties of the run of the algorithm. Those random variables are indexed by the call parameters $v = (a, c_1, \ldots, c_m)$ used for the recursive call $\text{BurstTree}_m(a, c_1, \ldots, c_m)$ that is being contemplated. The letter v is chosen for the parameters, because they also identify the vertex in the tree of transcript generated by this function call if this function call was successful.

Counting the visited Lookups The first random variables introduced are one binary and two integer variables concerned with the sampling behaviour of BurstTree:

$A_v \in \mathbb{N}_0$: This random variable counts the amount of lookups to H that happened during the call to $\text{BurstTree}_m(a, c_1, \ldots, c_m)$ summed over all subcalls.

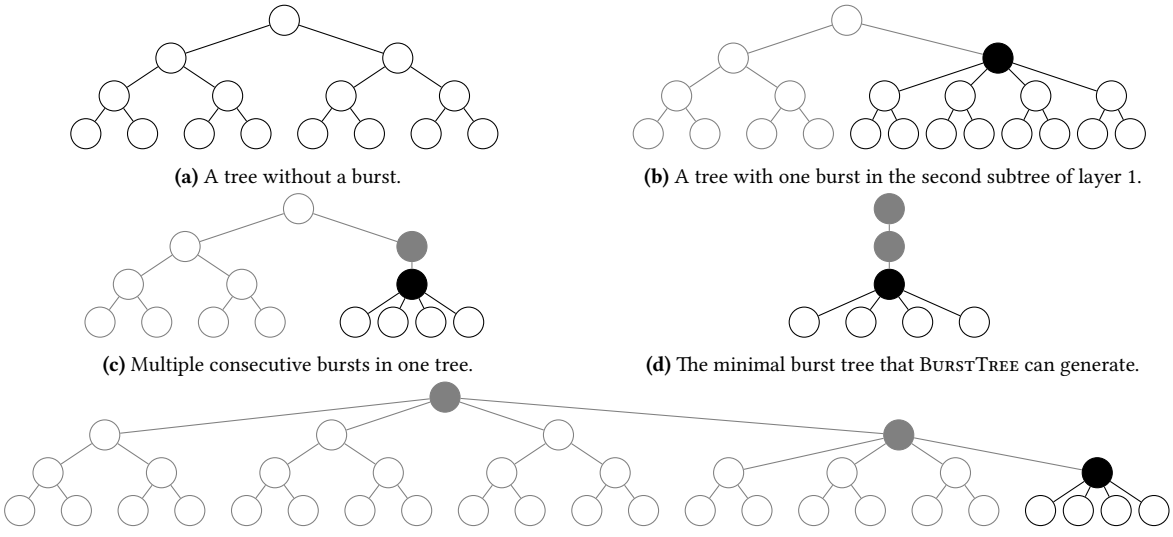
$\textsc{BurstTree}_m\,(a, c_1, \ldots, c_m)$ with parameters $a \in \{1, \ldots, R\}$ and $c_1, \ldots, c_m \in \{1, \ldots N\}$:

$m = \mu$: If $H\left[a, c_1, \ldots, c_\mu\right] = 1$, then return the tree just consisting of node $\left(a, c_1, \ldots, c_\mu\right)$, otherwise abort.

$0 \leq m < \mu$:
- Sample $c_{m+1} \leftarrow\!\!\$\ \{1, \ldots, N\}$.
- Run $\textsc{BurstTree}_{m+1}\,(a, c_1, \ldots, c_m, c_{m+1})$. If this run aborts, abort. If it returns a burst-subtree, immediately return this subtree.
- Repeat until k_{m+1} $\left(k_{m+2}, \ldots, k_\mu\right)$-subtrees have been found:
  - Sample another $c_{m+1} \leftarrow\!\!\$\ \{1, \ldots, N\}$ uniformly with replacement.
  - Run $\textsc{BurstTree}_{m+1}\,(a, c_1, \ldots, c_m, c_{m+1})$. If this run aborts, continue with this loop. If it returns a burst-subtree, immediately return this subtree.
- For the subtree τ that has been generated up to now, test whether $\beta\,(\tau) = 0$. If so, this subtree is complete and return τ. Otherwise, continue sampling as before until $\hat{k}_{m+1}$ children have been found.
- Return the resulting $\left(\hat{k}_{m+1}, k_{m+2}, \ldots, k_\mu\right)$-subtree.

The $(\hat{k}_1, \ldots, \hat{k}_{m+1})$-burst $(k_1, \ldots, k_\mu)$-collision game samples $a \in \{1, \ldots R\}$ and runs $\textsc{BurstTree}_0\,(a)$. If $\textsc{BurstTree}_0\,(a) = \perp$ it aborts, otherwise it outputs either the $(k_1, \ldots, k_\mu)$-tree of 1-entries in $H\,(a)$ found or the $(\hat{k}_n, k_{m+1}, \ldots, k_\mu)$-tree of 1-entries in $H\,(a)$ in case of a burst.

Figure 3.2.: Definition of the $(\hat{k}_1, \ldots, k_\mu)$-burst $\left(k_1, \ldots, k_\mu\right)$-sampling-game



(a) A tree without a burst.

(b) A tree with one burst in the second subtree of layer 1.

(c) Multiple consecutive bursts in one tree.

(d) The minimal burst tree that BurstTree can generate.

(e) The maximal burst tree that BurstTree can generate.

Figure 3.3.: Five examples of possible found trees of transcripts by the $(4, 4, 4)$-burst $(2, 2, 2)$-game. A filled node stands for a node for whose subtree β returns 1. All leaves printed here correspond to accepting transcripts, but BurstTree only outputs the black part of the tree, i.e. the last burst subtree.

$E_v \in \{0, 1\}$: We say that a node $v = (a, c_1, \ldots, c_m)$ was entered if for the first challenge c_{m+1} sampled the call to $\text{BurstTree}_m\,(a, c_1, \ldots, c_m, c_{m+1})$ was successful, and therefore the attempt to find k_{m+1} child trees was started.

$K_v \in \mathbb{N}_0$: This is the amount of children $\text{BurstTree}_m\,(a, c_1, \ldots, c_m)$ tries to get, e.g. $K_v = k_{m+1}$ in case of a regular extraction without a burst.

We are especially interested in the expected value of the random variable $A = A_{v_{\text{root}}}$ for the "root node" $v_{\text{root}} = ()$.

Success of Extraction The following random variables are connected to the process of sampling challenges:

$O_v \in \mathbb{N}$: This is the amount of challenges $c_{m+1} \in \{1, \ldots, N\}$ for which BurstTree_{m+1} was called during the execution of $\text{BurstTree}_m\,(a, c_1, \ldots, c_m)$ *excluding the first one.* This counts both successful and failed recursive calls.

$P_v \in \{1, \ldots N\}^*$: This is the tuple that contains all challenges $c_{m+1} \in \{1, \ldots, N\}$ which were tried in the run of BurstTree and for which $\text{BurstTree}_{m+1}\,(\ldots, c_{m+1}) \neq \perp$ did not abort.

$T_v \in \{1, \ldots N\}^*$: T_v is just P_v, but seen as a set, i.e. the order the challenges were tried in is omitted.

Bursts For the analysis of burst events, we introduce the following random variables:

$B_v \in \{0, 1\}$: This binary random variable is 1 if $\beta(\tau) = 1$ for the subtree generated in $\text{BurstTree}_m\,(a, c_1, \ldots, c_m)$ after successfully sampling the first k_{m+1} nodes.

$C_v \in \{0, 1\}$: For one of the possible subsequent challenges c_{m+1} occurred a burst in $\text{BurstTree}_{m+1}\,(a, \ldots, c_m, c_{m+1})$.

$R_v \in \{0,1\}$: R_v is defined as $B_v \vee C_v$, i.e. within τ with root v there exists a subtree τ' with corresponding root v' with $\beta(\tau') = 1$.

$L_v \in \{1,\dots,N\}^*$: The children in whose subtrees a node was marked by β are collected in the random variable $L_v = \{w \in \text{children}(v) \mid R_w = 1\}$.

3.3. Bounding the Worst-Case Amount of Leaves

Before starting with our proof for the expected amount of lookups in H, we recall [8, Corollary 2.20] where an upper bound for the nodes in the resulting tree of transcripts is given:

Lemma 9. *The maximal amount X of leaves in the full tree output by BurstTree in the $\left(\hat{k}_1,\dots,\hat{k}_\mu\right)$-burst $(k_1,\dots,k_\mu)$-collision game with $k_i, \hat{k}_i \in \mathbb{N}$ and $k_i \le \hat{k}_i$ can be bounded above by*

$$X \le \sum_{m=1}^{\mu}\left(\hat{k}_m - 1\right)\prod_{i=m+1}^{\mu} k_i \le \left(\sum_{m=1}^{\mu}\hat{k}_m\right)\left(\prod_{i=1}^{\mu} k_i\right). \tag{3.1}$$

Note that the first inequality in Equation (3.1) is tight and can be achieved by a burst on every level on the last child as shown in Figure 3.3e. This Lemma is different from Conjecture 10 in the next section, where the amount of lookups in H is counted instead of the leaves in the resulting tree.

Proof. We introduce the variable $n_{\text{burst}}(m)$ which counts the maximal amount of leafs in a subtree starting on level m, in which β marked a subtree. $n_{\neg\text{burst}}(m)$ is the amount of leaves BurstTree_m creates if no burst occurs. In total, we have

$$X \le \max\left\{n_{\text{burst}}(0), n_{\neg\text{burst}}(0)\right\}.$$

A node in a subtree without a burst event on layer m has at most k_{m+1} children – and therefore, by induction, is ancestor of $n_{\neg\text{burst}}(m) \le \prod_{i=m+1}^{\mu} k_i$ leaves. When considering n_{burst}, in worst case, we have regular extraction for the first $k_{m+1} - 1$ subtrees. This is followed by one subtree with a burst:

$$\begin{aligned} n_{\text{burst}}(m) &\le \left(\hat{k}_{m+1} - 1\right)\cdot n_{\neg\text{burst}}(m+1) + n_{\text{burst}}(m+1) \\ &\le \left(\hat{k}_{m+1} - 1\right)\left(\prod_{i=m+2}^{\mu} k_i\right) + n_{\text{burst}}(m+1) \end{aligned}$$

Setting $n_{\text{burst}}(\mu) = 0$ and running an induction from $m = \mu$ down to $m = 0$, we can collect the summands of the sum

$$n_{\text{burst}}(0) \le \sum_{m=0}^{\mu-1}\left(\hat{k}_{m+1} - 1\right)\left(\prod_{i=m+2}^{\mu} k_i\right) = \sum_{m=1}^{\mu}\left(\hat{k}_m - 1\right)\left(\prod_{i=m+1}^{\mu} k_i\right)$$

This is our result, since we have $X = n_{\mathsf{burst}}(0) \geq n_{\neg\mathsf{burst}}$. The second inequality of Equation (3.1) can be proven by bounding $\prod_{j=i+1}^{\mu} \hat{k}_j$ above by $\prod_{j=1}^{\mu} \hat{k}_j$ in every term and using distributivity of sums. □

As discussed by Hoffmann, Klooß, and Rupp [8], this promises already a more efficient extraction than the one given in [5]: In the Bulletproof Inner Product Argument, statements about vectors of length n are to be shown in a $\mu = \log(n)$-round protocol. The standard parameters correspond to a $(4, \ldots, 4)$-burst $(2, \ldots, 2)$-collision game. Without dynamic extraction, we can only build up a $(\hat{k}_1, \ldots, \hat{k}_\mu)$-tree, which corresponds to $4^\mu \in O\left(n^2\right)$ lookups in H. The worst case analysis from this Lemma 9 promises $(4\mu)\, 2^\mu \in O(n \log(n))$ H-lookups. In the following section, we discuss how we can achieve $2 \cdot 2^\mu \in O(n)$ in expectation by a more precise analysis.

3.4. Bounding the Expected Amount of Lookups in the Table

The following Conjecture estimates an upper bound for the expected amount of lookups in H when playing the $(k_1, \ldots, k_\mu)$-burst $(\hat{k}_1, \ldots, \hat{k}_\mu)$-collision game:

Conjecture 10 (Transcripts Generated by BurstTree)**.** *Let $H \in \{0,1\}^{R \times N \times \cdots \times N}$ be a $(\mu+1)$-dimensional matrix and let ϵ denote the fraction of 1-entries in H. Consider the $(\hat{k}_1, \ldots, \hat{k}_\mu)$-burst $(k_1, \ldots, k_\mu)$-sampling-game as defined in Figure 3.2 with $2 \leq k_m \leq \hat{k}_m$ for all $m \in \{1, \ldots, N\}$: and set*

$$\alpha = \max_{m \in \{1,\ldots,\mu\}} \frac{\hat{k}_m}{k_m} \tag{3.2}$$

Then the amount A of accesses to H by BurstTree *is bounded in expectation by*

$$\mathbb{E}[A] \leq \alpha \left(\prod_{i=1}^{\mu} k_i \right)$$

A proof idea to this conjecture is given in Appendix A.1. I still assume that Conjecture 10 or something similar holds. The following remark gives an overview of the assumptions made for this proof as well as the problems that occurred with different attempts. This suggests that it might be necessary to apply different proof techniques than used in [4, Lemma 1] or [1, Lemma 5].

Remark 11. The technique used to bound the expected amount of transcripts required for extraction as introduced by Bootle et al. [4] works as follows:

In the setting of [4], for every node the amount of nodes required for short-circuit extraction is fetched. This can be modelled as burst extraction with $k_i = \hat{k}_i$ and no bursts, i.e. $\beta \equiv 0$ To estimate the expected amount of lookups within a call to $\text{BurstTree}_m(a, c_1, \ldots, c_m)$ with $v = (a, c_1, \ldots, c_m)$, the expected amount of calls to $\text{BurstTree}_{m+1}(a, c_1, \ldots, c_m, \cdot)$ is determined. For this, two different elements have to be considered: The probability to enter a certain node v is $\mathbf{Pr}[E_v] = \epsilon = \epsilon(a, c_1, \ldots, c_m)$. The probability for a fixed challenge c_{m+1} is defined as $\epsilon(a, c_1, \ldots, c_m, c_{m+1})$. Therefore the probability of extraction to fail for a uniformly random challenge X is

$$\epsilon = \epsilon(a, c_1, \ldots, c_m) = \sum_{c_{m+1} \in C} \mathbf{Pr}[X = c_{m+1}] \; \epsilon(a, c_1, \ldots, c_m, c_{m+1})$$

When k nodes are required by the extractor, the expected number of times $\textsc{BurstTree}_m$ runs $\textsc{BurstTree}_{m+1}$ is $1 + \epsilon \cdot {}^{(k-1)}/_{\epsilon} = k$.

When restricting β to $\beta \equiv 0$ there are two different results from $\textsc{BurstTree}_{m+1}$: Either, regular extraction was successful, or extraction failed and only one lookup was required. With bursts, we have a third possible option: A burst occurred, and therefore more children have to be extracted, but extraction can be stopped at this point.

The burst is modelled by a predicate β that receives the whole extracted subtree as an input, and decides on this basis if the subtree is marked as burst. As a consequence of this modelling, nothing else about the burst is known. This makes this third possible result difficult to handle.

The proof sketch for Conjecture 10 was based on the following idea: As in [4], the amount of lookups in H is bounded for subtrees with their root in certain layers. For leaves this amount can be determined easily, so that we can inductively calculate the amount for larger subtrees. In this setting a third type of result has to be handled: The extraction algorithm may find a burst subtree. At some points of the proof idea, different cases with burst events at varying locations are considered, which then have to be weighted. Whenever there are two different expected bounds for the expected lookups in H for different situations, that have to be weighted by the probability of a burst event, this proof used the maximum of expected values.

Instead of only one induction for the expected amount of lookups in H, an induction was done for two different functions: $n_{\neg\mathsf{burst}}(m)$ tracks an upper bound for the expected amount of lookups to H for subtrees τ with its root in layer m of the tree if no bursts occurred in the extraction of τ. The other function $n_{\mathsf{burst}}(m)$ denotes an upper bound on the expected amount of looks to H if a burst has occurred within that subtree.

This distinction is necessary for linear extraction because a subtree with successful extraction is cheaper than one with a burst in expectation, and we would get results in terms of $\hat{k}_i$ instead of k_i as in Conjecture 10 otherwise.

One problem arises when looking at nodes of the tree $v = (a, c_1, \ldots, c_m)$ where $C_v = 1$. This random variable C_v signals that in one of the child subtrees of v there must have been a burst as illustrated in Figure 3.4. As before, we have a probability of entering the node is $\epsilon = \epsilon(a, c_1, \ldots, c_m)$. When distinguishing $n_{\mathsf{burst}}(m)$ and $n_{\neg\mathsf{burst}}(m)$ we may have different probabilities of the extraction succeeding, say $\epsilon_{\mathsf{burst}}$ and $\epsilon_{\neg\mathsf{burst}}$. Because of this there is no easy cancellation of ϵ as before. This suggests that functions for a bound on the expected number of lookups in H that are only dependent on the layer of the node like n_{burst} can not be used here.

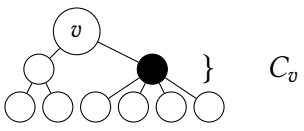


Figure 3.4.: C_v

This suggests that an approach very different from [4, Lemma 1] or [1, Lemma 5] has to be used.

In the following section, the Bulletproof Inner Product Argument will be introduced. The sampling pattern required for its extraction matches burst extraction introduced in this chapter.

4. Inner Product Argument

In the following chapter a modified version of the Bulletproof Inner Product Argument [5] with changes by Hoffmann, Klooß, and Rupp [8] is introduced. The purpose of the Inner Product Argument Protocol is to convince the Verifier $\mathscr{V}$ that the prover $\mathscr{P}$ has knowledge of two vectors that have been committed to, and that those two vectors have a certain inner product. We will focus on the extraction, especially on quick extraction, which allows us to use fewer transcript if extraction is successful.

To facilitate the understanding of the protocol, we first introduce a Vector Knowledge Argument VKA, which can be seen as a component of the Inner Product Argument and already contains many ideas and techniques used. Afterwards, this protocol is extended to the Inner Product Argument by Bünz et al. [5] with the improvements by Hoffmann, Klooß, and Rupp [8]. Afterwards, the connection between the version of IPA presented here, Bünz et al. [5] and Hoffmann, Klooß, and Rupp [8] is illustrated. As a last step, we show how to do quick extraction for the Range Proof and the Arithmetic Circuit Satisfiability Proof introduced in the Bulletproof Paper [5] and thereby use fewer transcripts for extraction.

4.1. Notation

In the following protocols, we will use an abelian group $\mathbb{G}$, for which the Discrete Logarithm Relation Assumption holds. We denote the group additively with coefficients from the field $\mathbb{F}_p = \mathbb{Z}/\mathbb{Z}_p$, as was already explained in 2.1.

A special notation that will be useful when describing the "folding" of vectors in the recursive steps of the protocols is $\vec{\mathbf{w}}$: Here, a vector $\mathbf{w} \in \mathbb{F}_p^n$ (or $\mathbf{g} \in \mathbb{G}^{1\times n}$) is split into k parts of equal size $\mathbb{F}_p^{n/k}$ (or $\mathbb{G}^{1\times n/k}$, respectively). We denote those parts by $\vec{\mathbf{w}}_1, \ldots, \vec{\mathbf{w}}_k \in \mathbb{F}_p^{n/k}$ and $\vec{\mathbf{g}}_1, \ldots, \vec{\mathbf{g}}_k \in \mathbb{G}^{1\times n/k}$. Those parts can be reassembled to a vector of vectors $\vec{\mathbf{w}}$, which contain the same entries as $\mathbf{w}$:

$$\vec{\mathbf{w}} = \begin{bmatrix} \vec{\mathbf{w}}_1 \\ \vdots \\ \vec{\mathbf{w}}_k \end{bmatrix} \in \left(\mathbb{F}_p^{n/k}\right)^k \qquad \vec{\mathbf{g}} = \begin{bmatrix} \vec{\mathbf{g}}_1 & \cdots & \vec{\mathbf{g}}_k \end{bmatrix} \in \left(\mathbb{G}^{1\times n/k}\right)^{1\times k} \tag{4.1}$$

Note that because the entries $\vec{\mathbf{w}}_i$ of $\vec{\mathbf{w}}$ are vectors themselves, the indexed entries are bold as well, whereas for an $\mathbf{x} \in \mathbb{F}_p^k$ the entries are denoted by $x_i \in \mathbb{F}_p$.

4.2. Vector Knowledge Argument

Instead of beginning with the complete Inner Product Argument, we start off with a simpler Vector Knowledge Argument, which will later be used as a part of the complete protocol: Given

a vector $\mathbf{w} \in \mathbb{F}_p^n$ in possession of prover $\mathscr{P}$ that has been "committed" to (using Pedersen Commitments without the blinding factor, which thereby loose their Hiding-Property) in $[c] \in \mathbb{G}$. For those "commitments", we have a vector known to both the verifier $\mathscr{V}$ and the prover $\mathscr{P}$ called $\mathbf{g} \in \mathbb{G}^n$. The commitment $[c]$ is calculated by $[c] = [\mathbf{g}]\,\mathbf{w}$. We can denote the statement to be shown as the relation $\mathcal{R}_{\mathsf{VKA}}$, with $[c]$ being the statement and $\mathbf{w}$ the witness for membership in this relation.

$$([c]\,;\mathbf{w}) \in \mathcal{R}_{\mathsf{VKA}} \Leftrightarrow [c] \in \mathbb{G} \wedge \mathbf{w} \in \mathbb{F}_p^n \wedge [c] = [\mathbf{g}]\,\mathbf{w}$$

The trivial protocol to prove membership in this relation is the prover $\mathscr{P}$ just sending $\mathbf{w}$ to the verifier $\mathscr{V}$, which then checks if $[c] = [\mathbf{g}]\,\mathbf{w}$. One problem of this solution is that it requires sending n elements of $\mathbb{F}_p^n$ to $\mathscr{V}$, whereas we would prefer communication to be in $O(\log(n))$. To achieve this goal, we use the following ideas:

4.2.1. Recursively Shrinking the Statement

For a given reduction factor $k \geq 2$, we "fold" the statement of initial size $n = k^d$ for some $d \in \mathbb{N}^\times$ to a smaller statement of size $\hat{n} = k^{d-1}$. Starting off with $[c] = \mathbf{w}\,[\mathbf{g}]$ with $\mathbf{w} \in \mathbb{F}_p^n$, we want to output a $[\hat{c}] = \hat{\mathbf{w}}\,[\hat{\mathbf{g}}]$ with a matching $\hat{\mathbf{w}} \in \mathbb{F}_p^{n/k}$.

