Tackling Bulletproofs

Christian Lewe, Eric Sirion

March 13, 2023

(*Snorlax uses Tackle on Team Rocket in bulletproof armor*)

Contents

1 Introduction

1.1 Note

In this document, we write "Bulletproofs" to refer to the scheme and "bulletproofs" to refer to the individual proof(s).

1.2 What Are Bulletproofs?

Bulletproofs are a system to efficiently prove *knowledge* of some value to another party in *zero-knowledge.* Often, an *inner product argument* is used in the background. Inner products can encode all sorts of things, such as all NP languages (word problem) and all arithmetic circuits (circuit satisfiability). This means that the statements that bulletproofs prove are (at least) NP-hard, which is enough power to encode practically all problems.

There are many upsides to using Bulletproofs as opposed to other schemes like KZG and FRI. The name "Bulletproofs" means "short as a bullet with bulletproof security assumptions", which hints at the logarithmic proof size and the conservative security assumptions. Bulletproofs are shorter than other proofs (FRI) although there exist schemes with proofs of constant size (KZG). Small proof size is perfect for blockchain applications, where block space is extremely valuable. Bulletproofs rely on the hardness of the *discrete logarithm*, which is a well-understood problem and which already underlies the security of digital signatures in Bitcoin and other cryptocurrencies. Other schemes assume pairings (KZG) or collision-resistant hashes (FRI), which are new assumptions that need to be adopted first. Bulletproofs are *transparent*, which means that anyone can create a proof that anyone can verify, without prior communication. There is no *trusted setup* like in KZG, where a shared secret (*common reference string*) is created among all participants in a *trusted setup ceremony*, which is used in all proofs.

One downside of bulletproofs is that they take linear time to verify, as opposed to logarithmic time (FRI) or constant time (KZG). This makes bulletproofs useless for applications that try to compress large computations (validity rollups).

There were many buzzwords in the past few paragraphs, terms that lacked any definition. I marked them in *italics.* As we move forward in this document, we will define these terms one by one in logical order. For now, they should give you a general overview of Bulletproofs and enable you to do some online searches.

1.3 What Is This Document?

In this document we build a deep understanding of what bulletproofs are and how they work. This write-up resulted from a reading group that I hosted, where we read *From Zero (Knowledge) to Bulletproofs*, which I highly recommend you read alongside the present document. I learned a lot, but there were many points where I wanted to dive deeper, in particular the proofs. In this document we try to explain everything from first principles, to the degree possible. We assume some familiarity with elliptic curves and cryptographic schemes, although this is explained, too. Everything related to zero-knowledge proofs is explained in great detail, including the foundations and intuition that are sometimes skimmed over. Most importantly, this is a technical write-up, so if you are looking for a high-level view without maths, better leave now; if you want to dive deeper, then welcome to paradise.

1.4 Resources

Here are some helpful resources that you should check out. In particular, you should read *From Zero (Knowledge) to Bulletproofs* which this document builds upon.

- Intuition: *The Strange Cave of Ali Baba*
- Main: *From Zero (Knowledge) to Bulletproofs*
- More detail: *Proofs, Argument, and Zero-Knowledge*
- Vitalik explains Bulletproofs: *Halo and more: exploring incremental verification and SNARKs without pairings*
- Other perspective on Bulletproofs: *Inner Product Arguments*
- Refresher on number theory: *Explained from first principles*
- List of resources: *Awesome zero knowledge proofs*

1.5 Elliptic Curves

The maths starts here; you have been warned. Check out the refresher on number theory if any of the following seems fuzzy.

An *elliptic curve* $\mathbb{E}$ is a set of two-dimensional *points* $P = (x, y) \in \mathbb{E}$ such that the coordinates $x, y \in \mathbb{F}$ stem from some field $\mathbb{F}$. In our case, we deal with the field $\mathbb{F} = \mathbb{Z}_p$ of integers modulo some number $p \in \mathbb{N}$. The points on the curve satisfy the equation $y^2 = x^3 + ax + b$ for some coefficients $a, b \in \mathbb{N}$.

The curve's operation operation is the *addition* $P + Q$ of two points $P, Q \in \mathbb{E}$ on the curve. *(We use the additive notation).* Point addition is *associative* $P + (Q + R) = (P + Q) + R$ and *commutative* $P + Q = Q + P$ for all points $P, Q, R \in \mathbb{E}$. There is an (additive) *identity* $\mathcal{O} \in \mathbb{E}$, called the *point at infinity*, which satisfies $\mathcal{O} + P = P + \mathcal{O} = P$ for all $P \in \mathbb{E}$. Furthermore, every point $P \in \mathbb{E}$ has an *inverse* $-P \in \mathbb{E}$ such that $P + (-P) = \mathcal{O}$.

We choose $\mathbb{E}$ such that its order is a prime number $n \in \mathbb{N}$. This means that $\mathbb{E}$ is cyclic, i.e., any point $P \in \mathbb{E} \setminus \{\mathcal{O}\}$ can be added onto itself n times to reach all points in $\mathbb{E}$ (exactly once) in a cycle which arrives at $\mathcal{O}$. (The loop is closed in step $n+1$ where we arrive at P again). We call points that can generate the entire curve in this manner *generators*, and in fact every point but $\mathcal{O}$ is a generator. Usually, we pick an arbitrary point $G \in \mathbb{E}$ and call it *the* generator of $\mathbb{E}$.

There is a scalar field $\mathbb{S}$ that is isomorphic to the curve $\mathbb{E}$. These are the integers $\mathbb{Z}_n$ modulo n. *Scalar multiplication* $n \cdot P$ refers to the repeated addition of P onto itself n times, for $n \in \mathbb{S}$ and $P \in \mathbb{E}$. We often write nP instead of $n \cdot P$, for brevity. Recalling the generators G from the previous paragraph, we can write $nG = \mathcal{O}$ for every generator $G \in \mathbb{E}$. Every point $P \in \mathbb{E}$ is the result of repeated additions of a generator G onto itself i times: $P = iG$ for some $i \in \mathbb{S}$. We call i the *discrete logarithm* of P with respect to G, which we may also write as $P/G = i$. *(This is division, but colloquially the operation is known as logarithm.)* Scalar multiplication is homomorphic: $(a+b)P = (a+b)(iG) = (ai+bi)G = aP + bP$, for all $a, b \in \mathbb{S}$ and all $P \in \mathbb{E}$. In other words, scalar multiplication *distributes* over point addition. In equations $aP = bP$, where $P \in \mathbb{E} \setminus \{\mathcal{O}\}$ and $a, b \in \mathbb{S}$, we may divide both sides by P because P is a generator. Both aP and bP are points on P's cycle through $\mathbb{E}$ which has no repetitions. Therefore, for $aP = bP$ to hold, $a = b$ must also hold.

There is no algorithm to efficiently compute the discrete logarithm of a random point. A brute-force algorithm has to check all scalars $0 \leq i < n$ (in case of secp256k1, $n \approx 2^{256}$). This is where the cryptographic security of elliptic curves comes from. Known algorithms such as Baby-step giant-step and Pollard's rho algorithm compute the discrete log in time $\approx \mathcal{O}(\sqrt{n})$ which is much faster but still not polynomial.