One strategy to reduce the statement size is to batch multiple equations into one by combining the single equations with random factors and adding those up afterwards. This technique can not be applied here, because $\sum_{i=1}^{k} \left[\vec{\mathbf{g}}_i\right] \vec{\mathbf{w}}_i = [c]$ is a single equation with $[c] \in \mathbb{G}$. Therefore, we need to embed our problem into a larger one to make batching possible:

4.2.2. Embedding the Problem

For this, we introduce the variables $\left[u_{i,j}\right] = \left[\vec{\mathbf{g}}_i\right] \vec{\mathbf{w}}_j$, which can be assembled as a matrix $[\mathbf{U}] = \left[u_{i,j}\right]_{i,j\in\{1,\dots k\}}$:

$$\begin{bmatrix}\vec{\mathbf{g}}_1 & \cdots & \vec{\mathbf{g}}_k\end{bmatrix} \cdot \begin{pmatrix}\vec{\mathbf{w}}_1 \\ \vdots \\ \vec{\mathbf{w}}_k\end{pmatrix} = \begin{bmatrix}\vec{\mathbf{g}}_1\vec{\mathbf{w}}_1 & \vec{\mathbf{g}}_2\vec{\mathbf{w}}_1 & \cdots & \vec{\mathbf{g}}_k\vec{\mathbf{w}}_1 \\ \vec{\mathbf{g}}_1\vec{\mathbf{w}}_2 & \vec{\mathbf{g}}_2\vec{\mathbf{w}}_2 & \cdots & \vec{\mathbf{g}}_k\vec{\mathbf{w}}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \vec{\mathbf{g}}_1\vec{\mathbf{w}}_k & \vec{\mathbf{g}}_2\vec{\mathbf{w}}_k & \cdots & \vec{\mathbf{g}}_k\vec{\mathbf{w}}_k\end{bmatrix} = [\mathbf{U}] \in \mathbb{G}^{k\times k} \tag{4.2}$$

In this larger problem, the original statement $[c]$ is the trace of the matrix $\mathrm{tr}([\mathbf{U}]) = \sum_{i=1}^{k} \left[\vec{\mathbf{g}}_i\right] \vec{\mathbf{w}}_i = [c]$. This larger problem can now be batched as follows: A random challenge $\xi \leftarrow\!\!\$\, \mathbb{F}_p$ is chosen, and the different equations are added up with different powers of ξ. As the coefficients, we will use the vectors

$$\mathbf{x} = \begin{pmatrix}1 \\ \xi \\ \xi^2 \\ \vdots\end{pmatrix} \qquad\qquad \mathbf{y} = \begin{pmatrix}1 \\ \xi^{-1} \\ \xi^{-2} \\ \vdots\end{pmatrix}$$

The vectors $\mathbf{w}$ and $\mathbf{g}$ can then be split into k parts as in $\vec{\mathbf{w}}$ and $\vec{\mathbf{g}}$. Then, they are folded by multiplying each component $\vec{\mathbf{w}}_i$ with the respective coefficient y_i and added up, which can be described by the vector multiplication $\hat{\mathbf{w}} = \mathbf{y}^\mathsf{T} \cdot \vec{\mathbf{w}}$. For $\mathbf{g}$ we proceed analogously with $\hat{\mathbf{g}} = \mathbf{x}^\mathsf{T} \cdot \vec{\mathbf{g}}$, which gives us the new, reduced statement

$$\mathbf{y}^\mathsf{T}\left(\begin{pmatrix}\vec{\mathbf{w}}_1\\ \vdots \\ \vec{\mathbf{w}}_k\end{pmatrix} \cdot \begin{bmatrix}\vec{\mathbf{g}}_1 & \cdots & \vec{\mathbf{g}}_k\end{bmatrix}\right)\mathbf{x} = \underbrace{\left(\mathbf{y}^\mathsf{T}\begin{pmatrix}\vec{\mathbf{w}}_1\\ \vdots \\ \vec{\mathbf{w}}_k\end{pmatrix}\right)}_{=\sum_{i=1}^{k} y_i \mathbf{w}_i = \hat{\mathbf{w}}} \cdot \underbrace{\left(\begin{bmatrix}\vec{\mathbf{g}}_1 & \cdots & \vec{\mathbf{g}}_k\end{bmatrix}\mathbf{x}\right)}_{=\sum_{i=1}^{k} x_i \vec{\mathbf{g}}_i = \hat{\mathbf{g}}}$$

To convince the verifier that the folding was done correctly, the prover $\mathscr{P}$ then calculates all $\left[u_{i,j}\right] = \mathbf{w}_i \left[\mathbf{g}_i\right]$ for $i, j \in \{1, \ldots, k\}$, sends them to the verifier and receives a random challenge ξ from the verifier. Both verifier and prover can now calculate the new statement $[\hat{c}] = [\mathbf{g}\mathbf{x}]\, \mathbf{y}^\mathsf{T}\mathbf{w} = \sum_{i=1}^{k}\sum_{j=1}^{k} x_i y_j \left[u_{i,j}\right]$ along with the shrunken witness $\hat{\mathbf{w}} = \mathbf{y}^\mathbf{T}\vec{\mathbf{w}}$ and the new vector of group elements $[\hat{\mathbf{g}}] = \left[\vec{\mathbf{g}}\right]\mathbf{x}$.

Using this technique, we can reduce the size of the statement to constant size in $d = \log_k(n)$ rounds, but k^2 elements need to be sent in every step. This factor can be reduced as follows:

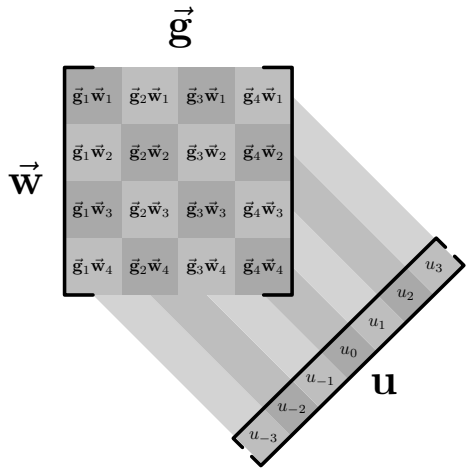


Figure 4.1.: Relation between $\mathbf{g}$, $\mathbf{w}$ and $\mathbf{u}$

4.2.3. Sending the Diagonals

When considering the coefficients of the $\left[u_{i,j}\right]$, we notice that for $\left[u_{i,j}\right]$ with identical difference of the components $\ell = i - j$, we have identical exponents of ξ:

$$\begin{aligned}[\hat{c}] &= \sum_{i=1}^{k}\sum_{j=1}^{k} x_i y_j \left[u_{i,j}\right] = \sum_{i=1}^{k}\sum_{j=1}^{k} \xi^{i-1}\xi^{-j+1}\left[u_{i,j}\right] \\ &= \sum_{i=1}^{k}\sum_{j=1}^{k} \xi^{i-j}\left[u_{i,j}\right] = \sum_{\ell=-k+1}^{k-1} \xi^{-\ell} \sum_{\ell=j-i} \left[u_{i,j}\right]\end{aligned}$$

Therefore, the prover $\mathscr{P}$ can reduce the amount of communication by summing up the $\left[u_{i,j}\right]$ to $[u_\ell] = \sum_{i-j=\ell} u_{i,j}$ for $\ell \in \{-k+1, \ldots, k-1\}$. For a vector notation of this folding, we collect the u_ℓ in the vector $\mathbf{u} = \begin{bmatrix}u_{-k+1} & \cdots & u_{k-1}\end{bmatrix} \in \mathbb{G}^{1\times(2k-1)}$, and write down the coefficients in $\mathbf{z}$ defined by $z_{j-i} = x_i y_j = \xi^{-\ell}$:

$$\mathbf{z} = \begin{pmatrix}\xi^{k-1} & \ldots & \xi^{-k+1}\end{pmatrix} \tag{4.3}$$

The different u_i correspond to the different second-diagonals from the matrix $\vec{\mathbf{w}}\vec{\mathbf{g}}$ from Equation (4.2), the relationship between those is visualized in Figure 4.1. Note that now also

$[u_0] = [c]$ doesn't have to be sent, since it is already known as the statement. For convenience, we will index the entries of $[\mathbf{u}] \in \mathbb{G}^{1\times(2k-1)}$ and $\mathbf{z} \in \mathbb{F}_p^{2k-1}$ with $i \in \{-k+1, \ldots, k-1\}$. Instead of sending k^2 elements, we are now down to $2k-2$ group elements to be sent in every iteration of the recursion.

4.2.4. Vector Knowledge Argument Protocol

Combining the techniques presented in the previous sections, we can now state the Vector Knowledge Argument VKA, which corresponds to Hoffmann, Klooß, and Rupp [8, Protocol 3.9]:

Protocol 12 (Vector Knowledge Argument)**.**
Recursive Step. Suppose $n = k^d$ with $d \geq 1$:

$\mathscr{P}$: Compute $[\mathbf{u}_\ell] = \sum_{j-i=\ell} \left[\mathbf{g}_i\right] w_j$.
$\mathscr{P} \to \mathscr{V}$: Send $[\mathbf{u}] \in \mathbb{G}^{1\times 2k-1}$.
Note that only $2k-2$ elements have to be sent, because $[u_0] = [c]$ is already known.
$\mathscr{V} \to \mathscr{P}$: Pick and send the challenges $\xi \leftarrow\!\!\$\, \mathbb{F}_p^n \setminus \{0\}$.
$\mathscr{P}, \mathscr{V}$: Calculate the coefficients $\mathbf{x} = \begin{pmatrix} 1 \\ \xi \\ \xi^2 \\ \vdots \end{pmatrix}$, $\mathbf{y} = \begin{pmatrix} 1 \\ \xi^{-1} \\ \xi^{-2} \\ \vdots \end{pmatrix}$ and $\mathbf{z} = \begin{pmatrix} \xi^{k-1} \\ \vdots \\ \xi^{-k+1} \end{pmatrix}$.
Compute $[\hat{\mathbf{g}}] = \mathbf{x}^\mathsf{T} \left[\vec{\mathbf{g}}\right]$ and $[\hat{c}] = [\mathbf{u}]\, \mathbf{z}$.

$\mathscr{P}$: Compute $\hat{\mathbf{w}} = \mathbf{y}^\mathsf{T} \vec{\mathbf{w}}$.
$\mathscr{P}, \mathscr{V}$ :: Set $[\mathbf{w}] = [\hat{\mathbf{w}}]$, $[\mathbf{g}] := [\hat{\mathbf{g}}]$ and $[c] := [\hat{c}]$ — and repeat the recursive step for $n := n/k$

Invariant. $[c] = [\mathbf{g}]\, \mathbf{w}$
Base Case.

$\mathscr{P} \to \mathscr{V}$: Send $\mathbf{w}$.
$\mathscr{V}$: Verify that $[\mathbf{g}]\, \mathbf{w} \stackrel{?}{=} [c]$ and accept if verification was successful.

In this protocol, the number of group elements required amounts to $(2k-2)\log_k(n)$. To minimize the amount of group elements sent, the protocol is optimal for $k = 2$, because $(2k-2)\log_k(n) = {}^{2k-2}/_{\log_2(k)} \cdot \log_2(n)$ with ${}^{2k-2}/_{\log_2(k)}$ being minimal for smallest possible value $k = 2$.

The protocol is complete, because the invariant $[\mathbf{g}]\, \mathbf{w} = [c]$ is maintained by the recursive folding step as the following calculation proves:

$$\begin{aligned}[\hat{c}] = [u_{-k+1}, \ldots, u_{k-1}]\, \mathbf{z} &= \sum_{\ell=-k+1}^{k-1} \left(z_\ell \sum_{i-j=\ell} \left[\vec{\mathbf{g}}_i\right] \vec{\mathbf{w}}_j \right) \\ &= \sum_{\ell=-k+1}^{k-1} \left(\sum_{i-j=\ell} \left[x_i \vec{\mathbf{g}}_i\right] y_j \vec{\mathbf{w}}_j \right) = \left(\sum_{i=1}^{k} x_i \left[\vec{\mathbf{g}}_i\right] \right) \left(\sum_{j=1}^{k} y_j \vec{\mathbf{w}}_j \right) = [\hat{\mathbf{g}}]\, \hat{\mathbf{w}} \end{aligned} \tag{4.4}$$

Vector Knowledge Protocol (Protocol 12)	
Common Input: $[\mathbf{g}]$	
Prover $\mathscr{P}$ Input: $\mathbf{w}$	Verifier $\mathscr{V}$ Input: $[c]$
Recursive Step. Suppose $n = k^d$ with $d \geq 1$	
$[u_\ell] = \sum_{j-i=\ell} [\mathbf{g}_i] w_j$ for $\ell \in \{-k+1, \ldots, k-1\}$	
$\xrightarrow{[\mathbf{u}]}$	
	$\xi \leftarrow \mathbb{F}_p \setminus \{0\}$ and therefore $\mathbf{x} = \begin{pmatrix} 1 \\ \xi \\ \xi^2 \\ \vdots \end{pmatrix}$, $\mathbf{y} = \begin{pmatrix} 1 \\ \xi^{-1} \\ \xi^{-2} \\ \vdots \end{pmatrix}$ and $\mathbf{z} = \begin{pmatrix} \xi^{k-1} \\ \vdots \\ \xi^{-k+1} \end{pmatrix}$
$\xleftarrow{\xi}$	
$[\mathbf{g}] = \mathbf{x}^\mathsf{T} [\vec{\mathbf{g}}]$ $[c] = \mathbf{z}^\mathsf{T} [\mathbf{u}]$ $\mathbf{w} = \mathbf{y}^\mathsf{T} \vec{\mathbf{w}}$ $n = n/k$	$[\hat{\mathbf{g}}] = \mathbf{x}^\mathsf{T} [\vec{\mathbf{g}}]$ $[\hat{c}] = \mathbf{z}^\mathsf{T} [\mathbf{u}]$ $n = n/k$
Start next recursion iteration.	
Base Case. Suppose $n = 1$	
$\xrightarrow{\mathbf{w}}$	
	return $[\mathbf{g}]\,\mathbf{w} \stackrel{?}{=} [c]$

Table 4.1.: Vector Knowledge Argument Protocol VKA, resembling [8, Protocol 3.9]

The knowledge soundness can be derived from the following analysis of the recursion step, whose proof and statement closely resemble [8, Lemma 3.10]:

Lemma 13 (Recursive Extraction)**.** *Using the variables and setting as defined in Protocol 12, the following statements hold for one recursive step of the protocol:*

1. *Given a non-trivial discrete logarithm relation $\hat{v}$ of $[\hat{\mathbf{g}}]$ folded using $\boldsymbol{x}$, we efficiently find a non-trivial discrete logarithm relation v within $[\boldsymbol{g}]$.*
2. (Unconditional Extraction) *Given $2k-1$ distinct challenges $\xi^{(1)}, \ldots, \xi^{(2k-1)}$ with accepting transcripts, one can extract unconditionally a witness $\boldsymbol{w}$ such that $[c] = [\boldsymbol{g}]\ \boldsymbol{w}$.*
3. (Short-Circuit Extraction) *Given $2k$ distinct challenges $\xi^{(1)}, \ldots, \xi^{(2k)}$ with accepting transcripts if the witness from above does not fit w.r.t. the $[u_\ell]$, i.e. if an honest prover would send different $[u_\ell]$ for $\boldsymbol{w}$, then we find (additionally) a non-trivial discrete logarithm relation $\boldsymbol{v}$, i.e. $0 = [\boldsymbol{g}]\ \boldsymbol{v}$ with $\boldsymbol{v} \neq 0$.*
4. (Quick Extraction) *From k distinct challenges $\xi^{(1)}, \ldots, \xi^{(k)}$ with accepting transcripts, one can compute a candidate witness $\boldsymbol{w}$ for quick extraction. If $\sum_{j-i=\ell} \left[\boldsymbol{g}_i\right] \boldsymbol{w}_j \neq [u_\ell]$ for some ℓ, then we are guaranteed to find a non-trivial discrete logarithm relation from $2k$ distinct challenges.*

.

Proof. For the first statement, we consider a non-trivial kernel element $\hat{\mathbf{v}} \neq 0$ with $0 = [\hat{\mathbf{g}}]\ \hat{\mathbf{v}}$. To see how to extend $\hat{\mathbf{v}}$, we make the following calculation:

$$0 = [\hat{\mathbf{g}}]\ \hat{\mathbf{v}} = \sum_{i=1}^{k} x_i \left[\vec{\mathbf{g}}_i\right] \hat{\mathbf{v}} = \sum_{i=1}^{k} \left[\vec{\mathbf{g}}_i\right] (x_i \cdot \hat{\mathbf{v}}) = \left[\vec{\mathbf{g}}\right] \underbrace{\begin{pmatrix} x_1\hat{\mathbf{v}} \\ x_2\hat{\mathbf{v}} \\ \vdots \\ x_k\hat{\mathbf{v}} \end{pmatrix}}_{\mathbf{x}\otimes\hat{\mathbf{v}}} = \left[\vec{\mathbf{g}}\right] (\mathbf{x} \otimes \hat{\mathbf{v}}) \tag{4.5}$$

$\mathbf{x} \otimes \hat{\mathbf{v}} \in \left(\mathbb{F}^{n/k}\right)^k$ is the tensor product of $\mathbf{x}$ and $\hat{\mathbf{v}}$ and by Equation (4.5) lies within $\mathbf{x} \otimes \hat{\mathbf{v}} \in \mathrm{Ker}\left(\left[\vec{\mathbf{g}}\right]\right)$, proving the first statement.

For the Unconditional Extraction, we use the equation for calculating the next $[\hat{c}]$ in the verifier $\mathscr{V}$ and — analogous to Equation (4.5) — use the following statement:

$$[\mathbf{u}]\ \mathbf{z}^{(i)} = \begin{bmatrix} u_{-k+1} & \cdots & u_{k-1} \end{bmatrix} \mathbf{z}^{(i)} = [\hat{\mathbf{g}}]\ \hat{\mathbf{w}} = \left(\sum_{i=1}^{k} x_i \left[\vec{\mathbf{g}}_i\right]\right) \hat{\mathbf{w}} = \left[\vec{\mathbf{g}}\right] (\mathbf{x} \otimes \hat{\mathbf{w}}) \tag{4.6}$$

To combine multiple challenges into one equation, we define the matrices $\hat{\mathbf{W}}$ and $\mathbf{Z}$ as follows:

$$\hat{\mathbf{W}} = \begin{bmatrix} \vdots & & \vdots \\ \mathbf{x}^{(1)} \otimes \hat{\mathbf{w}}^{(1)} & \cdots & \mathbf{x}^{(2k-1)} \otimes \hat{\mathbf{w}}^{(2k-1)} \\ \vdots & & \vdots \end{bmatrix} \qquad \mathbf{Z} = \begin{bmatrix} \vdots & & \vdots \\ \mathbf{z}^{(1)} & \cdots & \mathbf{z}^{(2k-1)} \\ \vdots & & \vdots \end{bmatrix}$$

So using $\hat{\mathbf{W}}$ to combine Equation (4.6) for all $2k-1$ challenges as each a column into one matrix equation gives us:

$$\begin{bmatrix} u_{-k+1} & \cdots & u_{k-1} \end{bmatrix} \cdot \mathbf{Z} = [\mathbf{g}]\,\hat{\mathbf{W}} \Leftrightarrow \begin{bmatrix} u_{-k+1} & \cdots & u_{k-1} \end{bmatrix} = [\mathbf{g}] \underbrace{\hat{\mathbf{W}} \cdot \mathbf{Z}^{-1}}_{\mathbf{W}} = [\mathbf{g}]\,\mathbf{W} \tag{4.7}$$

Note that inverting $\mathbf{Z}$ is possible, because $\mathbf{Z}$ is the Vandermonde matrix for the distinct challenges $\xi^{(1)}, \xi^{(2)}, ..., \xi^{(k)}$, which has been scaled in every row by $(\xi^{(i)})^{-k+1}$ and then transposed. Because of the structure of $\hat{\mathbf{W}}$, with $\mathbf{W}$ we have again a matrix of matrices:

$$\mathbf{W} \in \left(\mathbb{F}_p^{n/k}\right)^{k \times (2k-1)}$$

We name the columns of $\mathbf{W} = (\ \mathbf{w}^{(-k+1)} \ \cdots \ \mathbf{w}^{(0)} \ \cdots \ \mathbf{w}^{(k-1)}\ )$ and therefore have vectors of dimension $(\mathbb{F}_p^{n/k})^k$ with each of the columns being in $\mathbf{w}^{(i)} \in (\mathbb{F}_p^{n/k})^k$. When omitting all the columns except the one with index 0 from the equation, we have $[c] = [u_0] = \left[\vec{\mathbf{g}}\right] \mathbf{w}^{(0)}$, giving us unconditional extraction with $\vec{\mathbf{w}} = \mathbf{w}^{(0)}$. This concludes the proof of the second statement.

For the Short-Circuit Extraction, we want to show that for a given $\mathbf{w}$ that fulfils the equation from the relation $[c] = [\mathbf{g}]\,\mathbf{w}$, we either have $[u_\ell] = \sum_{j-i=\ell} \left[\mathbf{g}_i\right] \mathbf{w}_j$ or find a non-trivial discrete logarithm relation for $[\mathbf{g}]$. For this, we prove that $\mathbf{W}$ has a fixed structure using the equations applied by the verifier, and then infer that this also implies a correctly calculated $[\mathbf{u}]$. First we derive one equation about the columns $\mathbf{w}^{(i)}$ of $\mathbf{W}$ by finding two different ways to rephrase $[\mathbf{u}]\,\mathbf{z}$:

From Equation (4.7) derived from $[\mathbf{u}]\,\mathbf{z} = [\hat{\mathbf{g}}]\,\hat{\mathbf{w}}$ in the proof for the second statement, we have that $[\mathbf{u}] = [\mathbf{g}]\,\mathbf{W}$, and therefore that $[\mathbf{u}]\,\mathbf{z}^{(i)} = [\mathbf{g}]\,\mathbf{W}\mathbf{z}^{(i)}$. From the invariant of the protocol and the re-calculation of $[\hat{c}]$ we also have that $[\hat{\mathbf{g}}]\,\hat{\mathbf{w}} = [\hat{c}] = [\mathbf{u}]\,\mathbf{z}$. Combining those two equations, we see that

$$[\mathbf{g}]\,\mathbf{W}\mathbf{z}^{(i)} = [\mathbf{g}]\left(\mathbf{x}^{(i)} \otimes \hat{\mathbf{w}}^{(i)}\right).$$

Therefore, either the next equation holds, or we have found a non-trivial kernel of $[\mathbf{g}]$:

$$\mathbf{W}\mathbf{z}^{(i)} = \sum_{\ell=-k+1}^{k-1} z_\ell^{(i)} \mathbf{w}^{(\ell)} = \mathbf{x}^{(i)} \otimes \hat{\mathbf{w}}^{(i)}$$

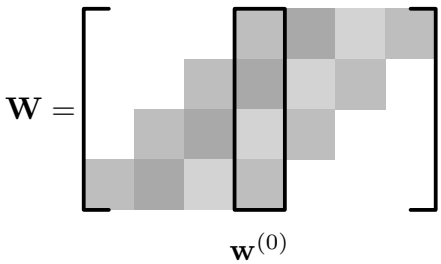


Figure 4.2.: The structure of W

We now look at the components $j \in \{1, \ldots, k\}$ of the previous equation:

$$\begin{aligned}
\sum_{\ell=-k+1}^{k-1} z_\ell^{(i)} \mathbf{w}_j^{(\ell)} = x_j^{(i)} \hat{\mathbf{w}}^{(i)} \qquad &\Leftrightarrow \\
\sum_{\ell=-k+1}^{k-1} \left(z_\ell^{(i)} \cdot \left(x_j^{(i)} \right)^{-1} \right) \mathbf{w}_j^{(\ell)} = \hat{\mathbf{w}}^{(i)} \qquad &\Leftrightarrow \\
\sum_{\ell=-k+1}^{k-1} \left(\left(\xi^{(i)} \right)^{-\ell} \left(\left(\xi^{(i)} \right)^{j-1} \right)^{-1} \right) \mathbf{w}_j^{(\ell)} = \sum_{\ell=-k+1}^{k-1} \left(\xi^{(i)} \right)^{-\ell-j+1} \mathbf{w}_j^{(\ell)} = \hat{\mathbf{w}}^{(i)} \qquad &\Leftrightarrow \\
\sum_{\ell=-k+1}^{k-1} y_{j+\ell}^{(i)} \cdot \mathbf{w}_j^{(\ell)} = \hat{\mathbf{w}}^{(i)} &
\end{aligned}$$

Recall that we extended $\mathbf{y}$ with $y_i = \xi^{-i+1}$ outside its definition radius as well. Comparing two adjacent rows of vectors j and $j+1$ for $j \in \{1, \ldots, k-1\}$, we get:

$$\sum_{\ell=-k+1}^{k-1} y_{j+\ell}^{(i)} \cdot \mathbf{w}_j^{(\ell)} = \sum_{\ell=-k+1}^{k-1} y_{j+1+\ell}^{(i)} \cdot \mathbf{w}_{j+1}^{(\ell)} = \sum_{\ell=-k+2}^{k} y_{j+\ell}^{(i)} \cdot \mathbf{w}_{j+1}^{(\ell-1)}$$

We have $2k$ different entries of $\mathbf{y}^{(i)}$. So that we can recover the coefficients:

$$\begin{aligned}
\forall j \in \{1, \ldots, k-1\} : \mathbf{w}_{j+1}^{(-k+1)} = 0 = \mathbf{w}_j^{(k-1)} \\
\forall j \in \{1, \ldots, k-1\} \, \forall \ell \in \{-k+2, \ldots, k-1\} : \mathbf{w}_{j+1}^{(\ell-1)} = \mathbf{w}_j^{(\ell)}
\end{aligned}$$

By induction, we prove that all entries of $\mathbf{W}$ are either 0 or parts of $\mathbf{w}^{(0)}$:

$$\mathbf{w}_j^{(i)} = \begin{cases} \mathbf{w}_{j+i}^{(0)} & 0 < j + i \leq k \\ 0 & \text{otherwise} \end{cases}$$

Those relations are also represented graphically in Figure 4.2. Defining $\mathbf{w}_j^{(0)} = 0$ for $j \notin \{1, \ldots, k\}$, we can derive:

$$\sum_{\ell=-k+1}^{k-1} \mathbf{y}_{j+\ell}^{(i)} \cdot \mathbf{w}_j^{(\ell)} = \sum_{\ell=-k+1}^{k-1} \mathbf{y}_{j+\ell}^{(i)} \cdot \mathbf{w}_{j+\ell}^{(0)} = \hat{\mathbf{w}}^{(i)}$$

Now we recall that according to Equation (4.7), we have that $[\mathbf{u}] = [\mathbf{g}]\,\mathbf{W}$ — so the $[u_\ell]$ are fixed by the choice of $\mathbf{w}$ if we have not found a non-trivial discrete logarithm relation along the way. This conlcudes the third part of the proof.