2 Commitments

In two-party protocols it can be useful to prove to the counterparty that a value used later (at time t_2) in the protocol has been fixed/defined at an earlier point in time (at $t_1 < t_2$) without having to reveal the concrete value at t_1. The technique used to prove this is called *commitments*, since they allow you to *commit* to a value at t_1 but to reveal it at t_2.

Alice wants to start using a given value, but she doesn't want to share this value with Bob yet. Instead, she gives Bob a handle that corresponds to her value. The handle makes it impossible that Alice changes her value over time without Bob noticing. Later, Alice can reveal the value to Bob and prove that her handle indeed corresponded to this value all along. This handle is called a commitment. Creating a handle for a value is called to *make* a commitment to a value. Proving that the handle corresponds to a value is called to *open* the commitment to the value.

2.1 Properties

Commitment schemes must satisfy two properties to be usable:

Hiding: Commitments do not reveal the value to which they were made.

Binding: Commitments cannot be opened to a different value than the one they were made to.

A scheme is *statistically (perfectly) hiding* if even a computationally unbounded adversary cannot extract information about a commitment's value with non-negligible probability. Likewise, a scheme is *statistically (perfectly) binding* if even such an adversary cannot open a commitment to a different value with non-negligible probability. Statistical binding excludes statistical hiding: A computationally unbounded adversary can simultaneously open a commitment to many messages. Statistical binding guarantees that this succeeds only for one value with non-negligible probability. This reveals the commitment's value. Therefore, the scheme is not statistically hiding.

A scheme is *computationally hiding* or *binding* if the adversary is restricted to polynomial-time algorithms. The schemes we will discuss in this document are all statistically hiding and computationally binding.

2.2 Pedersen Commitment

A Pedersen commitment to a value $a \in \mathbb{S}$ looks as follows:

$$C(r, a) = rH + aG$$

where $r \in \mathbb{S}$ is a random value (blinding factor), $H \in \mathbb{E}$ is a random point and $G \in \mathbb{E}$ is a generator. We write C instead of $C(r, a)$ if r and a are not important.

We could set $C = aG$, but then commitments to the same value a look identical. This would not be a hiding commitment. Instead, we add a random point rH onto aG to hide our value. Every time we make a commitment, we pick a random blinding factor r. The point H is constant across all commitments. It is important that no one knows the discrete logarithm of H. In other words, this point must be NUMS. If it weren't, then the commitment would not be binding.

2.2.1 Nothing up My Sleeve

Say Alice knew the DLog $H/G = i$ of H. Then she could make a commitment $C(r, a)$ and later open it to a different value $b \neq a$. She does so by constructing commitment $D(s, b) = C$ where $s = r + a - b$.

$$C(r, a) = rH + aG = (h + r + a)G = (h + s + b)G = sH + bG = D(s, b)$$

We avoid this catastrophe by constructing points that *no one* knows the DLog of. Take any nontrivial input, like the binary representation of generator G, and hash it with a cryptographically secure hash function. Interpret the resulting bytes as x coordinate of a curve point (and calculate the corresponding y coordinate). Because the hash function is secure, this gives you a pseudorandom point on the curve. We don't know the DLog of this point because computing it is infeasible. In fact, no one knows the DLog; the point was constructed using entirely different means than point addition. The point is genuinely random with no strings attached, so we call it NUMS. By trying different inputs, we can generate as many NUMS points as we want.

2.2.2 Hiding Property

For every commitment $C(r, a)$ and every $b \in \mathbb{S}$, there exists some $s \in \mathbb{S}$ such that $C(r, a) = D(s, b)$. In other words, if we set the blinding factors right, then we can open a commitment that was made to value $a \in \mathbb{Z}_p$ to any value $b \in \mathbb{Z}_p$. Commitments are sums which turn out to be a kind of one-way function: A sum can be decomposed into two summands in many ways. The larger the sum, the more combinations.

Switching values is infeasible in practice *(due to the DLog, see Binding)*, but we cannot detect it if it happened, even if we had unbounded computational resources. Pedersen commitments are *statistically hiding* (like the one-time pad).

2.2.3 Binding Property

If there are two commitments C and D that are equal, then this *proves* that both the blinding factors *and* and the values of C and D are equal.

$$C = rH + aG = sH + bG = D \text{ implies } r = s \text{ and } a = b$$

First, let's show that if the blinding factors are different then the values must *necessarily* be different, and vice versa. Point H is distinct from G, so there is some nonzero integer $x = P/G \neq 0$. We don't *know* x, but it *exists.* Given this knowledge, we transform the above equation as follows:

$$\begin{aligned}
rH + aG &= sH + bG \\
rxG + aG &= sxG + bG \\
(rx + a) \cdot G &= (sx + b) \cdot G \\
rx + a &= sx + b \\
(r - s)x &= b - a
\end{aligned}$$

The final equation holds if both sides are either zero or nonzero. We know that $x \neq 0$. The term $r - s$ is zero iff $r = s$ because each scalar has a unique additive inverse. Similarly, term $b - a$ is zero iff $b = a$. For the equation to hold, either blinding factors *and* values are equal, or they are all different.

Second, let's see what happens in case $r \neq s$ and $a \neq b$. Again, we transform the above equation:

$$\begin{aligned}
rH + aG &= sH + bG \\
aG - bG &= sH - rH \\
(a - b) \cdot G &= (s - r) \cdot H \\
(a - b) \cdot (s - r)^{-1} \cdot G &= H
\end{aligned}$$

The term $(a - b) \cdot (s - r)^{-1}$ efficiently computes the discrete logarithm of H with respect to G. Therefore, if Alice can open a commitment to a different value than the original, then she can efficiently solve the discrete logarithm of H, which is a random point. This violates our assumption that the DLog is hard, so this can never happen. Pedersen commitments are binding. $\square$

2.2.4 Homomorphism

The sum of two Pedersen commitments to values a_1 and a_2 is equal to the Pedersen commitment of $a_1 + a_2$, as long as the blinding factor of the last commitment is the sum of the blinding factors of the first commitments.

$$\begin{aligned}
C(r, a) + C(s, b) &= rH + aG + sH + bG \\
&= (r + s)H + (a + b)G \\
&= C(r + s, a + b)
\end{aligned}$$

2.3 Vector Pedersen Commitment

A Pedersen commitment to a vector $\mathbf{a} \in \mathbb{S}^k$ looks as follows:

$$C(r, \mathbf{a}) = rH + (a_1 G_1 + \ldots + a_k G_k) = rH + \mathbf{aG}$$

where $r \in \mathbb{S}$ is a random blinding factor, $H \in \mathbb{E}$ is some NUMS point and $G_1, \ldots, G_k \in \mathbb{E}$ are pairwise distinct generators. Again, we write just C for brevity.

The commitment is hiding thanks to rH. The commitment is binding because we would have to compute the DLog of vector component a_i with respect to generator G_i to change a_i, which is infeasible. This is why we have distinct generators for each component: If we had one generator, then the commitment would devolve into $rH(a_1 + \ldots + a_k)G$. We could change the order of components, subtract components from each other and so on.