To prove quick extraction, we show how to recover $\vec{\mathbf{w}}$ using only k challenges: For this, we recall how $\hat{\mathbf{w}}$ is calculated by an honest prover in the protocol: $\hat{\mathbf{w}} = \vec{\mathbf{w}}^\mathsf{T} \mathbf{y}$. Combining this equation for k different challenges $\xi^{(i)}$ and therefore k different $\mathbf{y}^{(i)}$, we get:

$$\left(\hat{\mathbf{w}}^{(1)} \quad \hat{\mathbf{w}}^{(2)} \quad \cdots \quad \hat{\mathbf{w}}^{(k)}\right) = \vec{\mathbf{w}}^{\mathsf{T}} \underbrace{\left(\mathbf{y}^{(1)} \quad \mathbf{y}^{(2)} \quad \cdots \quad \mathbf{y}^{(k)}\right)}_{=\mathbf{Y}}$$

$$\left(\hat{\mathbf{w}}^{(1)} \quad \hat{\mathbf{w}}^{(2)} \quad \cdots \quad \hat{\mathbf{w}}^{(k)}\right) \cdot \mathbf{Y}^{-1} = \vec{\mathbf{w}}^{\mathsf{T}} \tag{4.8}$$

Afterwards we verify that we have recovered a valid witness $\mathbf{w}$ by $[c] \stackrel{?}{=} [\mathbf{g}]\,\mathbf{w}$. When calculating $\hat{\mathbf{w}}$ as described in Protocol 14, equation (4.8) holds as well — and since multiplication by $\mathbf{Y}$ is bijective, both vectors have to be identical. If not, the $[u_\ell]$ sent by the prover $\mathscr{P}$ must have been chosen in a different way, yielding a break of the discrete logarithm assumption by the previous statement. □

Now that we have understood the Vector Knowledge Argument-Protocol, we can use it in the following section to assemble the Inner Product Protocol, which form part of the Bulletproof Proof System [5].

4.3. Inner Product Argument

Instead of proving knowledge of one vector as in the Vector Knowledge Argument-Protocol, we now want to do so for two vectors with a certain inner product. Because of this, we now need two vectors of group elements $[\mathbf{g}'], [\mathbf{g}''] \in \mathbb{G}^n$ along with an additional element $[Q] \in \mathbb{G}$, so in total we have the "commitment key" $[\mathbf{g}', \mathbf{g}'', Q] \in \mathbb{G}^{2n+1}$.

Our commitment $[c]$ now contains, in addition to $\mathbf{w}'$ and $\mathbf{w}''$, the supposed inner product $t = \langle \mathbf{w}', \mathbf{w}'' \rangle$ as $[c] = [\mathbf{g}']\,\mathbf{w}' + [\mathbf{g}'']\,\mathbf{w}'' + t\,[Q]$. As before, we can also state this as a relation $\mathcal{R}_{\mathsf{IPA}}$:

$$\begin{aligned}([c], t; \mathbf{w}', \mathbf{w}'') \in \mathcal{R}_{\mathsf{IPA}} \Leftrightarrow [c] \in \mathbb{G} \wedge \mathbf{w}', \mathbf{w}'' \in \mathbb{F}_p^n \wedge t \in \mathbb{F}_p \wedge \\ [c] = [\mathbf{g}']\,\mathbf{w}' + [\mathbf{g}'']\,\mathbf{w}'' + t\,[Q] \wedge \langle \mathbf{w}', \mathbf{w}'' \rangle = t\end{aligned} \tag{4.9}$$

So $[c]$ can be seen as three merged Pedersen Commitments to $\mathbf{w}'$, $\mathbf{w}''$ and t without the blinding factors. As before, one could successfully prove membership in $\mathcal{R}_{\mathsf{IPA}}$ by just handing over the $\mathbf{w}'$ and $\mathbf{w}''$, and verifying that both the calculation of $[c]$ and the scalar product t are correct. As before, we would like to reduce communication for this procedure:

4.3.1. Combination of Protocols

First, we focus on constructing the Vector Knowledge Argument for both vectors. As a first idea, we could just run two instances of the Vector Knowledge Argument VKA (Protocol 12) in parallel, both recursively shrinking $\mathbf{w}'$ and $\mathbf{w}''$. This would double the amount of group elements sent compared to VKA, which can be done better: For their explanation of the Inner Product Argument, Hoffmann, Klooß, and Rupp [8] observed that Pedersen Commitments keep their binding property when batched together – so we just add up the commitments for both vectors, resulting in $[c] = [\mathbf{g}']\,\mathbf{w}' + [\mathbf{g}'']\,\mathbf{w}''$. The protocol can then by adapted by updating $\mathbf{w}'$, $\mathbf{g}'$, $\mathbf{w}''$ and $\mathbf{g}''$ as before, and expanding $[u_\ell]$ to include both vectors $[u_\ell] = \sum_{i-j=\ell} \left(\left[\mathbf{g}_i\right] \mathbf{w}_j + \left[\mathbf{g}_j\right] \mathbf{w}_i \right)$.

4.3.2. Adding the Inner Product

To also track the inner product, we add another commitment using the group element $[Q]$ multiplied with $t = \langle \mathbf{w}', \mathbf{w}'' \rangle$ to $[c]$, which we then update accordingly when shrinking $\mathbf{w}'$ and $\mathbf{w}''$ — so that we are now using $[c] = [\mathbf{g}']\,\mathbf{w}' + [\mathbf{g}'']\,\mathbf{w}'' + t\,[Q]$.

Within a step of the recursion when folding $\hat{\mathbf{w}}' = \mathbf{y}^\mathsf{T}\mathbf{w}'$ and $\hat{\mathbf{w}}'' = \mathbf{x}^\mathsf{T}\mathbf{w}''$, we want the old $t = \langle \mathbf{w}', \mathbf{w}'' \rangle$ preserved in $[\mathbf{g}]\,\mathbf{w} = [u_0]$ as already seen for $[\mathbf{g}]\,\mathbf{w}$ in the Vector Knowledge Protocol.

When just using the same $\mathbf{x}$ to fold both vectors, we get:

$$\langle \hat{\mathbf{w}}', \hat{\mathbf{w}}'' \rangle = \sum_{i=1}^{k} \sum_{j=1}^{k} \left\langle \xi^{i-1}\vec{\mathbf{w}}_i, \xi^{j-1}\vec{\mathbf{w}}_j \right\rangle = \sum_{\ell=2}^{2k} \xi^{\ell-2} \sum_{j+i=\ell} \left\langle \vec{\mathbf{w}}'_i, \vec{\mathbf{w}}''_j \right\rangle$$

This does not contain $\langle \mathbf{w}', \mathbf{w}'' \rangle = \sum_{j-i=0} \left\langle \vec{\mathbf{w}}'_i, \vec{\mathbf{w}}''_j \right\rangle$, so we have to modify the protocol:

4.3.3. Swapping x and y

One solution is to swap the roles $\mathbf{x}$ and $\mathbf{y}$ for $\mathbf{w}''$ and update the vectors using the formulas $\hat{\mathbf{w}}'' = \mathbf{x}^\mathsf{T}\mathbf{w}''$ and $\left[\hat{\mathbf{g}}''\right] = \mathbf{y}^\mathsf{T}\,[\mathbf{g}'']$. This results in

$$\langle \hat{\mathbf{w}}', \hat{\mathbf{w}}'' \rangle = \sum_{i=1}^{k} \sum_{j=1}^{k} \left\langle \xi^{i-1}\vec{\mathbf{w}}_i, \xi^{-j+1}\vec{\mathbf{w}}_j \right\rangle = \sum_{\ell=-k+1}^{k-1} \xi^{-\ell} \sum_{j-i=\ell} \left\langle \vec{\mathbf{w}}'_i, \vec{\mathbf{w}}''_j \right\rangle$$

This equation contains $\langle \mathbf{w}', \mathbf{w}'' \rangle = \sum_{j-i=0} \left\langle \vec{\mathbf{w}}'_i, \vec{\mathbf{w}}''_j \right\rangle$ with coefficient $\xi^0 = 1$ as wanted.

4.3.4. Preprocessing: Fixing t

The last problem we have to solve before assembling the protocol is to ensure that the t used as an coefficient for $[Q]$ inside $[c]$ is indeed the t from the relation. This will be ensured with a preprocessing-step:

Denote by $[Q_{\text{old}}]$ the group element $[Q]$ used in the recursive step before preprocessing, and by $[Q]$ the group element after preprocessing $[Q] = \alpha \cdot [Q_{\text{old}}]$. So, before running the protocol, we have $[c] = \cdots + t\,[Q_{\text{old}}]$. In the preprocessing step, we scale $[Q] := \alpha^{-1}\,[Q_{\text{old}}]$ and update the commitment $[c] := ([c_{\text{old}}] - \alpha t\,[Q]) + t\,[Q]$; which yields a regular "commitment" to t after preprocessing $[c] = \cdots + t\,[Q] = \cdots + t \cdot \alpha^{-1}\,[Q_{\text{old}}]$ if $[c]$ was constructed in the correct way.

4.3.5. Inner Product Argument

Using the ideas collected over the previous subsections, we can now complete the Inner Product Argument as described in [8] or for fixed $k = 2$ in [5].

Protocol 14 (Inner Product Argument IPA)**.**
Preprocessing.

$\mathscr{V} \to \mathscr{P}$: Pick and send $\alpha \leftarrow\!\!\$\, \mathbb{F}_p^n \setminus \{0\}$
$\mathscr{P}, \mathscr{V}$: Both set $[Q] := \alpha^{-1}\,[Q]$, followed by $[c] := ([c] - \alpha t\,[Q]) + t\,[Q]$.

Recursive Step. Suppose $n = k^d$ with $d \geq 1$:

$\mathscr{P}$: Calculate $[u_\ell] = \sum_{i-j=\ell} \left(\left[\mathbf{g}'_i\right] \mathbf{w}'_j + \left[\mathbf{g}''_j\right] \mathbf{w}''_i + \left\langle \mathbf{w}'_i, \mathbf{w}''_j \right\rangle [Q] \right)$.

$\mathscr{P} \to \mathscr{V}$: Send $[u] \in \mathbb{G}^{1\times 2k-1}$. ($[c] = [u_0]$ already known, so $2k-2$ elements sent.)

$\mathscr{V} \to \mathscr{P}$: Sample and transmit $\xi \leftarrow\!\$\, \mathbb{F}_p^n \setminus \{0\}$.

$\mathscr{P}, \mathscr{V}$: Set $\mathbf{x} = \begin{pmatrix} 1 \\ \xi \\ \xi^2 \\ \vdots \end{pmatrix}$, $\mathbf{y} = \begin{pmatrix} 1 \\ \xi^{-1} \\ \xi^{-2} \\ \vdots \end{pmatrix}$ and $\mathbf{z} = \begin{pmatrix} \xi^{k-1} \\ \vdots \\ \xi^{-k+1} \end{pmatrix}$.

Calculate $\left[\hat{\mathbf{g}}'\right] = \mathbf{x}^\mathsf{T} \left[\vec{\mathbf{g}}'\right]$, $\left[\hat{\mathbf{g}}''\right] = \mathbf{y}^\mathsf{T} \left[\vec{\mathbf{g}}''\right]$ and $[\hat{c}] = [\mathbf{u}]\,\mathbf{z}$.

$\mathscr{P}$: Set $\hat{\mathbf{w}} = \mathbf{y}^\mathsf{T}\vec{\mathbf{w}}$, $\hat{\mathbf{w}}'' = \mathbf{x}^\mathsf{T}\vec{\mathbf{w}}''$.

$\mathscr{P}, \mathscr{V}$: Update $\mathbf{w}' = \hat{\mathbf{w}}'$, $\mathbf{w}'' = \hat{\mathbf{w}}''$, $[\mathbf{g}'] := \left[\hat{\mathbf{g}}''\right]$, $[\mathbf{g}''] := \left[\hat{\mathbf{g}}''\right]$ and $[c] := [\hat{c}]$; and repeat the recursive step for $n := n/k$.

Invariant. $[c] = [\mathbf{g}']\,\mathbf{w}' + [\mathbf{g}'']\,\mathbf{w}'' + t\,[Q]$

Base Case.

$\mathscr{P} \to \mathscr{V}$: Send $\mathbf{w}', \mathbf{w}''$.

$\mathscr{V}$: Verify that $[\mathbf{g}']\,\mathbf{w}' + [\mathbf{g}'']\,\mathbf{w}'' + t\,[Q] \stackrel{?}{=} [c]$ and that $\langle \mathbf{w}', \mathbf{w}'' \rangle = t$.

As before, the protocol requires sending $(2k-2)\log_k(n)$ group elements in every step, and the invariant is $[c] = [\mathbf{g}']\,\mathbf{w}' + [\mathbf{g}'']\,\mathbf{w}'' + t\,[Q]$ again. The protocol is correct, because the invariant is maintained in every step of the recursion, as can be seen in the following calculation:

$$
\begin{aligned}
[\hat{c}] = [u_{-k+1}, \ldots, u_{k-1}]\,\mathbf{z}^{(i)} &= \sum_{\ell=-k+1}^{k-1} \left(z_\ell \sum_{i-j=\ell} \left(\left[\vec{\mathbf{g}}'_i\right] \vec{\mathbf{w}}'_j + \left[\vec{\mathbf{g}}''_j\right] \vec{\mathbf{w}}''_i + \left\langle \vec{\mathbf{w}}'_i, \vec{\mathbf{w}}''_j \right\rangle [Q] \right) \right) \\
&= \sum_{\ell=-k+1}^{k-1} \left(\sum_{i-j=\ell} \left[x_i \vec{\mathbf{g}}'_i\right] y_j \vec{\mathbf{w}}'_j + \left[y_j \vec{\mathbf{g}}''_j\right] x_i \vec{\mathbf{w}}''_i + \left\langle x_i \vec{\mathbf{w}}'_i, y_i \vec{\mathbf{w}}''_j \right\rangle [Q] \right) \\
&= \left(\sum_{i=1}^{k} x_i \left[\vec{\mathbf{g}}'_i\right] \right) \left(\sum_{j=1}^{k} y_j \vec{\mathbf{w}}'_j \right) + \left(\sum_{j=1}^{k} y_j \left[\vec{\mathbf{g}}''_j\right] \right) \left(\sum_{i=1}^{k} x_i \vec{\mathbf{w}}''_i \right) + \left\langle \sum_{i=1}^{k} x_i \vec{\mathbf{w}}'_i, \sum_{j=1}^{k} y_j \vec{\mathbf{w}}''_j \right\rangle [Q] \\
&= [\hat{\mathbf{g}}]\,\hat{\mathbf{w}}
\end{aligned}
\tag{4.10}
$$

Lemma 15. *Using the variables and setting as defined in Protocol 14, the following statements hold for one recursive step:*

1. *Given a non-trivial discrete logarithm relation $\hat{\boldsymbol{v}}$ for $\left[\hat{\mathbf{g}}', \hat{\mathbf{g}}'', Q\right]$ folded using $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$, we can efficiently find a non-trivial discrete logarithm relation $\boldsymbol{v}$ for $[\mathbf{g}', \mathbf{g}'', Q]$.*
2. (Unconditional Extraction) *Given $2k-1$ distinct challenges $\xi_{(1)}, \ldots, \xi_{(2k-1)}$ with accepting transcripts, one can extract unconditionally a witness $\boldsymbol{w}'$, $\boldsymbol{w}''$ such that $[\mathbf{g}']\,\boldsymbol{w}' + [\mathbf{g}'']\,\boldsymbol{w}'' + \langle \boldsymbol{w}', \boldsymbol{w}'' \rangle\,[Q] = [c]$.*

Inner Product Argument (Protocol 14)		
	Common Input: $[\mathbf{g}]$	
Prover $\mathscr{P}$ Input: $\mathbf{w}$		Verifier $\mathscr{V}$ Input: $[c]$
	Preprocessing.	
		$\alpha \leftarrow\!\!\$\, \mathbb{F}_p \setminus \{0\}$
	$\xleftarrow{\alpha}$	
$[Q] := \alpha^{-1}\,[Q]$ $[c] := ([c] - \alpha t\,[Q]) + t\,[Q]$		$[Q] := \alpha^{-1}\,[Q]$ $[c] := ([c] - \alpha t\,[Q]) + t\,[Q]$
	Recursive Step. Suppose $n = k^d$ with $d \geq 1$	
$[u_\ell] = \sum_{i-j=\ell} \left(\left[\vec{\mathbf{g}}_i\right] \vec{\mathbf{w}}_j + \left[\vec{\mathbf{g}}_j\right] \vec{\mathbf{w}}_i + \left\langle \vec{\mathbf{w}}'_i, \vec{\mathbf{w}}''_j \right\rangle \right)$ for $\ell \in \{-k+1, \ldots, k-1\}$		
	$\xrightarrow{[\mathbf{u}]}$	
		$\xi \leftarrow \mathbb{F}_p \setminus \{0\}$
		and therefore $\mathbf{x} = \begin{pmatrix} 1 \\ \xi \\ \xi^2 \\ \vdots \end{pmatrix}$, $\mathbf{y} = \begin{pmatrix} 1 \\ \xi^{-1} \\ \xi^{-2} \\ \vdots \end{pmatrix}$ and $\mathbf{z} = \begin{pmatrix} \xi^{k-1} \\ \vdots \\ \xi^{-k+1} \end{pmatrix}$
	$\xleftarrow{\xi}$	
$[\mathbf{g}'] = \mathbf{x}^\mathsf{T} \left[\vec{\mathbf{g}}'\right]$, $[\mathbf{g}''] = \mathbf{x}^\mathsf{T} \left[\vec{\mathbf{g}}''\right]$ $[c] = \mathbf{z}^\mathsf{T}\,[\mathbf{u}]$ $\mathbf{w}' = \mathbf{y}^\mathsf{T} \vec{\mathbf{w}}'$, $\mathbf{w}'' = \mathbf{y}^\mathsf{T} \vec{\mathbf{w}}''$ $n = n/k$		$[\mathbf{g}'] = \mathbf{x}^\mathsf{T} \left[\vec{\mathbf{g}}'\right]$, $[\mathbf{g}''] = \mathbf{x}^\mathsf{T} \left[\vec{\mathbf{g}}''\right]$ $[\hat{c}] = [\mathbf{u}]\,\mathbf{z}$ $n = n/k$
	Start next recursion iteration.	
	Base Case. Suppose $n = 1$	
	$\xrightarrow{\mathbf{w}', \mathbf{w}''}$	
		return $[\mathbf{g}']\,\mathbf{w}' + [\mathbf{g}'']\,\mathbf{w}'' + t\,[Q] \stackrel{?}{=} [c]$

Table 4.2.: Inner Product Argument Protocol IPA [8, Protocol 4.1]

3. (Short-Circuit Extraction) *Given $2k$ distinct challenges $\xi_{(1)}, \ldots, \xi_{(2k)}$ with accepting transcripts if the witness from above does not fit w.r.t. the $[u_\ell]$, i.e. if an honest prover would send different $[u_\ell]$ for $\boldsymbol{w}'$ and $\boldsymbol{w}''$, then we find (additionally) a non-trivial kernel element $\boldsymbol{v}$, i.e. $[\boldsymbol{g}', \boldsymbol{g}'', Q]\ \boldsymbol{v} = 0$.*
4. (Quick Extraction) *From k distinct challenges $\xi_{(1)}, \ldots, \xi_{(k)}$, one can compute a candidate witness $\boldsymbol{w}'$, $\boldsymbol{w}''$ for quick extraction. If $\sum_{i-j=\ell}([\boldsymbol{g}_i]\,\boldsymbol{w}'_j + [\boldsymbol{g}_j]\,\boldsymbol{w}''_i + \langle \boldsymbol{w}'_i, \boldsymbol{w}''_j\rangle\,[Q]) \neq [u_\ell]$ for some ℓ, then we are guaranteed to find a non-trivial kernel relation from $2k$ distinct challenges.*

The following proof works analogously to Lemma 13, but with some steps being more complicated because of the combined commitments.

Proof. For the first statement, we assume that we have a kernel element $\hat{\mathbf{v}} = (\hat{\mathbf{v}}', \hat{\mathbf{v}}'', \hat{a})$, i.e. $\left[\hat{\mathbf{g}}'\right]\hat{\mathbf{v}}' + \left[\hat{\mathbf{g}}''\right]\hat{\mathbf{v}}'' + \hat{a}\,[Q] = 0$. Analogously to the proof of Lemma 13, we can derive the discrete logarithm relation for the original "commitment key" by

$$\begin{aligned}
0 &= \left[\sum_{i=1}^{k} \mathbf{x}_i \mathbf{g}'_i\right]\hat{\mathbf{v}}' + \left[\sum_{i=1}^{k} \mathbf{y}_i \mathbf{g}''_i\right]\hat{\mathbf{v}}'' + a\,[Q] \\
&= \sum_{i=1}^{k}\left[\mathbf{g}'_i\right]\mathbf{x}_i\hat{\mathbf{v}}' + \sum_{i=1}^{k}\left[\mathbf{g}''_i\right]\mathbf{y}_i\hat{\mathbf{v}}'' + a\,[Q] \\
&= [\mathbf{g}']\,(\mathbf{x}\otimes\hat{\mathbf{v}}') + [\mathbf{g}'']\,(\mathbf{y}\otimes\hat{\mathbf{v}}'') + a\,[Q]
\end{aligned}$$

To prove the second statement, we recall the calculation of $[\hat{c}]$ in the recursion step of the protocol:

$$\begin{aligned}
\mathbf{u}\cdot\mathbf{z}^{(i)} &= \left[\hat{\mathbf{g}}'\right]\hat{\mathbf{w}}' + \left[\hat{\mathbf{g}}''\right]\hat{\mathbf{w}}'' + \hat{t}\,[Q] \\
&= \left(\sum_{j=1}^{k}\mathbf{x}_j^{(i)}\left[\vec{\mathbf{g}}'_j\right]\right)\hat{\mathbf{w}}'^{(i)} + \left(\sum_{j=1}^{k}\mathbf{y}_j^{(i)}\left[\vec{\mathbf{g}}''_j\right]\right)\hat{\mathbf{w}}''^{(i)} + \hat{t}\,[Q] \\
&= \left[\vec{\mathbf{g}}'\right]\left(\mathbf{x}^{(i)}\otimes\hat{\mathbf{w}}'^{(i)}\right) + \left[\vec{\mathbf{g}}''\right]\left(\mathbf{y}^{(i)}\otimes\hat{\mathbf{w}}''^{(i)}\right) + \hat{t}\,[Q]
\end{aligned}$$

Defining $\hat{\mathbf{W}}'$ and $\hat{\mathbf{W}}''$ analogously to Lemma 13, and setting $\hat{\mathbf{T}} = (\,t^{(1)}\ \ldots\ t^{(2k-1)}\,)$, we can combine the different sample for $i \in \{1, \ldots, 2k-1\}$ to:

$$\begin{aligned}
\mathbf{u}\cdot\mathbf{Z} &= [\mathbf{g}']\,\hat{\mathbf{W}}' + [\mathbf{g}'']\,\hat{\mathbf{W}}'' + \mathbf{T}\,[Q] \\
&\Leftrightarrow \mathbf{u} = [\mathbf{g}']\,\underbrace{\hat{\mathbf{W}}'\cdot\mathbf{Z}^{-1}}_{\mathbf{W}'} + [\mathbf{g}'']\,\underbrace{\hat{\mathbf{W}}''\cdot\mathbf{Z}^{-1}}_{\mathbf{W}''} + \underbrace{\hat{\mathbf{T}}\cdot\mathbf{Z}^{-1}}_{\mathbf{T}}\,[\mathbf{Q}] \\
&= [\mathbf{g}']\,\mathbf{W}' + [\mathbf{g}'']\,\mathbf{W}'' + \mathbf{T}\,[Q]
\end{aligned}$$

As before, we have unconditionally extracted a witness $\mathbf{w}'^{(0)}$ as the mid column of $\mathbf{W}'$ and $\mathbf{w}''^{(0)}$ as the mid column of $\mathbf{W}''$ and $\mathbf{t}^{(0)}$ from $\mathbf{T}$, satisfying the relation $\mathcal{R}_{\mathsf{IPA}}$, because $[c] = [\mathbf{u}_0] = [\mathbf{g}']\,\mathbf{w}'^{(0)} + [\mathbf{g}'']\,\mathbf{w}''^{(0)} + \mathbf{t}^{(0)}\,[Q]$.

For the third statement, as in Lemma 13, we find two different alternative formulations of $[\mathbf{u}]\,\mathbf{z}$:

$$\begin{aligned}&\left[\vec{\mathbf{g}}'\right]\mathbf{W}'\mathbf{z}^{(i)} + \left[\vec{\mathbf{g}}''\right]\mathbf{W}''\mathbf{z}^{(i)} + \mathbf{T}\mathbf{z}^{(i)}\,[Q]\\ &= \mathbf{u}\mathbf{z}^{(i)} = \left[\vec{\mathbf{g}}'\right]\left(\mathbf{x}^{(i)} \otimes \hat{\mathbf{w}}'^{(i)}\right) + \left[\vec{\mathbf{g}}''\right]\left(\mathbf{y}^{(i)} \otimes \hat{\mathbf{w}}''^{(i)}\right) + \hat{t}^{(i)}\,[Q]\end{aligned}$$

Then we split those equations by the different parts of the vectors of group elements $[\mathbf{g}']$, $[\mathbf{g}'']$ and $[Q]$. If the following equations don't hold, we found a non-trivial discrete logarithm relation.

$$\begin{aligned}\mathbf{W}' \cdot \mathbf{z}^{(i)} &= \left(\mathbf{x}^{(i)} \otimes \hat{\mathbf{w}}'^{(i)}\right)\\ \mathbf{W}'' \cdot \mathbf{z}^{(i)} &= \left(\mathbf{y}^{(i)} \otimes \hat{\mathbf{w}}''^{(i)}\right)\\ \mathbf{T} \cdot \mathbf{z}^{(i)} &= \hat{t}^{(i)}\end{aligned} \tag{4.11}$$

Following exactly the steps for determining $\hat{\mathbf{w}}$ in Lemma 13, we find out for $\hat{\mathbf{w}}'$ and $\hat{\mathbf{w}}''$:

$$\begin{aligned}\sum_{j=1}^{k} \mathbf{y}_j^{(i)} \cdot \mathbf{w}_j'^{(0)} &= \hat{\mathbf{w}}'^{(i)}\\ \sum_{j=1}^{k} \mathbf{x}_j^{(i)} \cdot \mathbf{w}_j''^{(0)} &= \hat{\mathbf{w}}''^{(i)}\end{aligned} \tag{4.12}$$

Note that as in Lemma 13, this is where we need the $2k$ different challenges to compare the coefficients to determine the structure of $\mathbf{W}'$ and $\mathbf{W}''$. If now $t^{(0)} = \left\langle \mathbf{w}'^{(0)}, \mathbf{w}''^{(0)} \right\rangle$, extraction was successful and we have successfully found a matching witness for the relation. For the following formula, we split $T = \left(t^{(-k+1)} \ \dots \ t^{(k-1)} \right) \in \mathbb{F}_p^{1\times 2k-1}$:

$$\begin{aligned}\sum_{\ell=-k+1}^{k-1} \xi^{-\ell} t^{(\ell)} = \sum_{\ell=-k+1}^{k-1} \mathbf{z}_\ell t^{(\ell)} = \mathbf{T}\mathbf{z} \overset{①}{=} \hat{t} \overset{②}{=} \langle \hat{\mathbf{w}}', \hat{\mathbf{w}}'' \rangle \overset{③}{=} \left\langle \mathbf{x}^{\mathsf{T}}\mathbf{w}', \mathbf{y}^{\mathsf{T}}\mathbf{w}'' \right\rangle\\ = \sum_{\ell=-k+1}^{k-1} \mathbf{z}_\ell \sum_{\ell=i-j} \left\langle \mathbf{w}'_i, \mathbf{w}''_j \right\rangle = \sum_{\ell=-k+1}^{k-1} \xi^{-\ell} \sum_{\ell=i-j} \left\langle \mathbf{w}'_i, \mathbf{w}''_j \right\rangle\end{aligned}$$

Equality ① was already proven in Equation (4.11) and for ②, we know that this statement is a part of the definition of $[\hat{c}]$ and therefore has to be true or a non-trivial discrete logarithm relation has been found. ③ holds because of Equation (4.12).

This equation with $2k-1$ powers of ξ is valid for $2k-1$ distinct challenges, so that the coefficients of the $x^{-\ell}$ have to be identical. This implies that $t^{(0)} = \langle \mathbf{w}', \mathbf{w}'' \rangle$ and therefore that our extraction must have been successful. Having determined the structure of $\mathbf{W}'$, $\mathbf{W}''$ and $\mathbf{T}$, we now know that $[\mathbf{u}] = [\mathbf{g}']\,\mathbf{W}' + [\mathbf{g}'']\,\mathbf{W}'' + \mathbf{T}\,[Q]$ and therefore $[\mathbf{u}]$ is fixed.

Quick Extraction and therefore the last statement can be proven analogous to Lemma 13: With k different challenges $\xi^{(i)}$ and therefore k different $\mathbf{x}$ and $\mathbf{y}$, we can derive from the update equations in the protocol $\hat{\mathbf{w}}' = \mathbf{y}^{\mathsf{T}}\vec{\mathbf{w}}'$ and $\hat{\mathbf{w}}'' = \mathbf{x}^{\mathsf{T}}\vec{\mathbf{w}}''$ that:

$$\begin{pmatrix} \hat{\mathbf{w}}'^{(1)} & \hat{\mathbf{w}}'^{(2)} & \cdots & \hat{\mathbf{w}}'^{(k)} \end{pmatrix} = \vec{\mathbf{w}}'^{\mathsf{T}} \underbrace{\begin{pmatrix} \mathbf{y}^{(1)} & \mathbf{y}^{(2)} & \cdots & \mathbf{y}^{(k)} \end{pmatrix}}_{=\mathbf{Y}}$$

$$\begin{pmatrix} \hat{\mathbf{w}}'^{(1)} & \hat{\mathbf{w}}'^{(2)} & \cdots & \hat{\mathbf{w}}'^{(k)} \end{pmatrix} \cdot \mathbf{Y}^{-1} = \vec{\mathbf{w}}'^{\mathsf{T}}$$

$$\begin{pmatrix} \hat{\mathbf{w}}''^{(1)} & \hat{\mathbf{w}}''^{(2)} & \cdots & \hat{\mathbf{w}}''^{(k)} \end{pmatrix} \cdot \mathbf{X}^{-1} = \vec{\mathbf{w}}''^{\mathsf{T}}$$

Using this approach, we have extracted candidate solutions $\mathbf{w}'$ and $\mathbf{w}''$. If the equation from the relation for $[c]$ doesn't hold for the recovered $\mathbf{w}'$ and $\mathbf{w}''$, we have fulfilled the conditions of the third statement and therefore found a non-trivial discrete logarithm relation. □

Using the Lemma just proven for extraction of the recursive step, we can derive the following Corollary about protocol 14.

Corollary 16. *Protocol* IPA *is special* $(2, 2k, \ldots, 2k)$*-sound for finding a valid witness* $(\boldsymbol{w}', \boldsymbol{w}'')$ *or a non-trivial element in the kernel of* $[\boldsymbol{g}', \boldsymbol{g}'', Q]$*. It has* $(1, k, \ldots, k)$ *short-circuit extraction.*

Proof. Since extraction for the recursive step has already been shown in Lemma 15, we only have to consider the preprocessing step here. Assume we are given two transcripts with challenges $\alpha^{(1)}$, $\alpha^{(2)}$ and extracted witnesses $\mathbf{w}'^{(i)}$, $\mathbf{w}''^{(i)}$ and $s^{(i)} := \left\langle \mathbf{w}'^{(i)}, \mathbf{w}''^{(i)} \right\rangle$ for $i \in \{1, 2\}$. We will consider two different cases here:

In the first case, at least one of the following equations does not hold: $\mathbf{w}'^{(1)} = \mathbf{w}'^{(2)}$, $\mathbf{w}''^{(1)} = \mathbf{w}''^{(2)}$ or $s^{(1)} = s^{(2)}$. In this case, we have found two different openings for $\left[c^{(1)}\right] = \left[c^{(2)}\right]$, and therefore a non-trivial discrete logarithm relation.

In the second case, those values are all identical. Over the whole extraction procedure we have found $\mathbf{w}'$ and $\mathbf{w}''$ such that

$$[c] = [\mathbf{g}']\,\mathbf{w}' + [\mathbf{g}'']\,\mathbf{w}'' + \underbrace{\langle \mathbf{w}', \mathbf{w}'' \rangle}_{=s}\,[Q] = [\mathbf{g}']\,\mathbf{w}' + [\mathbf{g}'']\,\mathbf{w}'' + \alpha \cdot s\,[Q_{\text{old}}]$$

According to the statement and the description of the protocol, $[c]$ has to have the form

$$\begin{aligned} [c] &= [\mathbf{g}']\,\mathbf{w}' + [\mathbf{g}'']\,\mathbf{w}'' + t\,[Q_{\text{old}}] - t\,[Q_{\text{old}}] + \alpha \cdot t\,[Q_{\text{old}}] \\ &= [c] = [\mathbf{g}']\,\mathbf{w}' + [\mathbf{g}'']\,\mathbf{w}'' + \alpha \cdot t\,[Q_{\text{old}}] \end{aligned}$$

Comparing the coefficients of $[Q_{\text{old}}]$, we find out that $\alpha \cdot t = \alpha \cdot s$ and, since $\alpha \neq 0$, this implies $s - t = 0$ and therefore $t = \langle \mathbf{w}', \mathbf{w}'' \rangle$, giving us unconditional extraction with 2 transcripts. For quick extraction just verify if the witness fits for the unmodified relation with $[c_{\text{old}}]$ using $[Q_{\text{old}}]$, which only requires 1 transcript. □

Protocol 1 from **[5]**	Protocol 14
$\mathbf{a}$	$\mathbf{w}'$
$\mathbf{b}$	$\mathbf{w}''$
$\mathbf{a}'$	$\hat{\mathbf{w}}'$
c	t
$\mathbf{a}_\mathrm{L}$	$\mathbf{w}'^{(-1)}$
$\mathbf{a}_\mathrm{P}$	$\mathbf{w}'^{(0)}$
$\mathbf{a}_\mathrm{R}$	$\mathbf{w}'^{(1)}$
analogous for $\boldsymbol{b} \to \boldsymbol{w}''$, $c \to t$	

Protocol 1 from **[5]**	Protocol 14
$\mathbf{g}$	$[\mathbf{g}']$
$\mathbf{h}$	$[\mathbf{g}'']$
$\mathbf{g}'$	$[\hat{\mathbf{g}}']$
$\mathbf{u}$	$[Q]$
L	$[\mathbf{u}_{-1}]$
P	$[\mathbf{u}_0] = [c]$
R	$[\mathbf{u}_1]$
t	$[c]$
x_i	$\xi^{(i)}$

Table 4.3.: Translation between the variable names here and in Bünz et al. [5]

4.3.6. Comparison to other versions of the Inner Product Arguments

The protocols presented here are based on the modifications by Hoffmann, Klooß, and Rupp [8] to the original Bulletproof Inner Product Argument [5]. Instead of generalizing to testing distributions, we fixed $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$. The Protocol VKA corresponds to $\mathsf{LMPA}_{\mathrm{noZK}}$ (Protocol 3.9) from the paper, and was simplified by only allowing matrices with $m = 1$ columns and renaming $\mathbf{A}$ to $\mathbf{g}$ and t to c to match the notation in the Inner Product Argument. IPA corresponds to $\mathsf{IPA}_{\mathrm{noZK}}$ (Protocol 4.1) and is similar, but consistently uses a variable k instead of fixing $k = 2$ to minimize the amount of group elements sent.

Compared to the Bulletproof Paper by Bünz et al. [5], more modifications have been made: The notation of the group $\mathbb{G}$ was changed from the multiplicative style to the additive style, allowing for vector notations and therefore simplifying the transformations. To denote group elements, additional parenthesis are used, for example to denote $[Q]$. As in its predecessor by Bootle et al. [4], this paper generalizes to $k \in \mathbb{N}$. Almost all variables have been renamed to match [8], the translation between the different protocols has been summarized in Table 4.3. Additionally, the coefficients used for folding $\mathbf{w}'$ and $\mathbf{w}''$ have been simplified. In case of $k = 2$, they changed from $\mathbf{x} = (\,\xi^{-1}\ \xi\,)^\mathsf{T}$ to $\mathbf{x} = (\,1\ \xi\,)^\mathsf{T}$, because this way multiplications with 1 are free. In the notation used in this paper or by [8], the coefficients by Bünz et al. [5] are:

$$\mathbf{x} = \begin{pmatrix} \xi^{-1} \\ \xi \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} \xi \\ \xi^{-1} \end{pmatrix} \qquad \mathbf{z} = \begin{pmatrix} \xi^2 \\ 1 \\ \xi^{-2} \end{pmatrix}$$

The preprocessing in [5, Protocol 2] has been merged into Protocol 14; instead of fixing the inner product by adding it to the statement, as in [8] the statement is modified to ensure that the correct t was used.

The predecessor to the Bulletproof Inner Product Argument [4, Section 4] doesn't merge the commitments and therefore runs two distinct instances of the vector knowledge protocol with

challenges

$$\mathbf{x} = \begin{pmatrix} \xi^{-1} \\ \xi^{-2} \\ \vdots \\ \xi^{-k} \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} \xi \\ \xi^{2} \\ \vdots \\ \xi^{k} \end{pmatrix} \qquad \mathbf{z} = \begin{pmatrix} \xi^{k-1} \\ \vdots \\ \xi^{-k+1} \end{pmatrix}$$

The commitment $[c]$ to the vectors is split into A and B; and the value of the inner product t is called z. Otherwise, the notation resembles the one used in [5]. Instead of fixing $k = 2$, the version of the protocol by Bootle et al. [4] is defined for general k.

5. Dynamic Extraction Framework

In the following chapter, the framework of predicate-extended emulation introduced by Jaeger and Tessaro [9] is adapted to the dynamic setting required for extractors with quick-extraction as seen for Protocol 14. Then we rephrase the Forking Lemma for this new setting, which allows us to combine a dynamic extractor with a dynamic tree finder to construct an emulator for knowledge soundness. The results from this chapter can then be used together with statements as suggested in Conjecture 10 in the next Chapter 6 to prove the linear amount of required transcripts for the Bulletproof Inner Product Argument.

5.1. Extraction Framework

For a proof system PS, the Forking Lemma [4] proves that given an algorithm to extract a witness from a tree of transcripts of a certain size as introduced in Section 2.2.2, we have knowledge soundness. So the extraction process is split into two steps by the formulation of the lemma: First, a tree is constructed by rewinding the prover P^*; then this tree of transcripts is given to an extractor to calculate the witness. Burst-extraction as introduced in Chapter 3 can be used to model the sampling behaviour for the quick-extraction of the Inner Product Argument (Protocol 14). We have seen that many transcripts of the static tree have not been looked at in the extraction process and therefore were not necessary. This is why in this section a dynamic version of the extraction framework will be presented, where only the transcripts that are required for extraction are generated on demand.

For the following section, recall the syntax and security of proof systems introduced in the Preliminaries Section 2.2.

5.1.1. Predicate-Extended Emulation

For a proof system PS to be knowledge sound with knowledge error $\kappa(|u|)$, according to Definition 4 we need an emulator E, with rewind access to a possibly dishonest prover P^*. Whenever a regular execution of the protocol with the prover P^* for a statement u is accepting with probability $\epsilon(u)$, the emulator E extracts a witness with a probability of at least ${}^{(\epsilon(u)-\kappa(|u|))}/_{q(|u|)}$, where q is a polynomial.

Here, we will "split" emulation soundness into two game-based security properties, from which together knowledge soundness follows directly: *emulation security* demands that the transcripts generated by E are indistinguishable from those generated by a regular protocol execution with P^*. And *predicate extension*, which is parametrized by some predicate Π evaluated on the statement, the public parameters and the output of the extractor, is fulfilled if whenever E produces an accepting transcript, the auxiliary output w evaluates Π to be true.

Game $\mathsf{H}^{\mathsf{emu}}_{\mathsf{PS},\mathsf{E},b}(\mathsf{P}^*, \mathcal{A})$

1 : $global\ i \leftarrow 0$
2 : $(\mathsf{S}, \cdot, \cdot, \mathsf{V}, \mu) \leftarrow \mathsf{PS}$
3 : $pp \leftarrow\!\$\ \mathsf{S}$
4 : $(u, s, \sigma_{\mathcal{A}}) \leftarrow\!\$\ \mathcal{A}\,(pp)$
5 : $(\mathrm{tr}_1, \cdot) \leftarrow\!\$\ \left\langle \mathsf{P}^*_{pp}(u, s), \mathsf{V}_{pp}(u) \right\rangle$
6 : $(\mathrm{tr}_0, \cdot) \leftarrow\!\$\ \mathsf{E}^{\mathsf{Next},\mathsf{Rewind}}(pp, u)$
7 : $b' \leftarrow\!\$\ \mathcal{A}(\mathrm{tr}_b, \sigma_{\mathcal{A}})$
8 : return $b' \stackrel{?}{=} 1$

Game $\mathsf{H}^{\mathsf{predext}}_{\mathsf{PS},\mathsf{E},\Pi}(\mathsf{P}^*, \mathcal{A})$

1 : $global\ i \leftarrow 0$
2 : $(\mathsf{S}, \cdot, \cdot\mathsf{V}, \mu) \leftarrow \mathsf{PS}$
3 : $pp \leftarrow\!\$\ \mathsf{S}$
4 : $(u, s, \cdot) \leftarrow\!\$\ \mathcal{A}(pp)$
5 : $(\mathrm{tr}, w) \leftarrow\!\$\ \mathsf{E}^{\mathsf{Next},\mathsf{Rewind}}(pp, u)$
6 : return $(\mathsf{V}_{pp}(u, \mathrm{tr}) \wedge \neg\Pi(pp, u, w))$

Game $\mathsf{H}^{\mathsf{pred}}_{\mathsf{S},\Pi}(\mathcal{A})$

1 : $pp \leftarrow\!\$\ \mathsf{S}$
2 : $w \leftarrow\!\$\ \mathcal{A}(pp)$
3 : return $\Pi\,(pp, \epsilon, w)$

Figure 5.1.: Games defining the Security Properties of Proof Systems

Note that i is a global variable which denotes the round of the protocol and therefore the corresponding layer of the tree the extraction process is currently in. It is used in the oracles $\mathsf{Next}()$ and $\mathsf{Rewind}()$ defined in Figure 5.2. The empty value filling unused parameters is denoted by ϵ.

In the following paragraphs, we will have a closer look at those two properties:

Emulation Security The game $\mathsf{H}^{\mathsf{emu}}_{\mathsf{PS},\mathsf{E},b}(\mathsf{P}^*, \mathcal{A})$, which can be found as pseudo-code in Figure 5.1, is parametrized by a public coin proof system PS, an emulator E and a bit b. The adversaries considered here are a possibly cheating prover P^* and a distinguisher $\mathcal{A}$. This distinguisher $\mathcal{A}$ chooses the statement to be proven and tries to find out whether the transcript that is given was generated by a regular protocol execution or the emulator. In the game, public parameters are sampled from the setup algorithm S, which are then handed to $\mathcal{A}$ to choose the statement u along with the secret information s for the prover P specified by the protocol. Then the emulator E and the regular protocol $\langle \mathsf{P}^*_{pp}\,(u, s)\,, \mathsf{V}_{pp}\,(u)\rangle$ are run to obtain two transcripts tr_0 and tr_1. The transcript tr_b determined by the bit b from the specification of the game is then given to $\mathcal{A}$, which then attempts to guess the bit b correctly.

We define the advantage function for this game as $\mathsf{Adv}^{\mathsf{emu}}_{\mathsf{PS},\mathsf{E}}\,(\mathsf{P}^*, \mathcal{A}) = \mathbf{Pr}[\mathsf{H}^{\mathsf{emu}}_{\mathsf{PS},\mathsf{E},1}\,(\mathsf{P}^*, \mathcal{A})] - \mathbf{Pr}[\mathsf{H}^{\mathsf{emu}}_{\mathsf{PS},\mathsf{E},0}\,(\mathsf{P}^*, \mathcal{A})]$. Here in this thesis, the advantage will always be $\mathsf{Adv}^{\mathsf{emu}}_{\mathsf{PS},\mathsf{E}}\,(\mathsf{P}^*, \mathcal{A}) = 0$. Specifying an emulator with this property alone is not difficult, because we can just run the protocol with P^* by calling $\textsc{Next}\ \mu + 1$ times. In combination with the next security definition of Predicate Extension, this notion becomes useful.

Predicate Extension $\mathsf{H}^{\mathsf{predext}}_{\mathsf{PS},\mathsf{E},\Pi}\,(\mathsf{P}^*, \mathcal{A})$, which as $\mathsf{H}^{\mathsf{emu}}$ can be found as pseudo-code in Figure 5.1, is won by the adversary $\mathcal{A}$ and the possibly dishonest prover P^* if the transcript output by the emulator for a statement generated by the attacker $\mathcal{A}$ is accepting, but the predicate is not fulfilled on the witness extracted.

The advantage function is $\mathsf{Adv}^{\mathsf{predext}}_{\mathsf{PS,E},\Pi}(\mathsf{P}^*, \mathcal{A}) = \mathbf{Pr}[\mathsf{H}^{\mathsf{predext}}_{\mathsf{PS,E},\Pi}(\mathsf{P}^*, \mathcal{A}) = 1]$. Again, it is easy to construct an emulator with $\mathsf{Adv}^{\mathsf{predext}}_{\mathsf{PS,E},\Pi}(\mathsf{P}^*, \mathcal{A}) = 0$: If every transcript output is rejecting, the adversary will never win. This would prohibit emulation soundness.

Witness-Extended Emulation The special case of witness-extended emulation can be described by using the predicate Π_{wit}, which checks if w is a witness for u, i.e. $\Pi_{\mathsf{wit}}(pp, u, w) = ((u, w) \in \mathcal{R}_{pp})$. A proof system with negligible advantages for all provers P^* and all attackers $\mathcal{A}$ in both Predicate Extension for Π_{wit} and Emulation Security also fulfills Witness Extended Emulation as described in [5, Definition 10].

Hard Predicates When working with arguments of knowledge like the Bulletproof Inner Product Argument, we need to formalize solutions for the underlying hard problem about the public parameter pp: A witness-independent predicate is one where $\Pi(pp, u, w)$ does not depend on the second input u.