2.3.1 Homomorphism

$$\begin{aligned} C(r, \mathbf{a}) + C(s, \mathbf{b}) &= rH + \mathbf{aG} + sH + \mathbf{bG} \\ &= (r + s)H + (\mathbf{a} \oplus \mathbf{b})\mathbf{G} \\ &= C(r + s, \mathbf{a} \oplus \mathbf{b}) \end{aligned}$$

2.3.2 Multiple Vectors

We can construct Pedersen commitments to multiple vectors by adding more pairwise distinct generators:

$$C(r, \mathbf{a}, \mathbf{b}, \ldots) = rH + \mathbf{aG} + \mathbf{bH} + \ldots$$

Multi-vector commitments are equivalent to one large vector commitment that encompasses all their dimensions.

2.3.3 Compression

Pedersen commitments are a single curve point, which consists of two coordinates. Many vectors of many components are compressed into this single point, so there is obviously information loss. This is why the commitments we are using for Bulletproofs can never be perfectly binding.

3 Interactive Proofs

An *interactive proof* (IP) consists of a *prover* that tries to convince a *verifier* of a given statement. More concretely, there is a function $f : \{0,1\}^n \rightarrow \mathcal{R}$, for some fixed $n \geq 0$ and some finite range $\mathcal{R}$. The prover tries to convince the verifier that $y = f(x)$ holds for some value y (ideally without the verifier having to repeat the entire computation).

The IP is realized in an *interactive proof system.* At the start of the protocol, f and x are *public*, i.e., known to all parties. Then, the prover sends y to the verifier which it claims satisfies $y = f(x)$. What follows is an exchange of messages between prover and verifier. Finally, the verifier either *accepts* (convinced) or *rejects* (not convinced) that $y = f(x)$ holds. The sequence of messages is called the *transcript* of the exchange.

Prover and verifier are modelled as Turing machines: The prover is a *deterministic* algorithm of *arbitrary* complexity, which we denote by $\mathcal{P}$. This means that $\mathcal{P}$ has Turing-complete power (and potentially breaks our cryptography) if not otherwise specified. Its messages depend on x, y, f and the previous messages. The verifier on the other hand is a *probabilistic* algorithm, denoted by $\mathcal{V}$, that runs in polynomial time with respect to n. Probabilistic algorithms are easier modelled as determinstic algorithms that take a random seed as input. $\mathcal{V}$'s messages depend on x, y, f, its random seed and the previous messages.

3.1 Completeness

An IP system is *complete* if the prover (almost) always manages to convince the verifier of true statements $y = f(x)$. Formally, for every $y \in \mathcal{R}$ such that $y = f(x)$ holds, $\mathcal{V}$ accepts with high probability. ($\mathcal{V}$'s random seed is assumed to be uniformly random.) If $\mathcal{V}$ accepts with probability 1, then the system is *perfectly complete.* We can construct a perfectly complete system out of any system with acceptance probability > 0.5. In this document, we exclusively focus on perfect completeness.

3.2 Statistical Soundness

An IP system is *statistically sound* if the verifier (almost) always rejects false statements $y = f(x)$, considering all possible ways a prover might try to deceive the verifier. Formally, for every $y \in \mathcal{R}$ such that $y \neq f(x)$ holds, and for every *prover strategy* $\mathcal{P}'$, verifier $\mathcal{V}$ rejects with high probability. (Again, $\mathcal{V}$'s random seed is uniformly random.) Prover strategy means that $\mathcal{P}'$ may violate the protocol to increase its chances to convince $\mathcal{V}$ of the false statement. In other words, $\mathcal{P}'$ uses all computational resources it has to break the scheme. When we prove soundness, we *cannot* rely on the equations on the prover's side to hold, as those might be violated. We can only rely on the verifier's side. The rejection probability is common set to $1 - \frac{1}{|\mathbb{F}|}$ for IP systems that are defined over a field $\mathbb{F}$.

3.3 Computational Soundness

Statistical soundness has the problem that $\mathcal{P}'$ is Turing-complete, so it can break cryptographic primitives like digital signatures and encryption. *Computational soundness* on the other hand considers only *polynomial* prover strategies. An IP system where the prover is assumed to run in polynomial time is called an *argument system*, and its proofs are called *arguments*. In this document, we repeatedly use cryptography and thus make use of computational soundness.

3.4 Proofs for Languages

We can model an IP so that the prover tries to convince the verifier that a given word is in a given language. This is useful for our intuition and for defining complexity classes that are solvable by IP systems. Formally, there is a language $\mathcal{L} \subseteq \{0,1\}^n$ for some fixed $n \geq 0$. We set function $f : \{0,1\}^n \to \{0,1\}$ such that $f(x) = 1$ iff $x \in \mathcal{L}$ holds. In this case, x is a word over $\{0,1\}$ and f encodes membership in $\mathcal{L}$. An IP for a function $y = f(x)$ is equivalent to an IP for the language $\{(x,y) \mid y = f(x)\}$ that encodes f as a relation. Likewise, an IP for a language is defined as an IP for a function with range $\{0,1\}$.

3.5 Proofs of Knowledge

For every true statement $y = f(x)$ there is a *witness* that proves it. *Computing* the witness is hard, but once we have it, *verifying* it is easy. This is the fundamental difference between computation and verification. Since f is a computable function, there exists a *decidable* (i.e. simple) function g such that $y = f(x)$ holds iff $y = g(x,w)$ holds for some witness w.

In an extended IP, the prover tries to convince the verifier of a statement $y = g(x,w)$ without revealing w. In other words, the prover tries to convince the verifier that it *knows* a witness w such that $y = g(x,w)$ holds.

IPs of knowledge over functions are equivalent to IPs of knowledge over languages. In general, g is much easier to compute than f. If f encodes an NP-complete language then g is solvable in PTime.

3.6 Knowledge Soundness

The existence of a witness w such that $y = g(x, w)$ holds is trivial for many functions. A discrete log *exists* for *each* point on an elliptic curve, but *computing* the discrete log for *any* point is infeasible. We need to revise our soundness notion: An IP system is *knowledge-sound* if the verifier (almost) always rejects statements $y = f(x)$ if the prover doesn't *know* a witness w such that $g(x, w) = f(x) = y$ holds.

3.6.1 Computational Epistemology

What does it mean for an algorithm to "know" something? This seemingly harmless question underlies the definition of knowledge soundness and has far-reaching consequences for our work. We proceed by defining words like "know" and "learn" formally and build some intuition based on that. We have to understand that algorithms are not people, so we cannot apply human standards to them. We have to unlearn what it means to "know" and to build back up from first principles that make sense for algorithms.

Here is the mental model: An algorithm "knows" what it can compute in polynomial time. It obviously knows its initial parameters and everything it receives from the outside world, but it can also do computations based on known values to arrive at new ones that it also "knows". An algorithm "learns" a value if it receives it from the outside world or if it arrives at the value in a computation based on a value that it previously learned. "Learning" means that the algorithm gets to know a value that it previously did not know.