For example for the Bulletproof Inner Product Argument from Protocol 14, this could be a predicate verifying whether the witness $w = \mathbf{v}$ is a non-trivial discrete logarithm relation for $[\,\mathbf{g}'\ \mathbf{g}''\ Q\,]$. We can formalize this using a setup algorithm S denoted by $\mathsf{S}^n_{\mathbb{G}}$. The setup algorithm generates a vector $[\mathbf{g}]$ of group elements of length m, and with the witness-independent predicate $\Pi^{\mathbb{G},m}_{\mathsf{dl}}(pp = [\mathbf{g}], u, w)$. The predicate checks if $w = \mathbf{v}$ specifies a non-trivial discrete logarithm relation, i.e. $[\mathbf{g}]\,\mathbf{v} = 0$ and $\mathbf{v} \neq 0$.

Hardness of a Predicate To quantify the hardness of a predicate, we introduce one last game $\mathsf{H}^{\mathsf{pred}}$, where an attacker $\mathcal{A}$ is given public parameters pp generated by S, and then checks on the results whether the predicate is fulfilled. The advantage of $\mathcal{A}$ is defined as $\mathsf{Adv}^{\mathsf{pred}}_{S,\Pi}(\mathcal{A}) = \mathbf{Pr}\left[\mathsf{H}^{\mathsf{pred}}_{\mathsf{S},\Pi}(\mathcal{A})\right]$.

For our predicate $\Pi^{\mathbb{G},m}_{\mathsf{dl}}$, the advantage against the discrete logarithm relation assumption is captured by $\mathsf{Adv}^{\mathsf{pred}}_{\mathsf{S}^m_{\mathbb{G}},\Pi^{\mathbb{G},m}_{\mathsf{dl}}}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{dl\text{-}rel}}_{\mathbb{G},n}(\mathcal{A})$.

We can use logical operators like $\wedge$ or $\neg$ to modify and combine predicates. For example $\Pi_1 \vee \Pi_2$ evaluates to true if either Π_1 or Π_2 evaluate to true for a given input.

Tree of Transcripts Trees of Transcripts have already been introduced in Definition 7 or in a variant for 1-entries in the matrix H from the Burst Collision Game introduced in Section 3.1, so here we only describe the encoding used in this section. A tree node is described as a tuple (m_{2i-1}, m_{2i}, ℓ), with ℓ being a list of the children of this node. For leaves, we have $\ell = \emptyset$.

In the static setting used in [9], we first construct a full tree for certain branching factors $\mathbf{k} = (k_1, \ldots, k_\mu)$. Then the extractor is started with this tree as an input. In the dynamic setting used here those two steps are merged: At the beginning, only one accepting transcript is generated, and then the extractor X requests new accepting transcripts from the tree finder T on demand.

The following sections will define the behaviour of the tree finder and the interface of an extractor.

5.1.2. Oracles

To describe a regular run of the protocol, we define $\langle \mathsf{P}_{pp}(u, w), \mathsf{V}_{pp}(u) \rangle$. In the corresponding pseudo-code displayed at the left side of Figure 5.2, first the states of the prover σ_{P} and the

$\langle \mathsf{P}_{pp}(u, w), \mathsf{V}_{pp}(u) \rangle$

1 : $\sigma_{\mathsf{P}} \leftarrow \perp; \sigma_{\mathsf{V}} \leftarrow \perp; m_{-1} \leftarrow \perp$
2 : $(m_0, \sigma_{\mathsf{P}}) \leftarrow\$ \mathsf{P}_{pp}(u, w, m_{-1}, \sigma_{\mathsf{P}})$
3 : for $i = 1, \ldots, \mu$ do
4 : $\quad (m_{2i-1}, \sigma_{\mathsf{V}}) \leftarrow\$ \mathsf{V}_{pp}(u, m_{2i-2}, \sigma_{\mathsf{V}})$
5 : $\quad (m_{2i}, \sigma_{\mathsf{P}}) \leftarrow\$ \mathsf{P}_{pp}(u, w, m_{2i-2}, \sigma_{\mathsf{P}})$
6 : $tr \leftarrow (m_{-1}, m_0, m_1, \ldots, m_{2\mu})$
7 : $d \leftarrow \mathsf{V}_{pp}(m_{2\mu}, u, \sigma_{\mathsf{V}})$
8 : return (tr, d)

Next()

1 : **require** $i \leq \mu$
2 : if $i = 0$ then
3 : $\quad \sigma_{\mathsf{P}}^0 \leftarrow \perp; \sigma_{\mathsf{V}}^1 \leftarrow \perp; m_{-1} \leftarrow \perp$
4 : $\quad (u, m_0, \sigma_{\mathsf{P}}^1) \leftarrow\$ \mathsf{P}^*_{pp}(u, s, m_{-1}, \sigma_{\mathsf{P}}^0)$
5 : else
6 : $\quad (m_{2i-1}, \sigma_{\mathsf{V}}^{i+1}) \leftarrow\$ \mathsf{V}_{pp}(u, m_{2i-2}, \sigma_{\mathsf{V}}^i)$
7 : $\quad (m_{2i}, \sigma_{\mathsf{P}}^{i+1}) \leftarrow\$ \mathsf{P}^*_{pp}(u, s, m_{2i-1}, \sigma_{\mathsf{P}}^i)$
8 : $i \leftarrow i + 1$
9 : return (m_{2i-1}, m_{2i})

Rewind()

1 : **require** $i > 0$
2 : $i \leftarrow i - 1$
3 : return ϵ

Figure 5.2.: Oracles giving access to the protocol for the tree finder

verifier V are initialized empty. Also, the message m_{-1} is set to $\perp$ as the first message of the transcript — this message is only a placeholder and is not sent by the verifier V, but allows for easier indexing in line 1. Afterwards, both the prover P and the verifier V are called in an alternating way, their state is updated and the sent messages m_j are saved. Those messages are then assembled into a transcript tr in line 6, and are returned along with the bit d indicating whether the transcript was accepted by the verifier.

The complete run of a protocol as defined in $\langle \mathsf{P}_{pp}(u, w), \mathsf{V}_{pp}(u) \rangle$ can also be split into $\mu + 1$ calls to Next (): In the first call to Next () when $i = 0$, as for the complete run, the different state variables are initialized. To allow for rewinding, the states of the prover and the verifier are indexed with σ_{P}^i, where $i - 1$ is the round where this version of the state was output — so that in can be used as the input for round i. Otherwise, if $0 < i \leq \mu$, first the verifier is called to generate a new challenge, followed by a call to the prover. The messages and the states after this round are then saved after every step, the round counter is increased in line 8 and the new messages m_{2i-1} by the verifier V and m_{2i} by the prover P are returned in line 9.

To allow for fetching multiple challenges at a certain step of the transcript, we introduce the Rewind ()-function, which decreases the protocol round counter $i - 1$, so that the next call of Next () works with the σ_{P} and σ_{V} from the earlier round $i - 1$.

5.1.3. Emulator and Tree Finding Process

The next component of the framework considered is the Tree Finder T: T has rewind access to the prover P^* by using Next() and Rewind(), and dynamically attempts to find new accepting transcripts on requests from the extractor X.

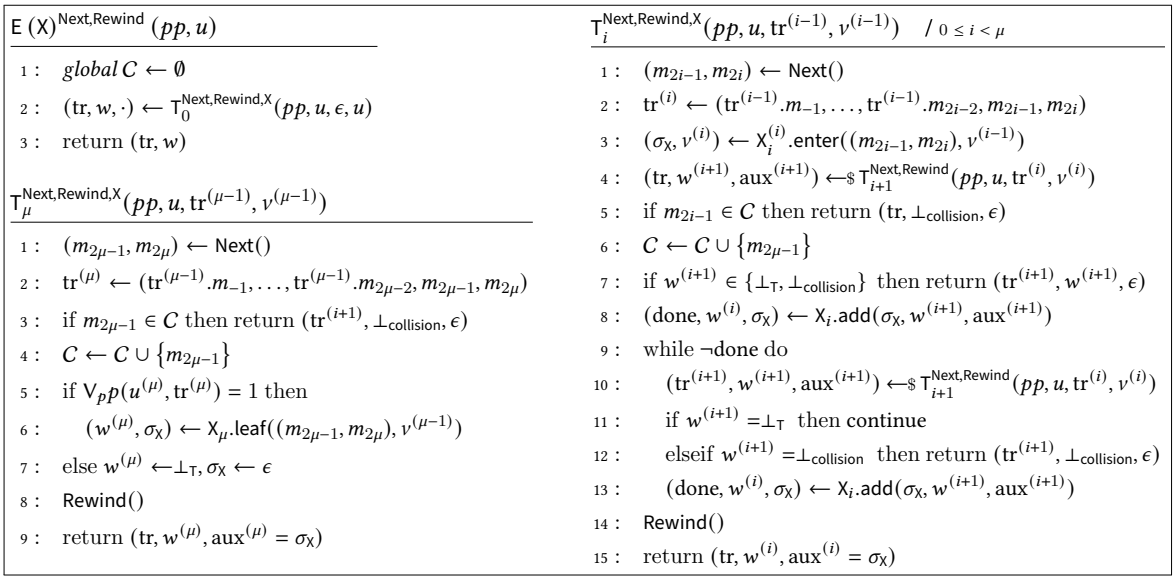


Figure 5.3.: The Tree-Finder T

Instead of explicitly building up a tree of transcripts of a certain size, here the tree is built up dynamically as follows: First, one initial transcript is generated, which corresponds to just a path with $\mu + 1$ nodes. If this transcript is rejected by the verifier V, the extraction process is stopped and the resulting transcript tr is returned — if it was accepted, extraction begins. The implicit tree of transcripts is traversed in the depth first order, i.e. we can always either generate a transcript branching off at the current message and enter extraction for the resulting new child node, or rewind the prover and ascend one level towards the root. If the tree finder and therefore the extractor enter a node outside a generated subtree, this subtree will not be entered again. The pseudo-code for the tree-finder can be found in Figure 5.3.

The starting point for the Tree Finder is the emulator $\mathsf{E}(\mathsf{X})^{\mathsf{Next},\mathsf{Rewind}}$, which has oracle access to the extractor X and to the rewindable protocol instance using $\mathsf{Next}()$ and $\mathsf{Rewind}()$. The call arguments of the emulator are the public parameters pp generated by the setup procedure S, as well as the statement u the proof has to be emulated for. In line 1, the global variable C is initialized as an empty set, which will contain the challenges used by the verifier and therefore allows the tree finder to notice collisions in one run. Afterwards, the tree finder is started on the root layer $i = 0$, and its returned witness and transcript are passed through to the caller.

The tree finder $\mathsf{T}_i^{\mathsf{Next},\mathsf{Rewind},\mathsf{X}}(pp, u, \mathrm{tr}^{(i-1)}, v^{(i-1)})$ itself has oracle access to the rewindable protocol using Next and Rewind and to the extractor X as well. T_i is parametrized by the layer $i \in \{0, \ldots, \mu\}$ of the tree corresponding to the round of the protocol $\mathsf{Next}()$ is currently in. As call arguments, in addition to the public parameters pp and the statement u, it receives the transcript generated up to this point tr along with the updated statement v. The updated statement v has been introduced, because in every recursive step of protocols like the Inner Product Argument (Protocol 14), the statement is shrunk and therefore altered. v is updated in a way, so that the predicate $\Pi(pp, v^{(i)}, w^{(i)})$ holds with the extracted witness for every node of the tree.

In the pseudo-code of the tree finder, two different error symbols can be found: $\perp_{\mathsf{collision}}$ is returned if a collision between two challenges was noticed. When the event occurs, this value

is always propagated upwards, the exploration of new transcripts is stopped, and the error value is returned by the emulator along with the first transcript generated. $\perp_{\mathsf{T}}$ is returned if the first attempt to generate a new transcript for this node has failed. If this occurred for the first transcript generated, no further expansion attempts are started and the error is propagated upwards. If the first extraction was successful and extraction for an additional child returns the error symbol $\perp_{\mathsf{T}}$, the result from the failed extraction is ignored and other child extraction is started.

We define the predicate $\Pi_{\mathsf{collision}}$ to evaluate to true if the given extraction result is the collision error symbol $\perp_{\mathsf{collision}}$, i.e. $\Pi_{\mathsf{collision}}(pp, u, w) = 1 \Leftrightarrow w = \perp_{\mathsf{collision}}$.

Where to add new transcripts is determined by the extractor, which will be considered more closely in the following subsection.

5.1.4. Interface of the Extractor

The extractor has three different methods: When the last message of a transcript, i.e. a leaf on layer $i = \mu$ of the implicit tree of transcripts, is entered, the method $\mathsf{X}_\mu.\mathsf{leaf}((m_{2\mu-1}, m_{2\mu}), v^{(\mu-1)})$ is called. This method receives the last two messages of the transcript $m_{2\mu-1}$ and $m_{2\mu}$ along with the updated statement $v^{(\mu-1)}$, and returns the witness for layer μ denoted by $w^{(\mu)}$ as well as some internal extractor state σ_{X}.

For inner nodes of the tree, there are two functions: $\mathsf{X}_i.\mathsf{enter}$, which is called when a node is entered by the tree finder T. As for entering leaves, parameters are the last two messages m_{2i-1} and m_{2i} as well as the updated state for this node $v^{(i-1)}$. The values returned here are the internal extractor state σ_{X} and a new updated statement $v_{\mathsf{X}}^{(i)}$. The last function $\mathsf{X}_i.\mathsf{add}$ consumes the result of the extraction of a child tree of a current node, and decides whether more children are needed: As parameters, the internal state σ_{X} initially generated in $\mathsf{X}_i.\mathsf{enter}$ on this layer is used, as well as the witness $w^{(i+1)}$ and the internal state $\mathrm{aux}^{(i+1)}$ from the successful extraction on the layer below. As a result, a bit done $\in \{0, 1\}$ requesting another child with "done = 0" and signaling that extraction has been finished with "done = 1" is returned, along with a witness $w^{(i)}$ to be propagated upwards if "done = 1" and the internal state σ_{X}.

5.1.5. Functionality of the Tree Finder

The tree finder itself proceeds as follows: When T_μ called for a leaf, i.e. a full transcript, as formalized in 5.3, the tree finder completes the run of the protocol by calling $\mathsf{Next}()$ in line 1 and compiling the full transcript in line 2. Then it checks whether the challenge was already used in this extraction, and if so, returns a special error symbol $\perp_{\mathsf{collision}}$ as the witness. This indicates that a collision has been found and will stop the entire extraction in line 3. In line 4, the set of used challenges C is updated by adding the new challenge m_{2i-1}. If the verifier accepts the current transcript, which is checked with $\mathsf{V}_{pp}(u^{(\mu)}, \mathrm{tr}^{(\mu)}) = 1$, then the extractor is called for this leaf and the corresponding witness $w^{(\mu)}$ and internal state σ_{X} are saved to be returned later. Otherwise, as the witness $w^{(i)}$ another special error symbol $\perp_{\mathsf{T}}$ is returned, which indicates that no extraction was started by the tree finder T because the corresponding transcript has not been accepted. Afterwards, the protocol instance is rewound using $\mathsf{Rewind}()$; and the transcript $\mathrm{tr}^{(\mu)}$, the witness $w^{(\mu)}$ and the internal state $\mathrm{aux}^{(\mu)} = \sigma_{\mathsf{X}}$ are returned.

When called for an inner node, i.e. with $i \in \{0, \dots, \mu - 1\}$, the procedure T_i in Figure 5.3 is used. As for the leaves, the protocol is run for one round using $\mathsf{Next}()$ and the new messages are appended to the transcript tr. Before checking for collisions and updating C in lines 5 – 6, first the extractor is called for the new messages, and the initial extractor state σ_{X} and the updated statement $v^{(i)}$ are generated. Then in line 4, the tree finder is called to complete the current transcript and additionally extract the witness for the corresponding subtree. The outcome of the extraction is checked in line 7, and if either a collision was found or the first transcript generated in this subtree was rejected, the error is propagated upwards. Otherwise, the result of the first extraction is handed to the extractor using $\mathsf{X}_i.\mathsf{add}()$. Now, the additionally requested subtrees are generated: Until the extractor returns done = 1, the tree finder $\mathsf{T}_{i+1}^{\mathsf{Next},\mathsf{Rewind}}$ is called for the next level in line 10. Next, possible error values are taken care of: If the first transcript generated was not accepting indicated by $w^{(i+1)} = \perp_{\mathsf{T}}$, another attempt is started by jumping back to the start of the loop using continue in line 11. If a collision signalled by $w^{(i+1)} = \perp_{\mathsf{collision}}$ was found, the collision is propagated upwards. If no error occurred, the procedure of the extractor $\mathsf{X}_i.\mathsf{add}()$ is called with the extracted witness $w^{(i+1)}$ and the internal state $\mathrm{aux}^{(i+1)}$, and returns an updated state σ_{X} along with "done" indicating whether additional subtrees are needed (done = 0). When the extractor X has indicated that additional children are required for this node with "done = 0", the protocol instance is rewound using $\mathsf{Rewind}()$ in line 14. The complete transcript $\mathrm{tr}^{(i)}$ along with the extracted witness $w^{(i)}$ and the state of the extractor as an auxiliary value aux are returned.

5.1.6. Tree-Extractor

As an example on how an extractor may look like, in Figure 5.4 the tree builder $\mathsf{X}^{\mathbf{k}-\mathsf{tree}}$ is presented. Recall that a node of the tree of transcripts is denoted by (m_{2i-1}, m_{2i}, ℓ), where ℓ is the list of the child nodes. The tree building extractor is parametrized with the branching factors $\mathbf{k} = (k_1, \dots, k_\mu) \in \mathbb{N}^\mu$. When asked to visit a leaf, the tree builder returns a tree just consisting of the root labelled with messages $m_{2\mu-1}$ and $m_{2\mu}$, i.e. $w^{(\mu)} = (m_{2\mu-1}, m_{2\mu}, \ell = \emptyset)$. When entering an inner node on layer i with $\mathsf{X}_i^{\mathbf{k}-\mathsf{tree}}.\mathsf{enter}()$, as the state the two new message (m_{2i-1}, m_{2i}, ℓ) along with an empty list of children is saved. To this list ℓ, $\mathsf{X}.\mathsf{add}()$ then adds the extracted child trees until k_i children have been found.

$\mathsf{X}_\mu^{\mathbf{k}-\mathsf{tree}}.\mathsf{leaf}((m_{2\mu-1}, m_{2\mu}), v^{(\mu-1)})$

1 : $\ell \leftarrow ()$
2 : $w^{(\mu)} \leftarrow (m_{2\mu-1}, m_{2\mu}, \ell)$
3 : return $(w^{(\mu)}, \sigma_{\mathsf{X}} = \epsilon)$

$\mathsf{X}_i^{\mathbf{k}-\mathsf{tree}}.\mathsf{enter}((m_{2i-1}, m_{2i}), v^{(i-1)})$

1 : $\sigma_{\mathsf{X}} \leftarrow (m_{2i-1}, m_{2i}, \ell = ())$
2 : return $(\sigma_{\mathsf{X}}, v^{(i)} = \epsilon)$

$\mathsf{X}_i^{\mathbf{k}-\mathsf{tree}}.\mathsf{add}(\sigma_{\mathsf{X}}, w^{(i+1)}, \mathrm{aux}^{(i+1)})$

1 : parse $(m_{2i-1}, m_{2i}, \ell) \leftarrow \sigma_{\mathsf{X}}$
2 : $\ell.\mathsf{add}(w^{(i+1)})$
3 : $\sigma_{\mathsf{X}} \leftarrow (m_{2i-1}, m_{2i}, \ell)$
4 : if $|\ell| = k_i$ then
5 : return $(\mathrm{done} = 1, w^{(i)} = \sigma_{\mathsf{X}}, \sigma_{\mathsf{X}})$
6 : return $(\mathrm{done} = 0, w^{(i)} = \epsilon, \sigma_{\mathsf{X}})$

Figure 5.4.: The Tree-Builder $\mathsf{X}^{\mathbf{k}-\mathsf{tree}}$

5.2. Point-Wise Proximity

For our dynamic version of the Forking Lemma, we need a way to bound the probability of a collision occurring. This is the only way our extraction process would fail, because an accepting transcript is returned without an usable result. One solution is to bound the probability by applying the PRP-PRF-switching lemma [11, Lemma 4.5.12], which gives a bound of $q^2/2N$ with q being maximal amount of oracle queries, and N the amount of possible values. This version of the switching lemma can not be applied here, because we only know the expected amount instead of a bound on the amount of oracle queries — therefore we need another approach.

In the following section, the indistinguishability framework used by Jaeger and Tessaro [9] is introduced. Additionally, a theorem giving a bound for the advantage of an arbitrary attacker with a known expected amount of oracle accesses is stated and proven. Then, this theorem will be used to state an expected-time version of the PRF-PRP-switching lemma by adapting the proof by Chang and Nandi [6].

5.2.1. Indistinguishability Framework

As Jaeger and Tessaro [9], we will follow the random systems abstraction by Maurer [10]. In this abstraction, there is a distinguisher D which has to find out whether it is interacting with the game F or G. To distinguish, D may send an element from the set $\mathcal{X}$ to the game, and receives an element of $\mathcal{Y}$ as an answer. This may continue over multiple rounds, until D stops the conversation.

Instead of providing algorithms for defining a game F and the distinguisher D, their behaviour is defined by families of probability functions like $\{p_i^F\}_{i\in\mathbb{N}_{>0}}$ returning the probability of a response given a certain prior "conversation". For the game F, we have the probability function $p_i^F : \mathcal{X}^i \times \mathcal{Y}^{i-1} \times \mathcal{Y} \to [0,1]$ for each i, which states a probability for a $y \in \mathcal{Y}$ being returned after a communication of $x \in \mathcal{X}^i$ by D and $y^{i-1} \in \mathcal{Y}^{i-1}$ by F, so the probability for every possible $y \in \mathcal{Y}$ returned is $\sum_{y\in\mathcal{Y}} p_i^F\left(x^i, y_{i-1}, y\right) = 1$. For the distinguisher D, analogously there is a family of functions $p_i^D : \mathcal{X}^{i-1} \times \mathcal{Y}^{i-1} \times \mathcal{X} \to [0,1]$ for each i, where for messages $x \in \mathcal{X}^{i-1}$ by D and $y^{i-1} \in \mathcal{Y}^{i-1}$ by F the probability for a next $x \in \mathcal{X}$ sent by D is assigned such that $\sum_{x\in\mathcal{X}} p_i^D\left(x^i, y_{i-1}, x\right) = 1$.

Because we don't know an upper bound on the requests by D, instead of only defining i for a fixed $i \in \{1,\ldots,q\}$, we have now a special value $\textsc{Fin} \in \mathcal{X}$ such that $x_i = \textsc{Fin}$ marks D's final query and therefore ends the transcript. For a transcript $\tau = \left(x_1, y_1, \ldots, x_q, y_q\right)$ we define the length of the transcript $|\tau| = q$. Denote by $\mathcal{T}$ the set of complete transcripts τ, where the last query $x_{|\tau|} = \textsc{Fin}$.

To calculate the probability of a certain transcript τ, we first specify the probability of one participant acting like in the transcript: For a game like F, we have $p^F(\tau) = \prod_{i=1}^{q} p_i^F\left(x^i, y^{i-1}, y_i\right)$ and for the distinguisher D, we have $p^D(\tau) = \prod_{i=1}^{q} p_i^D\left(x^{i-1}, y^{i-1}, x_i\right)$. Here, an index in the superscript x^i is the tuple of all messages up to the ith message x_i of the transcript τ, i.e. $x^i = (x_1, \ldots, x_i)$.

In the formulation of the statement, we will consider the random variable over all possible transcripts in $\mathcal{T}$. The probability for every $\tau \in \mathcal{T}$ is defined by $\mathbf{Pr}\left[T_F^D = \tau\right] = p^D(\tau) \cdot p^F(\tau)$. Analogously to the length of the transcript τ, we define the random variable $\left|T_F^D\right|$ to be the

amount of oracle queries made by D when interacting with F. Consequently, $\mathbf{E}\left[\left|T_F^D\right|\right]$ is the expected number of oracle queries made by D.