Importantly, an algorithm cannot do exponential computations to learn new values. It cannot compute the discrete logarithm of an elliptic curve point, it cannot produce the solution to a sudoku, etc. The exponential barrier is useful for our epistemology because polynomial computations are seen as feasible while exponential ones are seen as infeasible. It is fine to apply reasonable effort to learn something new. If I apply *un*reasonable effort, then that is cheating! If the prover knows the solution to a sudoku that it handcrafted in many laborious (exponential) hours, the verifier should not be able to steal it just like that. Cryptography builds on one-way functions that are easy (polynomial) to compute on way but hard (exponential) to compute the other way. The verifier should not be able to break any of the prover's cryptographic primitives and the prover should not be able to manufacture broken primitives that try to convince the verifier of false statements.

3.6.2 Extractor

How do we prove that $\mathcal{P}$ knows the required witness? We can do a thought experiment based on our newly gained epistemology: If we can make $\mathcal{P}$ spit out the witness in polynomial time, then it must have known it! Remember that the witness is the result of an exponential computation based on the public problem description. We start interacting with the prover, not knowing the witness, and we cannot learn it independently because of the polynomial time constraint. Only because of our *interaction* with the prover do we manage to learn the witness in polynomial time. Where did the witness come from? It could not come from us because we don't know it. It must have come from the prover how *knew* it!

Formally, there is a probabilistic algorithm, called the *extractor* and denoted by $\mathcal{E}$, that extracts the witness w from any prover *strategy* $\mathcal{P}'$ in polynomial time. Prover strategy, again, means that the prover can try to cheat and still we manage to extract. An IP system is knowledge-sound if there exists such an extraction algorithm.

Constructing an extractor involves making copies of $\mathcal{P}$ and running the protocol many times for strategically-chosen values on the verifier's side. Turing machines can be started, stopped and replicated. The extractor takes the role of verifier. Effectively, the extractor is *rewinding* the prover to a previous state in the protocol, having learned something from the prover's output so far. What is really happening is that the extractor makes a copy of the state of $\mathcal{P}$, runs $\mathcal{P}$ for some more steps, and then "rewinds" by using the copy. Nevertheless, rewinding is a useful abstraction that hides messy low-level details.

Safe to say, this kind of interaction would not be allowed in a real setting. No real prover would let a third party manipulate itself like that to extract the precious secrets that the prover holds. Provers would reveal the secret witness to every verifier that they interact with. But the above is a thought experiment. Precisely *because* it could happen (if the provers are not careful), they must know the witness.

3.7 Zero Knowledge

In an IP of knowledge the prover tries to convince the verifier that it knows a witness that satisfies some property. The prover does so without revealing the witness to the verifier. *Zero-knowledge* (ZK) means that the verifier learns nothing from its exchange with prover, besides that the statement which the prover means to prove is true. The verifier cannot use what it learned to convince a third party of the statement, since the verifier did not learn *why* the statement is true, just that it *is* true. The transcript of the exchange does not constitute a proof. A third party that eavesdrops on the conversation will not be convinced of the proof. What convinced the verifier is precisely its *interaction* with the prover: The prover has to meet the challenges from the verifier *in order* and those challenges are *random*. If the prover manages to meet those challenges then that implies that the prover was ready to meet a much larger set of challenges, enough to convince the verifier.

3.7.1 Simulator

How do we prove that the verifier doesn't learn anything? Again, we do a thought experiment: If the transcripts are so random-looking that the verifier could generate them itself, then there is no knowledge in the transcripts (that the verifier doesn't already know)! Information theory tells us that it does not matter when the verifier starts extracting, so we assume it starts at the end of the exchange when the transcript is finished. If there is anything of value that the verifier could learn, then it should not already know it (polynomial computations). If we can show that the verifier *can* always generate the same transcripts via a polynomial algorithm, then it cannot learn anything. Everything the verifier could extract from the transcripts it already knows, as by our epistemology.

Formally, an IP system is *zero-knowledge* if there there is a probabilistic algorithm, called the *simulator* and denoted by $\mathcal{S}$, that generates a random distribution of transcripts, which is indistinguishable from the distribution generated by $\mathcal{P}$ and $\mathcal{V}$, in polynomial time. The word "indistinguishable" is formally defined below (see Section 3.7.2).

Simulators commonly reverse the order of messages, pick random values that are not originally random, and calculate other values based on equations even though those values are originally random. What matters is the transcript alone and not how it was generated. The generated transcripts should be valid and have the same distribution as the original ones.

Again, this is a setting that would not occur in reality. Verifiers only care about their *interaction* with the prover, not about the transcript. Verifiers want to *choose* their random challenges. The simulator chooses its own challenges. No verifier would be fooled by a simulator. This is why an IP system can be sound and zero-knowledge at the same time.

3.7.2 ZK in Many Flavors

There are different definitions of indistinguishability of transcripts. The verifier may also violate the protocol to increase its chances at knowledge extraction. The different notions form natural hierarchies. We will focus on Perfect Honest-Verifier ZK in this document.

Perfect ZK: The transcripts generated by $\mathcal{S}$ and those generated by $\mathcal{P}$ and $\mathcal{V}$ are exactly the same distribution.

Statistical ZK (SZK): The transcripts generated by $\mathcal{S}$ and those generated by $\mathcal{P}$ and $\mathcal{V}$ have "negligible statistical distance". This means that the distributions cannot be distinguished by any algorithm with non-negligible probability given a polynomial number of samples.

Computational ZK (CZK): The transcripts generated by $\mathcal{S}$ and those generated by $\mathcal{P}$ and $\mathcal{V}$ cannot be distinguished with non-negligible probability by a polynomial-time algorithm given a polynomial number of samples.

Honest-Verifier ZK (HVZK): The verifier $\mathcal{V}$ behaves as prescribed by the protocol.

Dishonest-Verifier ZK: The verifier $\mathcal{V}$ is any probabilistic polynomial-time algorithm (verifier strategy $\mathcal{V}'$). A simulator for a dishonest verifier must know $\mathcal{V}'$ and interact to its challenges. The simulator takes the role of prover. This is more complicated than Honest-Verifier ZK and won't be covered in this document.

Plain ZK: The verifier $\mathcal{V}$ has inputs x, y, f. This means that every run of the protocol is independent.

Auxiliary-Input ZK (AIZK): The verifier $\mathcal{V}$ has an *auxiliary input* z in addition to the above, which is known to $\mathcal{V}$ and $\mathcal{S}$ but not to $\mathcal{P}$. The verifier can use z to learn information across multiple runs of the protocol. The *composition* of AIZK protocols in a larger protocol is itself AIZK. Honest verifiers don't make use of auxiliary inputs, so plain HVZK equals auxiliary-input HVZK.

4 Knowledge of Scalars

4.1 Knowledge of a Single Scalar

We can prove knowledge of a single scalar $x \in \mathbb{S}$ as follows: A point P is publicly known; this is called the *public key.* $\mathcal{P}$ tries to convince $\mathcal{V}$ that it knows x such that $xG = P$; this is called the private or *secret key.* In other words, $\mathcal{P}$ knows the discrete logarithm of P.

1. $\mathcal{P}$:
    - $k \leftarrow_{\$} \mathbb{S}$
    - $R = kG \in \mathbb{E}$
2. $\mathcal{P} \rightarrow \mathcal{V} : R$
3. $\mathcal{V} : e \leftarrow_{\$} \mathbb{S}$
4. $\mathcal{P} \leftarrow \mathcal{V} : e$
5. $\mathcal{P} : s = k + ex \in \mathbb{S}$
6. $\mathcal{P} \rightarrow \mathcal{V} : s$
7. $\mathcal{V} : sG \overset{?}{=} R + eP$

4.1.1 Perfect Completeness

$\mathcal{P}$ *always* manages to convince $\mathcal{V}$ because the following equalities hold:

$$R + eP = kG + exG = sG$$

4.1.2 Knowledge Soundness

This is also called *special soundness*, because Schnorr is a Σ-protocol. We construct $\mathcal{E}$ as follows:

- Run $\mathcal{P}$ until it sends R and make a copy
- Run the copy 2 times for different values $e \in \mathbb{S}$
- The transcripts are (R, e, s) and (R, e', s') where $e \neq e'$
- Compute $x = \frac{s-s'}{e-e'}$

$$\begin{aligned}
R + eP = kG + exG = sG &\implies k + ex = s \\
R + e'P = kG + e'xG = s'G &\implies k + e'x = s' \\
&\iff s' - e'x + ex = s \\
&\iff (e - e')x = s - s' \\
&\iff x = \frac{e - e'}{s - s'}
\end{aligned}$$

This algorithm runs in polynomial time and finds secret key x with probability 1.

4.1.3 Honest-Verifier Perfect ZK

We construct $\mathcal{S}$ for an honest verifier as follows:

- Choose $e, s \in \mathbb{S}$ randomly
- Set $R = sG - eP$

The simulated transcripts are valid:

$$R + eP = sG - eP + eP = (k + ex)G = sG$$

The simulated transcripts have *exactly* the same distribution as the original transcripts. The latter is as follows. *(The probabilities mean that the variables take a given value from their domain.)*

$$\Pr[R] = \frac{1}{n} \quad \Pr[e] = \frac{1}{n} \quad \Pr[R, e] = \frac{1}{n^2}$$
$$\Pr[s \mid R, e] = 1 \implies \Pr[R, e, s] = \frac{1}{n^2}$$

The simulated transcripts are as follows:

$$\Pr[e] = \frac{1}{n} \quad \Pr[s] = \frac{1}{n} \quad \Pr[e, s] = \frac{1}{n^2}$$
$$\Pr[R \mid e, s] = 1 \implies \Pr[R, e, s] = \frac{1}{n^2}$$

As you can see, the independent probability $\Pr[R, e, s]$ of the final transcript is the same for both distributions. We make use of the fact $\Pr[A, B] = \Pr[A \mid B] \cdot \Pr[B]$.

4.2 Knowledge of Multiple Scalars

We prove knowledge of a set of vectors as follows: A vector $\mathbf{C}$ of vector Pedersen commitments is publicly known, defined as below. $\mathcal{P}$ tries to convince $\mathcal{V}$ that it knows the openings $\{(r_i, \mathbf{x}_i) \mid 1 \leq i \leq m\}$ to these commitments.

$$\mathbf{C} = C_1, \ldots, C_m \text{ where } C_i = r_i H + \mathbf{x}_i \mathbf{G} \text{ and } \mathbf{x}_i \in (\mathbb{S})^k \text{ for } 1 \leq i \leq m$$

1. $\mathcal{P}$:
   - $r_0 \leftarrow\!\$\, \mathbb{S} \quad \mathbf{x}_0 \leftarrow\!\$\, \mathbb{S}^k$
   - $C_0 = r_0 H + \mathbf{x}_0 \mathbf{G} \in \mathbb{E}$

2. $\mathcal{P} \rightarrow \mathcal{V} : C_0$

3. $\mathcal{V} : e \leftarrow_{\$} \mathbb{S}$

4. $\mathcal{P} \leftarrow \mathcal{V} : e$

5. $\mathcal{P}$:
   - $\mathbf{z} = \sum_{i=0}^{m} e^i \mathbf{x}_i \in \mathbb{S}^k$
   - $s = \sum_{i=0}^{m} e^i r_i \in \mathbb{S}$

6. $\mathcal{P} \rightarrow \mathcal{V} : \mathbf{z}, s$

7. $\mathcal{V} : \sum_{i=0}^{m} e^i C_i \stackrel{?}{=} sH + \mathbf{z}\mathbf{G}$

4.2.1 Perfect Completeness

$\mathcal{P}$ always manages to convince $\mathcal{V}$ because of the following equalities:

$$sH + \mathbf{z}\mathbf{G} = \sum_{i=0}^{m} e^i r_i H + \sum_{i=0}^{m} e^i \mathbf{x}_i \mathbf{G} = \sum_{i=0}^{m} e^i (r_i H + \mathbf{x}_i \mathbf{G}) = \sum_{i=0}^{m} e^i C_i$$

4.2.2 Knowledge Soundness

We construct $\mathcal{E}$ as follows:

- Run $\mathcal{P}$ until it sends C_0 and make a copy
- Run the copy $m+1$ times for pairwise distinct values $e \in \mathbb{S}$
- The transcripts are $(C_0, e_j, (\mathbf{z}_j, s_j))$ for $0 \leq j \leq k$
- For valid transcripts, we have $\mathbf{z} = \sum_{i=0}^{m} e^i \mathbf{x}_i$ (see below)
- Every component $0 \leq j < k$ of $\mathbf{z}$ is the evaluation $z_j = \sum_{i=0}^{m} e^i (x_i)_j$ of a polynomial of degree m where e is the "x-value" and $\{(x_i)_j \mid 0 \leq i \leq m\}$ are the "coefficients"
- Using $m+1$ transcripts, we can interpolate the polynomials z_j using the so-called Vandermonde matrix (see below)
- The interpolation yields $\mathbf{x}_i$ for $0 \leq i \leq m$ (analogously, s can be interpolated to yield the blinding factors)

We start by establishing the relationship between $\mathbf{x}$, $\mathbf{z}$ and e. This is given on the Prover's side, but we don't know if the Prover is honest! The following lines hold with high probability because Pedersen commitments are binding.

$$\sum_{i=0}^{m} e^i C_i = sH + \mathbf{z}\mathbf{G}$$

$$\sum_{i=0}^{m} e^i (r_i H + \mathbf{x}_i \mathbf{G}) = sH + \mathbf{z}\mathbf{G}$$

$$\sum_{i=0}^{m} e^i \mathbf{x}_i = \mathbf{z}$$

The Vandermonde matrix is a tool to evaluate polynomials of degree m: Given this matrix V that contains the x-values and a vector A that contains the coefficients, we can compute a vector Y of $m+1$ many y-values by matrix multiplication $VA = Y$. We can invert V to compute A from Y by $A = V^{-1}Y$. V is invertible if its determinant is zero which in turn is the case if the x-values in V are pairwise distinct. All this is to say that we can interpolate a polynomial of degree m given $m+1$ evaluations at pairwise distinct x-values. The resulting polynomial is unique. Watch this YouTube video for an in-depth explanation.