Up to now, we have not given an obvious way for the distinguisher D to state whether they think they were interacting with F or G. We achieve this by adding a very last message to every transcript $\tau = (x_1, y_1, \ldots, x_q, y_q)$. We define the accepting probability as $p_{q+1}^D(x^q, y^q, b)$ with $b \in \{0,1\}$, such that $p_{q+1}^D(x^q, y^q, 0) + p_{q+1}^D(x^q, y^q, 1) = 1$. We denote by $D(T_F^D)$ a shorthand for the random variable of this final decision.

Using this notation, we can now define the advantage of D when distinguishing F and G: $\mathsf{Adv}_{G,F}^{\mathsf{dist}}(D) = \mathbf{Pr}\left[D\left(T_G^D\right) = 1\right] - \mathbf{Pr}\left[D\left(T_F^D\right) = 1\right]$.

When F and G are clear from the context, we define a partition of $\mathcal{T} = X^+ \dot\cup X^- \dot\cup X^=$:

$$X^+ = \left\{\tau \in \mathcal{T} : p^G(\tau) > p^F(\tau)\right\}$$
$$X^- = \left\{\tau \in \mathcal{T} : p^G(\tau) < p^F(\tau)\right\}$$
$$X^= = \left\{\tau \in \mathcal{T} : p^G(\tau) = p^F(\tau)\right\}$$

Here, for every transcript $\tau \in X^+$ the probability of τ being produced from interaction with G is higher than being produced from interaction with F.

5.2.2. Point-Wise Proximity

In this section, we define Point-Wise Proximity, which is a special case of the H-coefficient technique. The H-coefficient technique [12] is a framework for obtaining a bound on $\mathsf{Adv}_{G,F}^{\mathsf{dist}}(D)$ from a worst-case bound q on the number of oracle queries that the distinguisher D makes.

For this, X^+ is partitioned into two sets, Good and Bad, such that $X^+ = \mathsf{Good} \dot\cup \mathsf{Bad}$. Then, it is proven that $1 - {p^F(\tau)}/{p^G(\tau)} \le \epsilon(q)$ for all $\tau \in \mathsf{Good}$ with $|\tau| = q$ and that $\mathbf{Pr}\left[T_G^D \in \mathsf{Bad}\right] \le \delta(q)$. Having shown those facts, simple calculations yield that $\mathsf{Adv}_{G,F}^{\mathsf{dist}}(D) \le \epsilon(q) + \delta(q)$ for any distinguisher D making q queries and that $\mathbf{Pr}\left[T_G^D \in \mathsf{Bad}\right] \le \delta(q)$.

Pointwise Proximity is the special case of the H-coefficient technique, where $\mathsf{Bad} = \emptyset$ and therefore $X^+ = \mathsf{Good}$. Hence, $\delta(q) = 0$ satisfies $\mathbf{Pr}\left[T_G^D \in \mathsf{Bad}\right] \le \delta(q)$, and we have proven that $\mathsf{Adv}_{G,F}^{\mathsf{dist}}(D) \le \epsilon(q)$ by just showing that $1 - {p^F(\tau)}/{p^G(\tau)} \le \epsilon(|\tau|)$ for any D making at most q queries.

Theorem 17 was proven by Jaeger and Tessaro [9, Theorem 2] and expands the notion of Point-Wise Proximity to distinguishers whose amount of requests is not bounded, but their expected amount of requests is known. We define two games F and G to be ϵ-restricted if $1 - {p^F(\tau)}/{p^G(\tau)} \le \epsilon(|\tau|)$ for all $\tau \in X^+$. Here, we will consider the special case of $\epsilon(q) = {\left(\Delta \cdot q^d\right)}/{p}$ for some Δ, d and p. Since the expected runtime may vary between F and G, we make those two games interchangeable by renaming them to H and $\bar{H}$.

Theorem 17. *Let $\epsilon(q) = {(\Delta \cdot q^d)}/{N}$ for $\Delta, d, N \in \mathbb{N}_{>0}$ satisfying $N \ge \Delta \cdot 6^d$. Let (F, G) be an ϵ-restricted pair of games. Let $H \in \{F, G\}$ and $\bar{H} \in \{F, G\} \setminus \{H\}$. Then for any B,*

$$\mathsf{Adv}_{H,\bar{H}}^{\mathsf{dist}}(B) \le 5\sqrt[d]{\frac{\Delta \cdot \mathbf{E}\left[\left|T_H^B\right|\right]^d}{N}} = 5\sqrt[d]{\epsilon\left(\mathbf{E}\left[\left|T_H^B\right|\right]\right)}.$$

In the following section, we apply this theorem to prove an expected-time version of the PRP-PRF-switching Lemma.

5.2.3. Expected Time RP/RF Switching Lemma

Consider two games: The first game, R, returns a uniformly random $x \leftarrow\!\!\$\ \{1, \dots N\}$ sampled with replacement on every distinct, new request. Game Π in contrast returns uniformly random $x \leftarrow\!\!\$\ \{1, \dots N\}$ sampled without replacement. The PRP-PRF-switching lemma gives us a bound for the success of an attacker $\mathcal{A}$ succeeding in distinguishing those two games $\mathsf{Adv}^{\mathsf{dist}}_{R,\Pi}(\mathcal{A})$.

When $\mathcal{A}$ makes at most q oracle queries, we can apply the bounded version of the switching lemma [11, Lemma 4.5.12] and prove that $\mathsf{Adv}^{\mathsf{dist}}_{R,\Pi}(\mathcal{A}) \leq q^2/2N$. For the expected amount, we can apply Theorem 17 to the proof by Chang and Nandi [6] for the bounded version of the switching lemma. The following theorem is based on the considerations by Jaeger and Tessaro [9, Section 4.3, Switching Lemma].

Corollary 18 (Expected Time Switching Lemma)**.** *Let $\mathcal{A}$ be an attacker that in expectation makes $\mathsf{E}\left[\left|T^{\mathcal{A}}_{\Pi}\right|\right]$ queries to either a random function R or a random permutation Π with domains $\{1, \dots, N\}$, where $18 \leq N \in \mathbb{N}$. Then*

$$\mathsf{Adv}^{\mathsf{sl}}_{N}(\mathcal{A}) \leq 5\sqrt{\frac{\mathsf{E}\left[\left|T^{\mathcal{A}}_{\Pi}\right|\right]^2}{2N}}.$$

In the following proof, we first show that R and Π are ϵ-bounded as done by Chang and Nandi [6], and then apply Theorem 17.

Proof. Consider as Good the set of transcripts $\tau = (x_1, y_1, \dots, x_q, y_q)$, where the x_i are distinct and the y_i are distinct, which is also the set of transcripts that are likelier to be seen when interacting with random permutations $F = \Pi$ than with random functions $G = R$. Therefore we have $X^+ = \mathsf{Good}$ and $\mathsf{Bad} = \emptyset$ and have the special case of Pointwise Proximity. Next, we show ϵ-boundedness:

For a fixed τ_R that could have been seen when interacting with the random function R and τ_Π seen when interacting with the random permutation Π, for a $\tau \in \mathsf{Good}$ we have a probability that

$$\begin{aligned}
\mathbf{Pr}\left[\tau_R = \tau\right] &= \frac{1}{N^q} \\
\mathbf{Pr}\left[\tau_\Pi = \tau\right] &= \frac{1}{N} \cdot \frac{1}{N-1} \cdot \dots \cdot \frac{1}{N-q+1} \\
&= N^{-q} \cdot \frac{1}{1-\frac{1}{N}} \cdot \frac{1}{1-\frac{2}{N}} \cdot \dots \cdot \frac{1}{1-\frac{q-1}{N}} \\
&\leq N^{-q} \cdot \frac{1}{1-\frac{1+2+\dots+(q-1)}{N}} = N^{-q} \cdot \frac{1}{1-\frac{q(q-1)}{2N}}
\end{aligned}$$

Therefore we can deduce that

$$
\begin{aligned}
&\text{Set } \epsilon = \frac{q\,(q-1)}{2N} \\
&\Rightarrow 1 - \frac{q\,(q-1)}{2N} \geq 1 - \epsilon \\
&\Rightarrow \frac{1}{N^q} \geq (1-\epsilon) \cdot N^{-q} \cdot \frac{1}{1 - \frac{q(q-1)}{2N}} \\
&\Rightarrow \mathbf{Pr}\left[\tau_R = \tau\right] \geq (1-\epsilon) \cdot \mathbf{Pr}\left[\tau_\Pi = \tau\right]
\end{aligned}
$$

Since we have proven that the games Π and R are ϵ-restricted with $\epsilon\,(q) = q^2/2N$, we can now apply Theorem 17 with $\Delta = \frac{1}{2}$ and $N \geq \Delta \cdot 6^d = 18$, from which the statement directly follows. □

5.3. Forking Lemma

Adversary $\mathcal{B}_{\mathsf{E}(\mathsf{X})}(pp)$

1 : $i \leftarrow 0$
2 : $(u, s, \cdot) \leftarrow\!\$\ \mathcal{A}(pp)$
3 : $(\cdot, w) \leftarrow\!\$\ \mathsf{E}\,(\mathsf{X})^{\mathsf{Next},\mathsf{Rewind}}\,(pp, u)$
4 : return w

Figure 5.5.: Attacker $\mathcal{B}_{\mathsf{E}(\mathsf{X})}$ against Π^*.

Using the framework by Jaeger and Tessaro [9] introduced in Section 5.1 and the Expected Time Switching Lemma proven in Corollary 18, we can now state a dynamic version of the Forking Lemma [4] with the improvements by Jaeger and Tessaro [9]:

Theorem 19 (Dynamic Forking Lemma)**.** *Let* $\mathsf{PS} = (\mathsf{S}, \mathcal{R}, \mathsf{P}, \mathsf{V}, \mu)$ *be a public coin proof system. Suppose* V*'s challenges are uniformly drawn from* $S \times \mathbb{Z}_p$ *for some set* S *and* $p \in \mathbb{N}$*. Let* P^* *be a cheating prover and* $\mathcal{A}$ *be an adversary. Let* X *be a* $\Pi_{\mathsf{wit}} \vee \Pi^*$*-extractor, which generates at most* M *transcripts in expectation when combined with* T*. Define* $\mathsf{E} = \mathsf{E}(\mathsf{X})^{\mathsf{Next},\mathsf{Rewind}}$ *as shown in Figure 5.3.*

1. $\mathsf{Adv}^{\mathsf{emu}}_{\mathsf{PS},\mathsf{E}}\,(\mathsf{P}^*, \mathcal{A}) = 0$
2. *The expected number of queries that* E *makes to* Next *is less than* $\mu M + 1$*. Exactly one of these queries is made while* $i = 1$ *in* Next*, therefore there are at most* μM *challenges in expectation generated by* V*.*
3. $\mathsf{Adv}^{\mathsf{predext}}_{\mathsf{PS},\mathsf{E},\neg\Pi_{\mathsf{collision}}}\,(\mathsf{P}^*, \mathcal{A}) \leq {}^{5\mu M}/\sqrt{2p}$
4. $\mathsf{Adv}^{\mathsf{predext}}_{\mathsf{PS},\mathsf{E},\Pi_{\mathsf{wit}}}\,(\mathsf{P}^*, \mathcal{A}) \leq \mathsf{Adv}^{\mathsf{pred}}_{\mathsf{PS},\Pi^*}\left(\mathcal{B}_{\mathsf{E}(\mathsf{X})}\right) + {}^{5\mu M}/\sqrt{2p}$
5. *The expected runtime of* $\mathcal{B}_{\mathsf{E}}$ *is approximately* $T_{\mathcal{A}} + Q_{\mathsf{E}} \cdot T_{\mathsf{P}^*} + T_{\mathsf{E}}$ *where* T_x *is the worst-case runtime of* $x \in \{\mathcal{A}, \mathsf{P}^*, \mathsf{E}\}$ *and* $Q_{\mathsf{E}} < \mu N + 1$ *is the expected number of queries that* E *makes to* Next *in* $\mathsf{H}^{\mathsf{predext}}_{\mathsf{PS},\mathsf{E},\Pi^*}\,(\mathsf{P}^*, \mathcal{A})$*.*

Proof. For the first point, we recall how the transcript tr is generated: In Figure 5.3, we can see that in $\mathsf{E}(\mathsf{X})$, the tree finder is called on level 0 in line 2 and the corresponding transcript is being returned. On every layer until the end of the protocol with $i = \mu$, $\mathsf{Next}()$ is called in every case, the two new messages are appended and a recursion for the next step of the protocol is started. This results in just $\mu + 1$ calls to $\mathsf{Next}()$, which corresponds exactly to the behaviour of generating the protocol directly using $\langle \mathsf{P}_{pp}(u, w), \mathsf{V}_{pp}(u) \rangle$. Therefore, the transcripts are exactly identically distributed, so that an attacker can't have any advantage. This proves $\mathrm{Adv}^{\mathrm{emu}}_{\mathsf{PS,E}}(\mathsf{P}^*, \mathcal{A}) = 0$.

For the second statement, we know that in expectation on layer μ, at most M transcripts are being generated and therefore there are at most M calls to $\mathsf{Next}()$ in expectation on layer μ. On every layer above, by construction of the extractor, there must have been fewer calls to $\mathsf{Next}()$. Therefore a conservative bound for layers 1 to μ is μM. The extractor is only called once on layer 0 and invokes $\mathsf{Next}()$ once there, so in total we have at most $\mu M + 1$ function calls in expectation which concludes this part of the proof.

Since the verifier V is public coin, we know that the challenges c have been uniformly drawn from $S \times \mathbb{Z}_p$. The goal is to bound the probability of a collision occurring by $5\mu M/\sqrt{2p}$ with p being the size of the integer component of the challenges, and M being the expected amount of calls to the verifier. This will be achieved by applying the switching lemma to the integer component from $\mathbb{Z}_p$; if those are distinct, the whole challenge is.

For this, we consider the sampling of the random challenges during the execution of $\mathsf{E}(\mathsf{X})^{\mathsf{Next},\mathsf{Rewind}}(pp, u)$ (Figure 5.3) in the context of being called by Game $\mathsf{H}^{\mathsf{predext}}_{\mathsf{PS,E},\Pi}(\mathsf{P}^*, \mathcal{A})$ as an interaction with a random function. This interaction can be seen as communication with the game R from Corollary 18 (with incrementing inputs, so that no value is entered twice), and the rest of the game as the distinguisher D. We know that there will be at most μM queries in expectation to V as shown in the previous point. The distinguishing advantage between the random function R and the random permutation Π being used to generate V's challenges can be bounded above by

$$\mathrm{Adv}^{\mathsf{sl}}_{M}(\mathcal{A}) \leq 5\sqrt{\frac{\mathsf{E}\left[\left|T^{\mathcal{A}}_{\Pi}\right|\right]^2}{2M}} \text{ with } \mathsf{E}\left[\left|T^{\mathcal{A}}_{\Pi}\right|\right] = \mu M.$$

Because no value is used twice, in the "switched" game there is no collision because of a random permutation being used.

Therefore, the probability of a collision is $\mathrm{Adv}^{\mathsf{predext}}_{\mathsf{PS,E}[\mathsf{X}],\neg\Pi_{\mathsf{collision}}}(\mathsf{P}^*, \mathcal{A}) \leq 5\mu M/\sqrt{2p}$ which concludes the third part of the proof.

To prove the third inequality, we first prove some auxiliary equations. This first equation was part of [9, Lemma 9]:

$$
\begin{aligned}
&\mathsf{Adv}^{\mathsf{predext}}_{\mathsf{PS},\mathsf{E},\Pi_1}(\mathsf{P}^*, \mathcal{A}) \\
&= \mathbf{Pr}\left[\mathsf{V}_{pp}(u, \mathrm{tr}) \wedge \neg\Pi_1(pp, u, w) \text{ in } \mathsf{H}^{\mathsf{predext}}\right] \\
&= \mathbf{Pr}\left[\mathsf{V}_{pp}(u, \mathrm{tr}) \wedge \neg\Pi_1(pp, u, w) \wedge (\neg\Pi_2(pp, u, w) \vee \Pi_2(pp, u, w)) \text{ in } \mathsf{H}^{\mathsf{predext}}\right] \\
&= \mathbf{Pr}\left[\mathsf{V}_{pp}(u, \mathrm{tr}) \wedge \neg(\Pi_1(pp, u, w) \vee \Pi_2(pp, u, w)) \text{ in } \mathsf{H}^{\mathsf{predext}}\right] \\
&\quad + \mathbf{Pr}\left[\mathsf{V}_{pp}(u, \mathrm{tr}) \wedge \neg\Pi_1(pp, u, w) \wedge \Pi_2(pp, u, w) \text{ in } \mathsf{H}^{\mathsf{predext}}\right] \\
&\leq \mathbf{Pr}\left[\mathsf{V}_{pp}(u, \mathrm{tr}) \wedge \neg(\Pi_1(pp, u, w) \vee \Pi_2(pp, u, w)) \text{ in } \mathsf{H}^{\mathsf{predext}}\right] \\
&\quad + \mathbf{Pr}\left[\mathsf{V}_{pp}(u, \mathrm{tr}) \wedge \neg(\neg\Pi_2(pp, u, w)) \text{ in } \mathsf{H}^{\mathsf{predext}}\right] \\
&\leq \mathsf{Adv}^{\mathsf{predext}}_{\mathsf{PS},\mathsf{E},\Pi_1\vee\Pi_2}(\mathsf{P}^*, \mathcal{A}) + \mathsf{Adv}^{\mathsf{predext}}_{\mathsf{PS},\mathsf{E},\neg\Pi_2}(\mathsf{P}^*, \mathcal{A})
\end{aligned} \tag{5.1}
$$

Since the only reason for the $\mathsf{H}^{\mathsf{predext}}$ to fail is that a collision has occurred, we can additionally note that

$$
\mathsf{Adv}^{\mathsf{predext}}_{\mathsf{PS},\mathsf{E},\Pi_{\neg\perp\text{collision}}}(\mathsf{P}^*, \mathcal{A}) = \mathsf{Adv}^{\mathsf{predext}}_{\mathsf{PS},\mathsf{E},\Pi_{\mathsf{wit}}\vee\Pi^*}(\mathsf{P}^*, \mathcal{A}). \tag{5.2}
$$

The next equation uses $\mathcal{B}_{\mathsf{E}(\mathsf{X})}$ defined in Figure 5.5 to "harvest" the success probability from the predicate-extension game:

$$
\begin{aligned}
\mathsf{Adv}^{\mathsf{predext}}_{\mathsf{PS},\mathsf{E},\neg\Pi^*}(\Pi^*, \mathcal{A}) &= \mathbf{Pr}\left[\mathsf{V}_{pp}(u, \mathrm{tr}) \wedge \neg(\neg\Pi^*(pp, u, w)) \text{ in } \mathsf{H}^{\mathsf{predext}}\right] \\
&\leq \mathbf{Pr}\left[\Pi^*(pp, u, w) \text{ in } \mathsf{H}^{\mathsf{predext}}\right] \\
&= \mathbf{Pr}\left[\Pi^*(pp, \epsilon, w) \text{ in } \mathsf{H}^{\mathsf{pred}}\right] = \mathsf{Adv}^{\mathsf{pred}}_{\mathsf{S},\Pi^*}(\mathcal{B}_{\mathsf{E}(\mathsf{X})}).
\end{aligned} \tag{5.3}
$$

Using those equalities, we are now prepared to prove the fourth statement:

$$
\begin{aligned}
\mathsf{Adv}^{\mathsf{predext}}_{\mathsf{PS},\mathsf{E},\Pi_{\mathsf{wit}}}(\mathsf{P}^*, \mathcal{A}) &\overset{①}{\leq} \mathsf{Adv}^{\mathsf{predext}}_{\mathsf{PS},\mathsf{E},\Pi_{\mathsf{wit}}\vee\Pi^*}(\mathsf{P}^*, \mathcal{A}) + \mathsf{Adv}^{\mathsf{predext}}_{\mathsf{PS},\mathsf{E},\neg\Pi^*}(\mathsf{P}^*, \mathcal{A}) \\
&\overset{②}{\leq} \mathsf{Adv}^{\mathsf{predext}}_{\mathsf{PS},\mathsf{E},\Pi_{\neg\perp\text{collision}}}(\mathsf{P}^*, \mathcal{A}) + \mathsf{Adv}^{\mathsf{pred}}_{\mathsf{PS},\Pi^*}(\mathcal{B}_{\mathsf{E}(\mathsf{X})}) \\
&\overset{③}{=} 5\mu N/\sqrt{2p} + \mathsf{Adv}^{\mathsf{pred}}_{\mathsf{PS},\Pi^*}(\mathcal{B}_{\mathsf{E}(\mathsf{X})})
\end{aligned}
$$

In ①, we used Equation (5.1), in ② Equations (5.2) and (5.3) and in ③ the result from the second statement.

For the fifth point, we can easily read the runtime from Figure 5.5, because the view of $\mathsf{E}(\mathsf{X})$ is distributed identically to its view in $\mathsf{H}^{\mathsf{predext}}$ keeping the expected amount of queries to $\mathsf{Next}()$ unaltered. □

In the following chapter, an extractor for the Inner Product Argument from Protocol 14 will be given; which is an important step towards proving extraction using a linear amount of transcripts in the size of the witness.

6. Short Circuit IPA Extractor

We have introduced the Inner Product Argument and its extraction process in Chapter 4, a dynamic extraction framework in Chapter 5 and an abstraction of sampling behaviour with a conjecture on the bound of transcripts required in the sampling process in Chapter 3. Now those results can be combined to construct an emulator E for the Bulletproof Inner Product Argument. We conjecture that this emulator requires only an expected linear amount of transcripts for successful extraction.

6.1. Procedures used during Extraction

This section will provide another view on the extraction process described in Lemma 15: Here, the mathematical procedures used in the extraction process will be described. Those will later be utilized in the pseudo-code of the extractor to recursively recover the witness.

The first function is quick-extract, which takes k transcripts and a folded witness $\hat{\mathbf{w}}'$ and $\hat{\mathbf{w}}''$ in $\mathbb{F}_p^{n/k}$, and returns the recovered witnesses $\mathbf{w}'$ and $\mathbf{w}''$. As described in Lemma 15, this procedure may fail, which then guarantees a non-trivial discrete logarithm relation with $2k$ successfully extracted subtrees. The input and output parameters of this functions are:

$$\mathsf{quick\text{-}extract}\left(\left(\left[\mathbf{g}'\right],\left[\mathbf{g}''\right],\left[Q\right]\right),\left[c\right],\left(\mathbf{w}',\mathbf{w}'',\left(\mathbf{x},\mathbf{y},\cdot\right)\right)_{i\in\{1,\ldots,k\}}\right)\rightarrow\mathcal{OR}\left(\left(\mathbf{w}'\in\mathbb{F}_q^n,\mathbf{w}''\in\mathbb{F}_q^n\right),\perp\right)$$

In this function, we use the challenges to assemble the matrices $\mathbf{X} = \left[\begin{smallmatrix}\mathbf{x}^{(1)} & \cdots & \mathbf{x}^{(k)}\end{smallmatrix}\right]$ and $\mathbf{Y} = \left[\begin{smallmatrix}\mathbf{y}^{(1)} & \cdots & \mathbf{y}^{(k)}\end{smallmatrix}\right]$. Then we can recover the original witness by calculating $w' = \mathbf{Y}^{-1}\hat{w}'$ and $w'' = \mathbf{X}^{-1}\hat{w}''$. Afterwards, we verify that extraction was successful by checking $[c] = [\mathbf{g}']\,\mathbf{w}' + [\mathbf{g}'']\,\mathbf{w}'' + t\,[Q]$, and if true, we return $(\mathbf{w}', \mathbf{w}'')$ otherwise we return $\perp$. The next function we use is short-circuit, which can be called if quick-extraction failed:

$$\mathsf{short\text{-}circuit}\left(\left(\left[\mathbf{g}'\right],\left[\mathbf{g}''\right],\left[Q\right]\right),\left(\mathbf{w}',\mathbf{w}'',\left(\cdot,\cdot,\mathbf{z}\right)\right)_{i\in\{1,\ldots,2k\}}\right)\rightarrow\left(\mathrm{a}\in\mathrm{Ker}\left(\left[\begin{smallmatrix}\mathbf{g}' & \mathbf{g}'' & Q\end{smallmatrix}\right]\right)\right)$$

As a pre-condition for this function to work, quick-extract must have failed on the first k witnesses provided. First in this function, we calculate the result from quick-extraction, and additionally use the $2k$ transcripts to unconditionally extract a witness. If those two witnesses do not match, then $[\mathbf{u}]$ sent by P^* in the transcript must have been calculated in a different way to the specification in the protocol. We can extract the witnesses by calculating $\mathbf{W}'$, $\mathbf{W}''$ and $\mathbf{T}$, and comparing those as in Equation (4.11). There has to be a non-trivial discrete logarithm assumption $\left[\begin{smallmatrix}\mathbf{v}' & \mathbf{v}'' & A\end{smallmatrix}\right] \in \mathrm{Ker}\left(\left[\begin{smallmatrix}\mathbf{g}' & \mathbf{g}'' & Q\end{smallmatrix}\right]\right)$ we can recover, because otherwise we can prove that extraction must have been successful and must coincide with the result from quick-extraction.