Note that the interpolation works only if C_0 is fixed. C_0 blinds both $\mathbf{z}$ and s. Because $\mathcal{P}$ randomly chooses C_0 at the beginning, the polynomials that are to be interpolated keep changing. Interpolation works only because $\mathcal{E}$ fixes C_0 at the beginning.

4.2.3 Honest-Verifier Perfect ZK

We construct $\mathcal{S}$ for an honest verifier as follows:

- Choose $e, s \in \mathbb{S}$ and $\mathbf{z} \in (\mathbb{S})^k$ randomly
- Set $C_0 = (sH + \mathbf{z}\mathbf{G}) - \sum_{i=0}^{m} e^i C_i$

The simulated transcripts are valid:

$$sH + \mathbf{z}\mathbf{G} = C_0 + \sum_{i=1}^{m} e^i C_i = \sum_{i=0}^{m} e^i C_i$$

The simulated transcripts have the same distribution as the original transcripts. The latter is as follows:

$$\Pr[r_0] = \frac{1}{n} \quad \Pr[\mathbf{x}_0] = \frac{1}{n^k} \quad \Pr[C_0] = \frac{1}{n} \quad \Pr[e] = \frac{1}{n} \quad \Pr[C_0, e] = \frac{1}{n^2}$$

$$\Pr[\mathbf{z}, s \mid C_0, e] = \frac{1}{n^k} \implies \Pr[C_0, e, \mathbf{z}, s] = \frac{1}{n^{k+2}}$$

It is important to understand the conditional probability $\Pr[\mathbf{z}, s \mid C_0, e]$. $\mathbf{z}$ depends on $\mathbf{x}_0$, and s depends on r_0. For any given value of C_0, there are n^k combinations of $\mathbf{x}_0$ and r_0. Imagine n^k random vectors $\mathbf{x}_0$ and a fixed blinding factor r_0 for each. $\mathbf{z}$ and s depend on different values, but they *jointly* depend on C_0. The simulated transcripts are as follows:

$$\Pr[e] = \frac{1}{n} \quad \Pr[\mathbf{z}] = \frac{1}{n^k} \quad \Pr[s] = \frac{1}{n} \quad \Pr[e, \mathbf{z}, s] = \frac{1}{n^{k+2}}$$

$$\Pr[C_0 \mid e, \mathbf{z}, s] = 1 \implies \Pr[C_0, e, \mathbf{z}, s] = \frac{1}{n^{k+2}}$$

As you can see, the independent probability $\Pr[C_0, e, \mathbf{z}, s]$ of the final transcript is the same for both distributions.

5 Knowledge of Inner Products

Inner products are useful because they can encode NP-complete languages. You can prove that a Pedersen commitment opens to a value inside a range *(one proves knowledge of such an opening; because of binding, this is the only opening with high probability).* This is called a *range proof. Confidential transactions* have amounts hidden in Pedersen commitments and require range proofs to prevent overflows. You can prove knowledge of a *SHA256 preimage.* You can prove knowledge of an input that makes an *arithmetic circuit* return a given output. Finally, you can prove knowledge of solution to a *Sudoku puzzle* and sell it for Bitcoin (ZKCP), among many other things.

5.1 Linear Protocol

The first protocol we consider proves knowledge of an inner product, but its transcripts are rather large (linear in the vector dimension). Vectors $\mathbf{x}, \mathbf{y} \in \mathbb{S}^k$ are publicly known where $k \geq 0$. $\mathcal{P}$ tries to convince $\mathcal{V}$ that it knows some scalar $z \in \mathbb{S}$ that is the inner product $\mathbf{x} \cdot \mathbf{y} = z$ of $\mathbf{x}$ and $\mathbf{y}$. *In Euclidean space, the inner product is the dot product.* For zero-knowledge, we rephrase the above in terms of Pedersen commitments:

$$C_x = r_x H + \mathbf{x}\mathbf{G} \quad C_y = r_y H + \mathbf{y}\mathbf{G} \quad C_z = r_z H + zG$$

$\mathcal{P}$ claims that it knows openings $(r_x, \mathbf{x})$, $(r_y, \mathbf{y})$ and (r_z, z) to these commitments such that $\mathbf{x} \cdot \mathbf{y} = z$ holds.

1. $\mathcal{P}$:
   - $s_x, s_y, s_1, s_0 \leftarrow\!\$\, \mathbb{S} \quad \mathbf{d}_x, \mathbf{d}_y \leftarrow\!\$\, \mathbb{S}^k$
   - $D_x = s_x + \mathbf{d}_x \mathbf{G} \in \mathbb{E}$
   - $D_y = s_y + \mathbf{d}_y \mathbf{G} \in \mathbb{E}$
   - $D_1 = s_1 H + (\mathbf{x} \cdot \mathbf{d}_y + \mathbf{y} \cdot \mathbf{d}_x) G \in \mathbb{E}$
   - $D_0 = s_0 H + (\mathbf{d}_x \cdot \mathbf{d}_y) G \in \mathbb{E}$
2. $\mathcal{P} \rightarrow \mathcal{V} : D_x, D_y, D_1, D_0$
3. $\mathcal{V} : e \leftarrow\!\$\, \mathbb{S}$
4. $\mathcal{P} \leftarrow \mathcal{V} : e$
5. $\mathcal{P}$:
   - $\mathbf{f}_x = e\mathbf{x} + \mathbf{d}_x \in \mathbb{S}^k \quad \mathbf{f}_y = e\mathbf{y} + \mathbf{d}_y \in \mathbb{S}^k$
   - $t_x = e r_x + s_x \in \mathbb{S} \quad t_y = e r_y + s_y \in \mathbb{S}$
   - $t_z = e^2 r_z + e s_1 + s_0 \in \mathbb{S}$
6. $\mathcal{P} \rightarrow \mathcal{V} : \mathbf{f}_x, \mathbf{f}_y, t_x, t_y, t_z$
7. $\mathcal{V}$:
   - $eC_x + D_x \overset{?}{=} t_x H + \mathbf{f}_x \mathbf{G}$
   - $eC_y + D_y \overset{?}{=} t_y H + \mathbf{f}_y \mathbf{G}$
   - $t_z H + (\mathbf{f}_x \cdot \mathbf{f}_y) G \overset{?}{=} e^2 C_z + e D_1 + D_0$

5.1.1 Perfect Completeness

$\mathcal{P}$ always manages to convince $\mathcal{V}$ because of the following equalities:

$$\begin{aligned}
eC_x + D_x &= e(r_xH + \mathbf{x}\mathbf{G}) + (s_xH + \mathbf{d}_x\mathbf{G}) = (er_x + s_x)H + (e\mathbf{x} + \mathbf{d}_x)\mathbf{G} \\
&= t_xH + \mathbf{f}_x\mathbf{G} \\
eC_y + D_y &= e(r_yH + \mathbf{y}\mathbf{G}) + (s_yH + \mathbf{d}_y\mathbf{G}) = (er_y + s_y)H + (e\mathbf{y} + \mathbf{d}_y)\mathbf{G} \\
&= t_yH + \mathbf{f}_y\mathbf{G} \\
\mathbf{f}_x \cdot \mathbf{f}_y &= (e\mathbf{x} + \mathbf{d}_x) \cdot (e\mathbf{y} + \mathbf{d}_y) = e^2(\mathbf{x} \cdot \mathbf{y}) + e(\mathbf{x} \cdot \mathbf{d}_y) + e(\mathbf{y} \cdot \mathbf{d}_x) + \mathbf{d}_x \cdot \mathbf{d}_y \\
&= e^2(\mathbf{x} \cdot \mathbf{y}) + e(\mathbf{x} \cdot \mathbf{d}_y + \mathbf{y} \cdot \mathbf{d}_x) + \mathbf{d}_x \cdot \mathbf{d}_y \\
t_zH + (\mathbf{f}_x \cdot \mathbf{f}_y)G &= (e^2r_z + es_1 + s_0)H + (e^2(\mathbf{x} \cdot \mathbf{y}) + e(\mathbf{x} \cdot \mathbf{d}_y + \mathbf{y} \cdot \mathbf{d}_x) + \mathbf{d}_x \cdot \mathbf{d}_y)G \\
&= e^2(r_zH + zG) + e(s_1H + (\mathbf{x} \cdot \mathbf{d}_y + \mathbf{y} \cdot \mathbf{d}_x)G) + (s_0H + (\mathbf{d}_x \cdot \mathbf{d}_y)G) \\
&= e^2C_z + eD_1 + D_0
\end{aligned}$$

5.1.2 Knowledge Soundness

We construct $\mathcal{E}$ as follows:

- Run $\mathcal{P}$ until it sends its first message and make a copy
- Run the copy 3 times for different values $e \in \mathbb{S}$
- The transcripts are $((D_x, D_y, D_1, D_0), e, (\mathbf{f}_x, \mathbf{f}_y, t_x, t_y, t_z))$ and $(\dots, e', (\mathbf{f}'_x, \mathbf{f}'_y, t'_x, t'_y, t'_z))$
- Compute $\mathbf{x}, \mathbf{y}$ and finally z as below

$$\begin{aligned}
& eC_x + D_x = t_xH + \mathbf{f}_x\mathbf{G} \wedge e'C_x + D_x = t'_xH + \mathbf{f}'_x\mathbf{G} \\
\iff & (e - e')C_x = (t_x - t'_x)H + (\mathbf{f}_x - \mathbf{f}'_x)\mathbf{G} \qquad\qquad \mid \cdot\, \eta \coloneqq (e - e')^{-1} \\
\iff & C_x = \eta(t_x - t'_x)H + \eta(\mathbf{f}_x - \mathbf{f}'_x)\mathbf{G}
\end{aligned}$$

Because Pedersen commitments are binding, we can deduce $\mathbf{x} = \eta(\mathbf{f}_x - \mathbf{f}'_x)$ with high probability. Analogously, we get $\mathbf{y} = \eta(\mathbf{f}_y - \mathbf{f}'_y)$. Now we turn our attention to z which requires some more transformations. Let $(\alpha_1, \beta_1), (\alpha_0, \beta_0) \in \mathbb{S}^2$ be openings of D_1 and D_0, respectively. We can open C_z to (r_z, z) with high probability because of binding.

$$\begin{aligned}
t_zH + (\mathbf{f}_x \cdot \mathbf{f}_y)G &= e^2C_z + eD_1 + D_0 \\
&= e^2(r_zH + zG) + e(\alpha_1H + \beta_1G) + (\alpha_0H + \beta_0G) \\
&= H(e^2r_z + e\alpha_1 + \alpha_0) + G(e^2z + e\beta_1 + \beta_0)
\end{aligned}$$

Again, because Pedersen commitments are binding, we deduce $\mathbf{f}_x \cdot \mathbf{f}_y = e^2z + e\beta_1 + \beta_0$ with high probability. This is a polynomial of degree 2. Having three evaluations $\mathbf{f}_x \cdot \mathbf{f}_y$ of the polynomial at different x-values e, we can interpolate the coefficients z, β_1 and β_0.

5.1.3 Soundness

We want to show that if $\mathcal{P}$ manages to convince $\mathcal{V}$ with a valid transcript, then $\mathbf{x} \cdot \mathbf{y} = z$ actually holds.

Assume we have a valid transcript $((D_x, D_y, D_1, D_0), e, (\mathbf{f}_x, \mathbf{f}_y, t_x, t_y, t_z))$. We need to deconstruct $\mathbf{f}_x$, $\mathbf{f}_y$, D_1 and D_0 based on the equations on the Verifier's side. We start with $\mathbf{f}_x$: Let $(\alpha_x, \beta_x) \in \mathbb{S}^2$ be any opening of D_x.

$$\begin{aligned} eC_x + D_x &= t_x H + \mathbf{f}_x \mathbf{G} \\ e(r_x H + \mathbf{x}\mathbf{G}) + (\alpha_x H + \beta_x \mathbf{G}) &= t_x H + \mathbf{f}_x \mathbf{G} \\ e\mathbf{x} + \beta_x &= \mathbf{f}_x \end{aligned}$$

The last line holds because of binding. Similarly, we get $e\mathbf{y} + \beta_y = \mathbf{f}_y$ for any opening (α_y, β_y) of D_y. As for knowledge soundness *(see above)*, we have $\mathbf{f}_x \cdot \mathbf{f}_y = e^2 z + e\beta_1 + \beta_0$ for openings $(\alpha_1, \beta_1), (\alpha_0, \beta_0)$ of D_1 and D_0.

$$\begin{aligned} \mathbf{f}_x \cdot \mathbf{f}_y &= e^2 z + e\beta_1 + \beta_0 \\ (e\mathbf{x} + \beta_x) \cdot (e\mathbf{y} + \beta_y) &= e^2 z + e\beta_1 + \beta_0 \\ e^2(\mathbf{x} \cdot \mathbf{y}) + e(\mathbf{x} \cdot \beta_y + \mathbf{y} \cdot \beta_x) + \beta_x \cdot \beta_x &= e^2 z + e\beta_1 + \beta_0 \\ e^2(\mathbf{x} \cdot \mathbf{y} - z) + e(\ldots - \beta_1) + \beta_x \cdot \beta_x - \beta_0 = 0 & \end{aligned}$$

This is a polynomial $g(x)$ of degree $d = 2$ over $\mathbb{S}$. The Schwarz-Zippel lemma says that the probability of $g(x) = 0$ is at most $\frac{d}{n} \approx \frac{2}{2^{256}} \approx 0$ if g is a nonzero polynomial. We conclude that g is the zero polynomial with high probability and thus $\mathbf{x} \cdot \mathbf{y} = z$ holds.