The function unfold-dlog-break() takes a folded non-trivial discrete logarithm relation and expands it as described in the first part of the proof of Lemma 15. For this, it calculates for a $[\,\hat{\mathbf{v}}'\ \hat{\mathbf{v}}''\ A\,] \in \mathrm{Ker}([\,\hat{\mathbf{g}}'\ \hat{\mathbf{g}}''\ Q\,])$ the "unfolded" discrete logarithm relation $[\,\mathbf{x}\otimes\hat{\mathbf{v}}'\ \mathbf{y}\otimes\hat{\mathbf{v}}''\ Q\,] \in \mathrm{Ker}([\,\mathbf{g}'\ \mathbf{g}''\ Q\,])$:

$$\text{unfold-dlog-break}\left([\,\hat{\mathbf{v}}'\ \hat{\mathbf{v}}''\ A\,] \in \mathrm{Ker}([\,\hat{\mathbf{g}}'\ \hat{\mathbf{g}}''\ Q\,]), (\mathbf{x},\mathbf{y},\cdot)\right) \longrightarrow \left([\,\mathbf{v}'\ \mathbf{v}''\ A\,] \in \mathrm{Ker}([\,\mathbf{g}'\ \mathbf{g}''\ Q\,])\right) \tag{6.1}$$

We need one last function to shrink the statement for the next layer of the tree downwards:

$$\text{fold-statement}\left(([\mathbf{g}'], [\mathbf{g}''], [Q], [c]), [\mathbf{u}], (\mathbf{x},\mathbf{y},\mathbf{z})\right) \longrightarrow \left(\left[\hat{\mathbf{g}}'\right], \left[\hat{\mathbf{g}}''\right], [Q], [\hat{c}]\right)$$

Here, we calculate the folded $\left[\hat{\mathbf{g}}'\right]$, $\left[\hat{\mathbf{g}}''\right]$ and $[Q]$ as in Protocol 14, by setting $\left[\hat{\mathbf{g}}'\right] = [\mathbf{g}']\,\mathbf{y}$ $\left[\hat{\mathbf{g}}''\right] = [\mathbf{g}'']\,\mathbf{x}$ and $[\hat{c}] = [\mathbf{u}]\,\mathbf{z}$.

6.2. Witness-Extractor

In the following section, an extractor $\mathsf{X}^{\mathsf{wit}}$ extracting the witness of the Inner Product Argument in Protocol 14 in the notation of the framework defined in the previous chapter is given. We only consider the recursive step of Protocol 14 and omit the preprocessing. The results by this extractor satisfy $\Pi_{\mathsf{dlog}} \vee \Pi_{\mathsf{witness}}$ if no collision occurs.

$\mathsf{X}^{\mathsf{wit}}_{\mu}$.leaf$((m_{2\mu-1}, m_{2\mu}), \nu^{(\mu-1)})$

1: $(\mathbf{x},\mathbf{y},\mathbf{z}) \leftarrow$ parse $m_{2\mu-1}$
2: $(\mathbf{w}',\mathbf{w}'') \leftarrow$ parse $m_{2\mu}$
3: return $(w^{(\mu)} = (\mathbf{w}',\mathbf{w}''), \mathrm{aux}^{(\mu)} = (\epsilon, (\mathbf{x},\mathbf{y},\mathbf{z})))$

$\mathsf{X}^{\mathsf{wit}}_{i}$.enter$((m_{2i-1}, m_{2i}), \nu^{(i-1)} = ([\mathbf{g}'], [\mathbf{g}''], [Q], [c]))$

1: $(\mathbf{x},\mathbf{y},\mathbf{z}) \leftarrow$ parse m_{2i-1}
2: $([\mathbf{u}]) \leftarrow$ parse m_{2i}
3: $\sigma_{\mathsf{X}} \leftarrow (\nu^{(i-1)}, \ell = (), ([\mathbf{u}], (\mathbf{x},\mathbf{y},\mathbf{z})))$
4: $\nu^{(i)} = (\left[\hat{\mathbf{g}}'\right], \left[\hat{\mathbf{g}}''\right], [Q], [\hat{c}]) \leftarrow$ fold-statement$(\nu^{(i-1)}, [\mathbf{u}], (\mathbf{x},\mathbf{y},\mathbf{z}))$
5: return $(\sigma_{\mathsf{X}}, \nu^{(i)})$

$\mathsf{X}^{\mathsf{wit}}_{i}$.add$(\sigma_{\mathsf{X}}, w^{(i+1)}, \mathrm{aux}^{(i+1)})$

1: $(\nu^{(i-1)}, \ell, ([\mathbf{u}], (\mathbf{x},\mathbf{y},\mathbf{z}))) \leftarrow \sigma_{\mathsf{X}}$
2: if $w^{(i+1)}$ is a short-circuit witness then
3: $\quad w^{(i)} \leftarrow$ unfold-dlog-break$(w^{(i+1)}, \mathrm{aux}^{(i+1)})$
4: $\quad$ return $(\mathrm{done} = 1, w^{(i)})$
5: ℓ.add$((w^{(i+1)}, \mathrm{aux}^{(i+1)}))$
6: if $\ell = k$ then
7: $\quad$ if $(\mathbf{w}',\mathbf{w}'') \leftarrow$ quick-extract$(\nu^{(i-1)}, \ell)$ then
8: $\quad\quad$ return $(\mathrm{done} = 1, w^{(i)} = (\mathbf{w}',\mathbf{w}''), \mathrm{aux}^{(i)} = ([\mathbf{u}], (\mathbf{x},\mathbf{y},\mathbf{z})))$
9: elseif $\ell = 2k$ then
10: $\quad w^{(i)} \leftarrow$ short-circuit$(\nu^{(i-1)}, \ell)$
11: $\quad$ return $(\mathrm{done} = 1, w^{(i)}, \mathrm{aux}^{(i)} = ([\mathbf{u}], (\mathbf{x},\mathbf{y},\mathbf{z})))$
12: $\sigma_{\mathsf{X}} \leftarrow (\nu^{(i-1)}, ([\mathbf{u}], (\mathbf{x},\mathbf{y},\mathbf{z})))$
13: return $(\mathrm{done} = 0, w^{(i)} = \epsilon, \sigma_{\mathsf{X}})$

Figure 6.1.: The HKR-Witness-Extractor $\mathsf{X}^{\mathsf{wit}}$

Using the state variables This paragraph is about the use of the different variables of the extraction framework. From the odd messages m_{2i-1} sent by the verifier V, we can derive ξ and therefore the folding coefficients $(\mathbf{x},\mathbf{y},\mathbf{z})$. The even messages m_{2i} for $i \in \{0,\ldots,\mu-1\}$ contain the $[\mathbf{u}]$ needed for unconditional extraction. The final message $m_{2\mu}$ by the prover contains $\hat{\mathbf{w}}'$ and $\hat{\mathbf{w}}''$, which are the witness for the folded statement up to that point.

The variable $w^{(i)}$ returned by the extractor $\mathsf{X}^{\mathsf{wit}}$ when "done = 1" as the second output, or as the first output when visiting leaf nodes, contains either the two witness vectors $(\mathbf{w}',\mathbf{w}'')$ satisfying $\Pi_{\mathsf{wit}}(\pi, \nu^{(i)}, w)$, or a non-trivial discrete logarithm relation $\mathbf{a} \in \mathrm{Ker}([\,\mathbf{g}'\ \mathbf{g}''\ Q\,])$.

The additional information $\text{aux}^{(i)}$ returned along with $w^{(i)}$ contains the $([\mathbf{u}], (\mathbf{x}, \mathbf{y}, \mathbf{z}))$, which corresponds to the information of the verifier's messages sent from this round with the folding coefficients $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ and the $[\mathbf{u}]$.

In this formalization, the statement $u = v^{(0)}$ as well as the updated statements $v^{(i)}$ consist of $v^{(i)} = ([\hat{\mathbf{g}}'], [\hat{\mathbf{g}}''], [Q], [\hat{c}])$. $[\mathbf{g}']$, $[\mathbf{g}'']$ and $[Q]$ form the commitment key and $[c]$ is the commitment.

The internal state used by an extractor to collect the information about all children on a node $\sigma_{\mathsf{X}} = (v^{(i)}, \ell, ([\mathbf{u}], (\mathbf{x}, \mathbf{y}, \mathbf{z})))$ contains the updated statement $v^{(i)}$, the list of the extracted $(w^{(i+1)}, \text{aux}^{(i+1)})$ from the children, and the content of the messages needed for extraction in the parent step that is returned as $\text{aux}^{(i)}$ in line 8 and 11.

Note that while in this listing $\mathbf{w}'$ and $\mathbf{w}''$ correspond to the same entities as in the description of the Inner Product Argument in Protocol 14, $w^{(i)}$ is the returned witness variable from the pseudo-code in Figure 5.3.

Behaviour of the Extractor When visiting a leaf node with $\mathsf{X}_{\mu}^{\mathsf{wit}}.\mathsf{leaf}((m_{2\mu-1}, m_{2\mu}), v^{(\mu-1)})$, the extractor just saves the content of the messages in its returned values.

On entering a node $\mathsf{X}_{i}^{\mathsf{wit}}.\mathsf{enter}((m_{2i-1}, m_{2i}), v^{(i-1)})$, the new messages are parsed and stored in the state. Then, the updated statement $v^{(i)}$ is calculated from $v^{(i-1)}$. The state σ_{X} is initialized with the statement for this layer $v^{(i)}$, the empty list ℓ and the information that will later be used on the parent layer for extraction $([\mathbf{u}], (\mathbf{x}, \mathbf{y}, \mathbf{z}))$. Then, the statement $v^{(i)}$ and the state σ_{X} are returned.

In $\mathsf{X}_{i}^{\mathsf{wit}}.\mathsf{add}(\sigma_{\mathsf{X}}, w^{(i+1)}, \text{aux}^{(i+1)})$, the extractor receives the witness $w^{(i+1)}$ and the auxiliary input $\text{aux}^{(i+1)}$ required for extraction, which are collected in the state σ_{X} in the list ℓ. If the added witness was a non-trivial discrete logarithm relation, the relation is expanded and extraction for the subtree (and therefore recursively the whole tree) is stopped with done $= 1$ in line 4. As soon as k transcripts have been reached, quick extraction is attempted in line 7, and if it was successful, the extracted witness is returned and extraction for this subtree is stopped. If $2k$ subtrees have been added to the extractor, the non-trivial discrete logarithm relation is calculated and then returned. Otherwise, the extractor signals with done $= 0$ in line 13 that more subtree extractions are required.

6.3. Required Amount of Transcripts

When merging the tree finder T with the extractor $\mathsf{X}^{\mathsf{wit}}$, we get identical sampling behaviour as in the $(2k, \ldots, 2k)$-burst $(k, \ldots, k)$-sampling game defined in Figure 3.2. The following statement can be proven, assuming that Conjecture 10 holds.

Theorem 20. *Suppose Conjecture 10 is true. Then the Inner Product Argument in Protocol 14 is knowledge sound with knowledge error $\kappa(n) = 10\mu n/\sqrt{2p}$, with n being the size of the witnesses $\mathbf{w}' \in \mathbb{F}_p^n$ and $\mathbf{w}'' \in \mathbb{F}_p^n$.*

The corresponding extractor expected polynomial-time extractor E runs P^* at most $2n$ times in a complete protocol.

Proof. We instantiate the protocol with $k = 2$. First, we recall that $\mathsf{E}[\mathsf{X}]$ has identical sampling behaviour to the algorithm BurstTree in Figure 3.2. To observe this, we define β to be the

predicate evaluating to true when quick-extract fails to extract a witness using k successfully extracted child nodes in line 7 of $\mathsf{X}_i^{\mathsf{witness}}.\mathsf{add}()$. The execution of quick-extract only depends on the corresponding subtree, so that the decision to expand to $\hat{k}_i = 2k = 4$ children instead of $k_i = k = 2$ can be rephrased as a predicate only depending on the subtree.

Therefore, we can apply Conjecture 10 to receive an upper bound of $\alpha \prod_{i=1}^{\mu} k_i$ with $\alpha = 4/2 = 2$. Thus, at most $2^{\mu+1}$ transcripts are required in expectation. Because $\mu = \log_2(n)$ by the choice of μ in Protocol 14, we have at most $2n$ required transcripts in expectation.

Theorem 19 now gives us the error probability of the extractor with $N = 2n$, whereas Lemma 15 proves the mathematical correctness of the extraction process. □

7. Conclusion

The overall objective of this thesis was to provide an overview over quick- and short-circuit extraction in protocols like the Bulletproof Inner Product Argument, and to prepare the ground for proving linear extraction. For this, the Inner Product Argument along with the techniques used were explained and defined, and a precise proof for extraction was given. A special focus was put on quick- and short-circuit extraction. A dynamic extraction framework along with a tree finder necessary for linear extraction was proposed, and the existing proofs for static extractors were adapted to the dynamic setting.

The sampling behaviour required for quick-extraction was formalized into a simple sampling game, which can be used as a basis for a proof for the linear amount of required transcripts. A formulation of the theorem was conjectured, and a possible structure of such a proof was sketched. Additionally, an explanation of why proving this statement using known techniques might be difficult was given.

A proof for Conjecture 10 would directly yield a linear extractor for the Bulletproof Inner Product Argument. The proof itself remains an open problem for further research.

Bibliography

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

[2] Thomas Attema, Serge Fehr, and Michael Klooß. *Fiat-Shamir Transformation of Multi-Round Interactive Proofs*. Cryptology ePrint Archive, Report 2021/1377. https://ia.cr/2021/1377. 2021.

[3] Mihir Bellare and Oded Goldreich. "On defining proofs of knowledge". In: *Annual International Cryptology Conference*. Springer. 1992, pp. 390–420.

[4] Jonathan Bootle et al. "Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting". In: https://eprint.iacr.org/2016/263. May 2016, pp. 327–357. ISBN: 978-3-662-49895-8. DOI: 10.1007/978-3-662-49896-5_12.

[5] Benedikt Bünz et al. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *2018 IEEE Symposium on Security and Privacy (SP)*. https://eprint.iacr.org/2017/1066. 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020.

[6] Donghoon Chang and Mridul Nandi. *A Short Proof of the PRP/PRF Switching Lemma*. Cryptology ePrint Archive, Report 2008/078. https://ia.cr/2008/078. 2008.

[7] Ashrujit Ghoshal and Stefano Tessaro. "Tight State-Restoration Soundness in the Algebraic Group Model". In: *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III*. Ed. by Tal Malkin and Chris Peikert. Vol. 12827. Lecture Notes in Computer Science. https://eprint.iacr.org/2020/1351. Springer, 2021, pp. 64–93. DOI: 10.1007/978-3-030-84252-9_3. URL: https://doi.org/10.1007/978-3-030-84252-9%5C_3.

[8] Max Hoffmann, Michael Klooß, and Andy Rupp. "Efficient Zero-Knowledge Arguments in the Discrete Log Setting, Revisited". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 2093–2110. ISBN: 9781450367479. DOI: 10.1145/3319535.3354251. URL: https://doi.org/10.1145/3319535.3354251.

[9] Joseph Jaeger and Stefano Tessaro. "Expected-time cryptography: generic techniques and applications to concrete soundness". In: *Theory of Cryptography Conference*. Springer. 2020, pp. 414–443.

[10] Ueli Maurer. "Indistinguishability of Random Systems". In: *Advances in Cryptology — EUROCRYPT 2002*. Ed. by Lars Knudsen. Vol. 2332. Lecture Notes in Computer Science. Springer-Verlag, May 2002.

[11] Jeremias Mechler et al. *Theoretische Grundlagen der Kryptographie*. Jan. 2020.

[12] Jacques Patarin. "The "Coefficients H" Technique". In: *Selected Areas in Cryptography*. Ed. by Roberto Maria Avanzi, Liam Keliher, and Francesco Sica. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 328–345. ISBN: 978-3-642-04159-4.

[13] Torben P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *CRYPTO*. 1991.

A. Appendix

A.1. Proof Attempt for Conjecture 10

The following proof idea contains some questionable steps, which have been marked by Remark 21 and Remark 22. An overview is given in Remark 11. It still provides the structure and some ideas a proof to this problem may need; and helps to understand the problems when applying the techniques by [4, Lemma 1] or [1, Lemma 5].

When trying to calculate an exact value for $\mathbb{E}[A]$, we require the probability of a burst, which we don't know. The basic idea of this proof idea is to always take the maximum of the amount of expected lookups in H over all possible locations of bursts instead of weighting them by probability of the burst.

Note that Conjecture 10 can also be applied to protocols with $k_i = 1$, because every protocol with a $k_i = 1$ can also be seen as one with $k_i = 2$, where the additional transcript is not used at "runtime". When extending the amount of transcripts required, the value of k_i has to be increased when used in every part of Conjecture 10, i.e. also when determining $\mathbb{E}[A]$.

Proof Idea. As described previously, we take the maximum of the expected value of those two cases instead of weighting them by their probability, because the probability of a burst subtree is unknown.

$$\mathbb{E}[A_v] \leq \max\{\mathbb{E}[A_v \,|R_v], \mathbb{E}[A_v \,|\neg R_v]\}$$

To have a nicer handle for those two expectation values during the rest of our proof, we define two new functions: We define $n_{\mathsf{burst}}(m) \geq \mathbb{E}[A_v \,|R_v]$, which denotes an upper bound for the expected amount of lookups to H in a subtree generated by a call to $\textsc{BurstTree}_m(a, c_1, \ldots, c_m)$ on layer m with parameters $a, c_1, \ldots, c_{m+1}$, where some subtree was marked by β as burst. Note, that n_{burst} is an upper bound for layer m and therefore independent of the specific call parameters. The second one, $n_{\neg\mathsf{burst}}(m) \geq \mathbb{E}[A_v \,|R_v]$, is the corresponding value, but without any node being marked by β. Since in every call to $\textsc{BurstTree}_\mu$ by definition there is one lookup to H, we have $n_{\neg\mathsf{burst}}(\mu) = 1$; and since there is no call to β in $\textsc{BurstTree}_\mu$, we will never have bursts for leaves and therefore $n_{\mathsf{burst}}(\mu) = 0$.

In the first part of the proof, we want to determine $n_{\neg\mathsf{burst}}(m)$, i.e. the amount of lookups to H if β always returned 0 (as shown in Figure A.1), and prove that for $m \in \{0, \ldots, \mu\}$:

$$n_{\neg\mathsf{burst}}(m) = \prod_{i=m+1}^{\mu} k_i \tag{A.1}$$

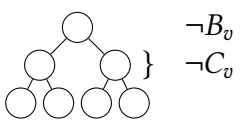


Figure A.1.: $\neg B_v \wedge \neg C_v$

We will use an induction over m starting from $m = \mu$ going down towards $m = 0$ to prove this equation. As the induction hypothesis, we use Equation (A.1). For the base case $m = \mu$, we observe that by definition

$$n_{\neg\text{burst}}(m) = 1 = \prod_{i=\mu+1}^{\mu} k_i$$

For the induction step, we consider an execution of $\textsc{BurstTree}_m(a, c_1, \ldots, c_m)$ with arbitrary but fixed $a \in \{1, \ldots, R\}$ and $c_1, \ldots, c_m \in \{1, \ldots, N\}$. We now know that Equation (A.1) holds for $m+1$, i.e. for all calls to $\textsc{BurstTree}_m(a, c_1, \ldots, c_m)$ with arbitrary $c_{m+1} \in \{1, \ldots, N\}$, and want to show that it holds for m as well.

To determine the amount of calls to $\textsc{BurstTree}_{m+1}$, we first note that the requested amount of transcripts must have been $K_v = k_{m+1}$ because we are conditioned on β not marking any subtree.

Denote by $\epsilon(a, c_1, \ldots, c_m)$ the fraction of 1-entries within the submatrix $H(a, c_1, \ldots, c_m)$. Since the criterion to enter $\textsc{BurstTree}_m(a, c_1, \ldots, c_m, c_{m+1})$ for the challenge c_{m+1}, is whether the first transcript generated from there is accepting, we have for this sequence of challenges $v = (a, c_1, \ldots, c_m)$ that the probability to continue extraction $\mathsf{Pr}[E_v] = \epsilon(a, c_1, \ldots, c_m)$.

Now we also assume that $E_v = 1$, i.e. the first call to $\textsc{BurstTree}_{m+1}$ was accepting, and we want to find the amount of calls to $\textsc{BurstTree}_{m+1}$ for the $k_{m+1} - 1$ remaining successful runs of $\textsc{BurstTree}_{m+1}$. For this, we first note that for the experiment of first choosing a c_{m+1} uniformly random from $\{1, \ldots, N\}$ and then running $\textsc{BurstTree}_{m+1}(a, c_1, \ldots, c_{m+1})$ and testing if it was successful is

$$\frac{1}{N}\left(\sum_{i=1}^{N} \epsilon(a, c_1, \ldots, c_m, c_{m+1})\right) = \epsilon(a, c_1, \ldots, c_m).$$

Therefore, we know that $\mathbb{E}[O_v \,|E_v = 1 \wedge K_v = k_{m+1}]$ is distributed according to the negative binomial distribution with success probability of $\epsilon = \epsilon(a, c_1, \ldots, c_m) \in [0, 1]$:

$$\mathbb{E}[O_v \,|E_v = 1 \wedge K_v = k_{m+1}] = \frac{(k_{m+1} - 1) \cdot (1 - \epsilon)}{\epsilon} \leq \frac{k_{m+1} - 1}{\epsilon}$$

Combining those results, we get

$$\begin{aligned}\mathbb{E}[O_v \,|K_v = k_{m+1}] &= \sum_{\ell=0}^{N} \mathbf{Pr}[E_v = 1 \,|K_v = k_{m+1}] \cdot \mathbb{E}[O_v \,|E_v = 1 \wedge K_v = k_{m+1}] \\ &\overset{(1)}{=} \sum_{\ell=0}^{N} \mathbf{Pr}[E_v = 1] \cdot \mathbb{E}[O_v \,|E_v = 1 \wedge K_v = k_{m+1}] \leq \epsilon \cdot \frac{k_{m+1} - 1}{\epsilon} = k_{m+1} - 1\end{aligned}$$

Remark 21. This equation is the first of the questionable statements found during the finalization of the thesis. Here, in (1), the following equality was used:

$$\mathbf{Pr}[E_v = 1 \,|K_v = k_{m+1}] = \mathbf{Pr}[E_v = 1]$$

It was assumed here that the probability of entering the node E_v because the first transcript generated was accepting is independent from the amount of children K_v visited if the node was entered. This assumption cannot be made in general, because the nodes in which a burst would occur ($K_v = \hat{k}_i$) might have a smaller ratio of accepting transcripts than those with regular extraction and therefore might be more difficult to enter.

This also makes the basic technique of [4, Lemma 1] harder to apply, which was having the probability $\epsilon = \epsilon(a, c_1, \ldots, c_m)$ of entering a node cancelling itself out with the increased difficulty to find accepting transcripts starting from this node ${}^{(k_{m+1}-1)}/_{\epsilon}$. Instead we would now have to consider two types of children — those, that would result in a burst if entered, and those that would not.

The basic idea of this proof was omitting the unknown probabilities of burst events by instead using a maximum of the probabilities. With those two different "classes" of results when calling BurstTree$_{m+1}$ with different challenges, we have two probabilities for those calls on the next level to be successful, say ϵ_{burst} and $\epsilon_{\neg\text{burst}}$, for which the achieving such an cancellation of the factor is not obvious.

Along with the additional first successful call to BurstTree$_{m+1}$, we have k_{m+1} calls to BurstTree$_{m+1}$ and therefore lookups in H in total. So by multiplying this value with the expected amount of lookups in H provided by the induction hypothesis, we get

$$n_{\neg\text{burst}}(m) \leq k_{m+1} \cdot n_{\neg\text{burst}}(m+1) \leq k_{m+1} \cdot \left(\prod_{i=m+2}^{\mu} k_i \right) = \prod_{i=m+1}^{\mu} k_i$$

This concludes the induction for $n_{\neg\text{burst}}$. Next, we turn to n_{burst}: Here, we will consider a subcall to BurstTree$_m$ again with prover randomness a and $c_1, \ldots, c_m$ as challenges, which we denote by the implicit node $v = (a, c_1, \ldots, c_m)$. Because $n_{\text{burst}}(m) = \mathbb{E}[A_v \,|R_v]$, we are now conditioned on some subtree being marked by β. We recall that $R_v = B_v \vee C_v$, which allows us to split up $\mathbb{E}[A_v \,|R_v]$ into:

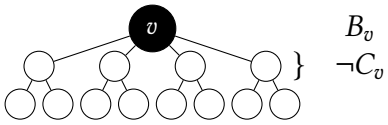


Figure A.2.: $B_v \wedge \neg C_v$

$$\mathbb{E}[A_v \,|R_v] \leq \max\{\mathbb{E}[A_v \,|B_v \wedge \neg C_v], \mathbb{E}[A_v \,|\neg B_v \wedge C_v], \mathbb{E}[A_v \,|B_v \wedge C_v]\}$$

We will now consider all three cases: $\mathbb{E}[A_v \,|B_v \wedge \neg C_v]$ as shown in Figure A.2 is conditioned on β only having marked the full tree (B_v), but none of the subtrees ($\neg C_v$). An example for this case can be found in Figure A.2. Because of this, BurstTree$_m$ will now call BurstTree$_{m+1}$ exactly $K_v = \hat{k}_{m+1}$ times and perform a regular extraction without any burst for all children, which again by the argument used for $n_{\neg\text{burst}}$ requires $\mathbb{E}\left[O_v \middle| K_v = \hat{k}_{m+1}\right] = \hat{k}_{m+1} - 1$ additional calls to BurstTree$_{m+1}$ in expectation. Therefore we have, together with the first call to BurstTree$_{m+1}$:

$$\mathbb{E}[A_v \,|B_v \wedge \neg C_v] \leq \hat{k}_{m+1} \cdot n_{\neg\text{burst}}(m+1) = \hat{k}_{m+1} \cdot n_{\neg\text{burst}}(m+1)$$

$\mathbb{E}[A_v \,|B_v \wedge C_v]$ (Figure A.3) is conditioned on β marking the full tree and at least an additional subtree. Denote by $v_1, \ldots, v_k$ the children of v, and recall that $C_v = R_{v_1} \vee \cdots \vee R_{v_k}$. The children in whose subtrees a node was marked by β are collected in the random variable $L_v = \{w \in \text{children}(v) \,|R_w = 1\}$. Since $R_v = 1$, we know that $|L_v| \geq 1$.

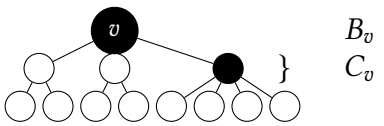


Figure A.3.: $B_v \wedge C_v$

When considering the sampling of $c_{M+1} \leftarrow\!\!\$\ \{1, \ldots, N\}$ together with the following call to the function $\textsc{BurstTree}_{m+1}\ (a, c_1, \ldots, c_{m+1})$ as one procedure, all of the executions (and therefore all executions, where the first subcall doesn't abort) have identical starting conditions and don't interfere with each other, so that the probability of a burst is equal for every call.

We now know, that there were at most $\hat{k}_{m+1}$ successful runs of $\textsc{BurstTree}_{m+1}$, and because of $C_v = 1$ that among those there was one burst event. To simplify the analysis, we first assume that there was exactly one burst event and therefore $|L_v| = 1$, and then show that this yields the highest expected amount of calls. In this case, the position of the call to $\textsc{BurstTree}_{m+1}$ with the burst is uniformly distributed in $\left\{1, \ldots, \hat{k}_{m+1}\right\}$, which yields an expected amount of $\left(1+\hat{k}_{m+1}\right)/2$ subcalls to reach the burst.

If there were more than one subcalls with burst events, this value is lower and therefore this case doesn't need to be considered for our upper bound because of the following argument: Assume there were $1 \leq x = |L_v| \leq \hat{k}_{m+1}$ subtrees found with burst events. Then we can consider their positions of uniformly sampled x-element subsets of $\left\{1, \ldots, \hat{k}_{m+1}\right\}$; and we are interested in the minimum of the subset sampled. We can sample this x-subset by first choosing uniformly random a value from $\left\{1, \ldots, \hat{k}_{m+1}\right\}$ and removing it from the set, and continuing with this procedure until x elements have been removed. Since the first sampling step is identical to the case where we only have $x = 1$, we will only get a smaller minimum when continuing with this procedure.

So, on average, conditioned on one child out of the $\hat{k}_{m+1}$ has a child marked by β, we need to traverse $\leq \left(\hat{k}_{m+1}+1\right)/2$ child subtrees until we have found the one where a child was marked by β. In the call to $\textsc{BurstTree}_m$, along with the argument used for $n_{\neg\mathsf{burst}}$, to find the $\leq \left(\hat{k}_{m+1}+1\right)/2$ additional children we need $\left(\hat{k}_{m+1}-1\right)/2$ additional calls to $\textsc{BurstTree}_{m+1}$. Taking into account the first call to $\textsc{BurstTree}_{m+1}$, we get $\hat{k}_{m+1}-1/2$ regular extractions followed by one extraction with a burst in expectation, which totals to

$$\mathbb{E}\left[A_v \,|B_v \wedge C_v\right] \leq \frac{\hat{k}_{m+1} - 1}{2} \cdot n_{\neg\mathsf{burst}}\,(m+1) + n_{\mathsf{burst}}\,(m+1) \tag{A.2}$$

Remark 22. In Equation A.2, another problem was found: Here, the value $n_{\mathsf{burst}}(m) = \mathbb{E}\left[A_v \,|R_v\right]$ for a node v on layer m was used. This is the expected amount of transcripts requested for a subcall to $\textsc{BurstTree}_{m+1}$, independent of whether extraction was successful or not. Using this value is not correct here, because we would have required the probability of an extraction with a burst being successful at this point.

Fixing this problem does not seem to be trivial, and has been explained in further detail in Remark 11.

$\mathbb{E}\left[A_v \,|\neg B_v \wedge C_v\right]$ as displayed in Figure A.4 can be bounded with the same argument as $\mathbb{E}\left[A_v \,|B_v \wedge C_v\right]$, but with a lower upper bound of $k_{m+1} \leq \hat{k}_{m+1}$ on the amount of successful calls to $\textsc{BurstTree}_{m+1}$ required. Because we want to find the maximum of all cases in Equation (A.2), and $\mathbb{E}\left[A_v \,|\neg B_v \wedge C_v\right] \leq \mathbb{E}\left[A_v \,|B_v \wedge C_v\right]$, we don't need this case for our analysis.

In the previous paragraphs, we have shown that Equation (A.2) can be written as:

$$n_{\text{burst}}(m) \leq \max\left\{\frac{\hat{k}_{m+1}-1}{2}\cdot n_{\neg\text{burst}}(m+1)+n_{\text{burst}}(m+1), \hat{k}_{m+1}\cdot n_{\neg\text{burst}}(m+1)\right\} \tag{A.3}$$

The last step of the proof is now to bound this expression for $\forall m \in \{0,\ldots,\mu\}$ by

$$n_{\text{burst}}(m) \leq \alpha \cdot \prod_{i=m+1}^{\mu} k_i \tag{A.4}$$

Equation (A.4) serves as the induction hypothesis here, again; and the base case $m = \mu$ corresponding to the leaves of the tree of 1-entries is

$$n_{\text{burst}}(\mu) = 0 \leq \alpha \cdot \prod_{i=m+1}^{\mu} k_i = \alpha$$

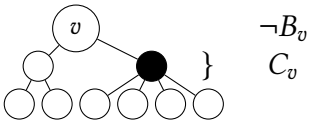


Figure A.4.: $\neg B_v \wedge C_v$

For the induction step, we have that Equation (A.4) holds for $m+1$ and want to prove that this is also the case for m. Recall our choice of α where

$$\alpha = \max_{m\in\{1,\ldots,\mu\}}\left\{\frac{\hat{k}_m}{k_m}\right\} \tag{A.5}$$

This guarantees us, that

$$\forall m \in \{1,\ldots,\mu\} : \hat{k}_m \leq \alpha \cdot k_m \tag{A.6}$$

and that

$$\forall m \in \{1,\ldots,\mu\} : \left(\frac{\hat{k}_m-1}{2}+\alpha\right) = \left(\frac{\hat{k}_m-1}{2}+\frac{\hat{k}_m}{k_m}\right) \overset{(2)}{\leq} \hat{k}_m - \frac{1}{2} \overset{(3)}{\leq} \alpha \cdot k_m. \tag{A.7}$$

In (2), we used that by the statement of the conjecture, we have that $k_m \geq 2$, and in (3), we applied equation (A.6).

Now we can bound the first element of the maximum in Equation (A.3) by

$$\begin{aligned}
n_{\text{burst}}(m) &\leq \frac{\hat{k}_{m+1}-1}{2}\cdot n_{\neg\text{burst}}(m+1)+n_{\text{burst}}(m+1)\\
&= \frac{\hat{k}_{m+1}-1}{2}\cdot\prod_{i=m+2}^{\mu} k_i + n_{\text{burst}}(m+1)\\
&\overset{(4)}{\leq} \frac{\hat{k}_{m+1}-1}{2}\cdot\prod_{i=m+2}^{\mu} k_i + \alpha\cdot\prod_{i=m+2}^{\mu} k_i\\
&= \left(\frac{\hat{k}_{m+1}-1}{2}+\alpha\right)\cdot\prod_{i=m+2}^{\mu} k_i \overset{(5)}{=} \alpha\cdot\prod_{i=m+1}^{\mu} k_i
\end{aligned}$$

For (4), we applied the induction hypothesis, and for (5), we used the definition of α from Equation (A.7). The bound for the second element of the maximum in Equation (A.3) can be verified using

$$n_{\mathsf{burst}}(m) \leq \hat{k}_{m+1} \cdot n_{\neg\mathsf{burst}}(m+1) \overset{(6)}{\leq} \alpha \cdot k_{m+1} \cdot \left(\prod_{i=m+2}^{\mu} k_i \right) \overset{(7)}{=} \alpha \cdot \prod_{i=m+1}^{\mu} k_i$$

Here, the induction hypothesis was applied in step (6), and in (7), we used the definition of α from Equation (A.5). This concludes this second proof by induction.

Summarizing the whole proof, we have now seen that in total the amount of lookups of H-entries in the burst collision game is

$$\mathbb{E}[A] \leq \max\{n_{\mathsf{burst}}(0), n_{\neg\mathsf{burst}}(0)\} \leq \max\left\{ \alpha \left(\prod_{i=1}^{\mu} k_i \right), \prod_{i=1}^{\mu} k_i \right\} \leq \alpha \left(\prod_{i=1}^{\mu} k_i \right).$$

□

A.2. Quick-Extraction for the Bulletproof Range Proof

In the Bulletproof Range Proof, the prover attempts to convince the verifier that the given commitment $[V] \in \mathbb{G}$ hides a number $v \in \mathbb{F}_p$ in a certain range.

The range proof was originally introduced in [5, Section 4]. The description of the protocol used here is analogous to [5], but the additive group notation was used instead of the multiplicative. The number n corresponds to the bit length of the upper bound of the range $\{0, \ldots, 2^n - 1\}$ we want to prove membership in.

We directly include the modifications from Section 4.2 and 4.3 of the original Bulletproof paper. Therefore the version for m batched range proofs is presented here. Set $m = 1$ for the version without the batching introduced in Section 4.1 of [5].

For a fixed integer $i \in \mathbb{F}^p$, we denote by $\mathbf{i}^n$ the vector of the first n exponentiations of n, i.e. $\mathbf{i}^n = (\, i \; i^2 \; \cdots \; i^n \,)^\mathsf{T} \in \mathbb{F}_p^n$. The vector $\mathbf{i}^{-n}$ corresponds to $\mathbf{j}^n$ with $j = -i$. The common reference string, which will be used to form the commitment keys, is $[\, \mathbf{g} \; \mathbf{h} \; g \; h \,] \in \mathbb{G}^{2nm+2}$. Note, that $[\mathbf{g}] \in \mathbb{G}^{nm}$ and $[g] \in \mathbb{G}$ are two distinct variables.

As before, we can also phrase the statement as a relation $\mathcal{R}_{\mathsf{RP}}$:

$$([\mathbf{V}]\,; \mathbf{v}, \boldsymbol{\gamma}) \in \mathcal{R}_{\mathsf{RP}} \Leftrightarrow [\mathbf{V}] \in \mathbb{G}^m \wedge \mathbf{v}, \boldsymbol{\gamma} \in \mathbb{F}_p^m \wedge \\ \forall i \in \{1, \ldots, m\} : \; [\mathbf{V}_i] = v_i\,[g] + \gamma_i\,[h] \wedge v_i \in \{0, \ldots, 2^n - 1\} \tag{A.8}$$

For an intuition of the protocol, we refer to [5, Section 4.1]. The function $\delta(y, z) = \left(z - z^2\right) \cdot \langle \mathbf{1}^{n \cdot m}, \mathbf{y}^{n \cdot m} \rangle - \sum_{j=1}^{m} \left(z^{j+2} \cdot \langle \mathbf{1}^n, \mathbf{2}^n \rangle\right)$ only depends on the challenges y and z used in the definition of the protocol, and not on any secret value.

Protocol 23 (Bulletproof Range Proof RP)**.**
Preprocessing.

$\mathscr{P}$ Pick $(\mathbf{a}_L)_i \in \{0,1\}^n$ such that $\langle (\mathbf{a}_L)_i, \mathbf{2}^n \rangle = v$. Combine them to $\mathbf{a}_L \in \{0,1\}^{nm} \subseteq \mathbb{F}_p^{nm}$. Set $\mathbf{a}_R = \mathbf{a}_L - \mathbf{1}^n$.
Commit to $\mathbf{a}_L$ and $\mathbf{a}_R$ with $[A] = \mathbf{a}_L\,[\mathbf{g}] + \mathbf{a}_R\,[\mathbf{h}] + \alpha\,[h]$ using the blinding coefficient $\alpha \leftarrow\!\!\$\, \mathbb{F}_p$.
Sample the hiding vectors $\mathbf{s}_L, \mathbf{s}_R \leftarrow\!\!\$\, \mathbb{F}_p^{n \cdot m}$ and commit to them with $[S] = \mathbf{s}_L\,[\mathbf{g}] + \mathbf{s}_R\,[\mathbf{h}] + \rho\,[h]$ using the blinding coefficient $\rho \leftarrow\!\!\$\, \mathbb{F}_p$.

$\mathscr{P} \rightarrow \mathscr{V}$: Send $[A]$ and $[S]$.

Step 1.

$\mathscr{V} \rightarrow \mathscr{P}$ Sample and send $y, z \leftarrow\!\!\$\, \mathbb{F}_p$.

$\mathscr{P}$ Define the polynomials $\mathbf{l}(X) = (\mathbf{a}_L - z \cdot \mathbf{1}^{n \cdot m}) + \mathbf{s}_L \cdot X \in \mathbb{F}_p^{n \cdot m}[X]$ and $\mathbf{r}(X) = \mathbf{y}^n \circ (\mathbf{a}_R + z \cdot \mathbf{1}^{n \cdot m} + \mathbf{s}_R \cdot X) + \left(z^2 \cdot \mathbf{z}^n\right) \otimes \mathbf{2}^n \in \mathbb{F}_p^{n \cdot m}[X]$.
Extract the coefficients t_0, t_1 and t_2 from $t(X) = \langle \mathbf{l}(X), \mathbf{r}(X) \rangle = t_0 + t_1 \cdot X + t_2 \cdot X \in \mathbb{F}_p[X]$.
Commit to t_1 and t_2 with $[T_i] = t_i\,[g] + \tau_i\,[h]$ for $i \in \{1, 2\}$ using blinding coefficients $\tau_1, \tau_2 \leftarrow\!\!\$\, \mathbb{F}_p$.
Set $\Gamma = \left\langle z^2 \cdot \mathbf{z}^m, \boldsymbol{\gamma} \right\rangle$.

$\mathscr{P} \rightarrow \mathscr{V}$ Send the commitments $[T_1]$ and $[T_2]$.

Step 2.

$\mathscr{V} \rightarrow \mathscr{P}$ Sample and send $x \leftarrow\!\!\$\, \mathbb{F}_p$.

$\mathscr{P}$ Evaluate the polynomials $\mathbf{l} = \mathbf{l}(x) \in \mathbb{F}_p^n$ and $\mathbf{r} = \mathbf{r}(x) \in \mathbb{F}_p^n$.
Calculate the combined blinding factor for the t_i by $\tau_x = \tau_2 \cdot x^2 + \tau_1 \cdot x + \Gamma \in \mathbb{F}_p$.

$\mathscr{P} \rightarrow \mathscr{V}$ Send $\mu = \alpha + \rho \cdot x$, $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$ and τ_x.

Verification.

$\mathscr{P}, \mathscr{V}$ Set $[\mathbf{h}'] = \mathbf{y}^{-n} \circ [\mathbf{h}] = \left[h_1 \; h_2^{y^{-1}} \; h_3^{y^{-2}} \; \ldots \; h_n^{y^{-n+1}} \right] \in \mathbb{G}^n$.
Set $[P] = [A] + x\,[S] - z\,[\mathbf{g}] + \left(z \cdot \mathbf{y}^{n \cdot m} + z^2 \cdot \mathbf{2}^n\right)[\mathbf{h}'] \in \mathbb{G}$.

$\mathscr{V}$ Verify that the polynomial $\mathbf{t}$ has been calculated correctly by checking $\hat{t}\,[g] + \tau_x\,[h] \stackrel{?}{=} \left(z^2 \cdot \mathbf{z}^m\right)[\mathbf{V}] + \delta(y,z)\,[g] + x\,[T_1] + x^2\,[T_2]$.

$\mathscr{P} \leftrightarrow \mathscr{V}$ We want to verify, that $[P] \stackrel{?}{=} \mathbf{l}\,[\mathbf{g}] + \mathbf{r}\,[\mathbf{h}'] + \mu\,[h]$ is a commitment to two vectors $\mathbf{l}$ and $\mathbf{r}$ with $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$ and known blinding coefficient $\mu\,[h]$.
This can be achieved by sending $\mathbf{l}$ and $\mathbf{r}$ directly using linear communication.
Alternatively, we can use Inner Product Argument from Protocol 14 with $\mathbf{w}' = \mathbf{l}$, $\mathbf{w}'' = \mathbf{r}$ and $[c] = [P] - \mu\,[h] + \hat{t}\,[h]$ and the public common reference string $[\mathbf{g}'] = [\mathbf{g}]$, $[\mathbf{g}''] = [\mathbf{h}]$ and $[Q] = [h]$.

In the following, quick extraction for the range proof will be sketched. Theorem 3 from [5] gives us unconditional extraction — and therefore also short-circuit extraction — for a $(m+2, nm, 3)$-tree of transcripts, i.e. $m+2$ different challenges for z, nm different challenges for y and 3 different challenges for x. This section of the appendix now explains quick-extraction using a $(m, 1, 3)$-tree of transcripts.

First, we reverse the calculations of Step 2: In this extraction step, the extractor knows the fixed y and z and has three different challenges $x^{(1)}$, $x^{(2)}$ and $x^{(3)}$.

To recover the polynomials $\mathbf{l}(X)$ and $\mathbf{r}(X)$, we need the evaluations $\mathbf{l}(x^{(i)})$ and $\mathbf{r}(x^{(i)})$ from two transcripts.

$$\begin{aligned}
\mathbf{l}\left(x^{(i)}\right) &= (\mathbf{a}_L - z \cdot \mathbf{1}^{n \cdot m}) + \mathbf{s}_L \cdot x^{(i)} \\
\mathbf{r}\left(x^{(i)}\right) &= \mathbf{y}^{n \cdot m} \circ \left(\mathbf{a}_R + z \cdot \mathbf{1}^{n \cdot m} + \mathbf{s}_R \cdot x^{(i)}\right) + \left(\left(z^2 \cdot \mathbf{z}^m\right) \otimes \mathbf{2}^n\right)
\end{aligned}$$

Therefore we can calculate

$$\begin{aligned}
\mathbf{s}_L &= \frac{\mathbf{l}(x_1) - \mathbf{l}(x_2)}{x_1 - x_2} \\
\mathbf{s}_R &= \frac{\mathbf{r}(x_1) - \mathbf{r}(x_2)}{x_1 - x_2} \circ \mathbf{y}^{-n} \\
\mathbf{a}_L &= \mathbf{l}(x_1) - \mathbf{s}_L \cdot x_i + z \cdot \mathbf{1}^{n \cdot m} \\
\mathbf{a}_R &= \left(r(x_1) - \left(\left(z^2 \cdot \mathbf{z}^m\right) \otimes \mathbf{2}^n\right)\right) \circ \mathbf{y}^{-n} - z \cdot \mathbf{1}^{n \cdot m} - \mathbf{s}_R \cdot x_i
\end{aligned}$$

With those values extracted, we have also found the polynomials $\mathbf{l}(X)$ and $\mathbf{r}(X)$. Slicing $\mathbf{a}_L$ into m vectors of length n also allows us to calculate the v_i using their binary representation.

Two different challenges $x^{(1)}$ and $x^{(2)}$ also allows us to find α and ρ:

$$\begin{bmatrix} 1 & x^{(1)} \\ 1 & x^{(2)} \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \rho \end{bmatrix} = \begin{bmatrix} \mu^{(1)} \\ \mu^{(2)} \end{bmatrix}$$

All matrices in this section can be inverted, because they are (sometimes scaled) Vandermonde matrices for distinct challenges. The weighted sum of the blinding factors Γ, which we defined by $\Gamma = \sum_{j=1}^{m} z^{j+1} \cdot \gamma_j$, can also be extracted using three distinct challenges:

$$\begin{bmatrix} 1 & x^{(1)} & (x^{(1)})^2 \\ 1 & x^{(2)} & (x^{(2)})^2 \\ 1 & x^{(3)} & (x^{(3)})^2 \end{bmatrix} \cdot \begin{bmatrix} \Gamma \\ \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} \hat{t}^{(1)} \\ \hat{t}^{(2)} \\ \hat{t}^{(3)} \end{bmatrix}$$

This concludes extraction for Step 2. For Step 1, we only need to calculate from m different extracted $\Gamma^{(i)}$ the blinding factors $\boldsymbol{\gamma}$:

$$\begin{bmatrix} (z^{(1)})^2 & \cdots & (z^{(1)})^{m+1} \\ \vdots & & \vdots \\ (z^{(m)})^2 & \cdots & (z^{(m)})^{m+1} \end{bmatrix} \cdot \begin{bmatrix} \gamma_1 \\ \vdots \\ \gamma_m \end{bmatrix} = \begin{bmatrix} \xi^{(1)} \\ \vdots \\ \xi^{(m)} \end{bmatrix}$$

This concludes quick extraction, because both $\mathbf{v}$ and $\boldsymbol{\gamma}$ have been recovered. The extractor now checks whether the extracted values match the commitment. If so, extraction was successful, otherwise a non-trivial discrete logarithm relation has been found.

Because of the transcripts used were successful, the verification equations hold for the extracted values. The unconditionally extracted values from the proof of [5, Theorem 3] also have to fulfill those equations. Therefore we have found a second opening for either $[A]$, $[S]$, $[V]$ or one of the $[T_i]$, which corresponds to a discrete logarithm-relation.