5.1.4 Honest-Verifier Perfect ZK

We construct $\mathcal{S}$ for an honest verifier as follows:

- Choose $e, t_x, t_y, t_z, \alpha \in \mathbb{S}$ and $\mathbf{f}_x, \mathbf{f}_y \in \mathbb{S}^k$ randomly
- Set $D_1 = \alpha H + 0G$ *(zero times the generator)*
- Set $D_x = (t_x H + \mathbf{f}_x \mathbf{G}) - eC_x$
- Set $D_y = (t_y H + \mathbf{f}_y \mathbf{G}) - eC_y$
- Set $D_0 = t_z H + (\mathbf{f}_x \cdot \mathbf{f}_y)G - eD_1 - e^2 C_z$

The simulated transcripts are valid:

$$\begin{aligned}
eC_x + D_x &= eC_x + (t_x H + \mathbf{f}_x \mathbf{G}) - eC_x = t_x H + \mathbf{f}_x \mathbf{G} \\
eC_y + D_y &= eC_y + (t_y H + \mathbf{f}_y \mathbf{G}) - eC_y = t_y H + \mathbf{f}_y \mathbf{G} \\
t_z H + (\mathbf{f}_x \cdot \mathbf{f}_y) G &= D_0 - ((\mathbf{f}_x \cdot \mathbf{f}_y) G - eD_1 - e^2 C_z) + (\mathbf{f}_x \cdot \mathbf{f}_y) G \\
&= e^2 C_z + eD_1 + D_0
\end{aligned}$$

Here is the probability distribution of the original transcripts:

$$\begin{aligned}
&\Pr[D_x] = \Pr[D_y] = \Pr[D_1] = \Pr[D_0] = \Pr[e] = \frac{1}{n} \quad \Pr[D \ldots, e] = \left(\frac{1}{n}\right)^5 \\
&\Pr[\mathbf{f}_x, t_x \mid D \ldots, e] = \Pr[\mathbf{f}_y, t_y \mid D \ldots, e] = \left(\frac{1}{n}\right)^{2k} \quad \Pr[t_z \mid D \ldots, e] = 1 \\
&\Pr[\mathbf{f} \ldots, t \ldots \mid D \ldots, e] = \left(\frac{1}{n}\right)^{2k} \quad \Pr[D \ldots, e, \mathbf{f} \ldots, t \ldots] = \left(\frac{1}{n}\right)^{2k+5}
\end{aligned}$$

Again, the conditional probability $\Pr[\mathbf{f}_x, t_x \mid D \ldots, e]$ is the crux of the computation: $\mathbf{f}_x$ depends on e and $\mathbf{d}_x$, while t_x depends on e and s_x. Both *jointly* depend on D_x. There are $\frac{1}{n^k}$ combinations to form the same D_x; an arbitrary vector $\mathbf{d}_x \in \mathbb{S}^k$ and a matching blinding factor $s_x \in \mathbb{S}$. This is why we have to consider the probability of $\mathbf{f}_x$ *together* with t_x. There is a similar argument for $\mathbf{f}_y$ and t_y. Here is the distribution of the simulated transcripts:

$$\begin{aligned}
&\Pr[e, \mathbf{f} \ldots, t \ldots] = \left(\frac{1}{n}\right)^{2k+4} \quad \Pr[D_1] = \frac{1}{n} \\
&\Pr[D_x, D_y, D_0 \mid e, \mathbf{f} \ldots, t \ldots] = 1 \\
&\Pr[D \ldots \mid e, \mathbf{f} \ldots, t \ldots] = \frac{1}{n} \quad \Pr[D \ldots, e, \mathbf{f} \ldots, t \ldots] = \left(\frac{1}{n}\right)^{2k+5}
\end{aligned}$$

The independent probability $\Pr[D \ldots, e, \mathbf{f} \ldots, t \ldots]$ of the final transcript is the same for both distributions.

5.2 Logarithmic Protocol

Our goal is to compress the inner product proof. We do so by recursively splitting our vectors **a** and **b**. In each step, the size of **a** and **b** is divided by a common factor. Commitments to challenges ensure that the Prover cannot cheat. In the end, the Prover reveals a small vector that the Verifier verifies. Instead of sending large vectors, we send some curve points (two coordinates) and a small vector in the end. The total transcript is logarithmic (!) in the size of **a** and **b**.

Bootle's original protocol splits **a** and **b** by an arbitrary divisor of their length. This is fine, but some decompositions are better than others. In particular, we want to minimize the sum of the factors of a number. Prime factors uniquely fulfill this property. Among all prime numbers, two is the smallest. If the lengths of **a** and **b** is a power of two, then we can prove their inner product with the smallest possible transcript. Therefore and for simplicity, we restrict ourselves to binary splits in this document.

A few things to note. Here, Pedersen commitments are *not hiding*. This protocol has no zero-knowledge, but it is *not used directly*. *Other protocols* define a zero-knowledge inner product which they prove with this protocol. This is a repeating pattern for Bulletproofs. Therefore, the lack of ZK is no problem. We will first concentrate on **a** (in black font) and then generalize to **b** and z (in gray font). A recursive protocol for proving knowledge of a vector (**a**) is very close to one for proving an inner product, it turns out. There are few steps required to make the change.

5.2.1 Base Case

Vectors **a** and **b** have length $m = 2$. The Prover sends them in plain and the Verifier checks if they match the commitments A and B as well as z which the Verifier has computed in the previous step.

1. $\mathcal{P} \to \mathcal{V} : \mathbf{a}, \mathbf{b}$
2. $$\mathcal{V} : A \overset{?}{=} \textstyle\sum_{i=1}^{2} a_i G_i \quad B \overset{?}{=} \textstyle\sum_{i=1}^{2} b_i H_i \quad z \overset{?}{=} \mathbf{a} \cdot \mathbf{b}$$

5.2.2 Reduction

In each step of the recursion, we split **a** and **G** of dimension m into chunks (vectors) $\mathbf{a}_1, \mathbf{a}_2$ and $\mathbf{G}_1, \mathbf{G}_2$ of dimension $\frac{m}{2}$. Similarly for **b** and **H**. z is the sum of dot products over the chunks of **a** and **b**.

$$A = \mathbf{aG} = \sum_{i=1}^{m} a_i G_i = \sum_{i=1}^{2} \mathbf{a}_i \mathbf{G}_i$$

$$B = \mathbf{bH} = \sum_{i=1}^{m} b_i H_i = \sum_{i=1}^{2} \mathbf{b}_i \mathbf{H}_i \quad z = \mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{2} \mathbf{a}_i \cdot \mathbf{b}_i$$

Next, the Prover sends the diagonals A_k, B_k and z_k of the following matrices, respectively. There are three diagonals for a 2×2 matrix. The middle diagonal is exactly $\mathbf{a}$, $\mathbf{b}$ and z, respectively.

$$\begin{pmatrix} \mathbf{a}_1\mathbf{G}_1 & \mathbf{a}_2\mathbf{G}_1 \\ \mathbf{a}_2\mathbf{G}_2 & \mathbf{a}_2\mathbf{G}_2 \end{pmatrix} \quad \begin{pmatrix} \mathbf{b}_1\mathbf{H}_1 & \mathbf{b}_2\mathbf{H}_1 \\ \mathbf{b}_2\mathbf{H}_2 & \mathbf{b}_2\mathbf{H}_2 \end{pmatrix} \quad \begin{pmatrix} \mathbf{a}_1\mathbf{b}_1 & \mathbf{a}_2\mathbf{b}_1 \\ \mathbf{a}_2\mathbf{b}_2 & \mathbf{a}_2\mathbf{b}_2 \end{pmatrix}$$

Finally, after challenge x has been received from the Verifier, both sides reduce the problem $\mathbf{a} \cdot \mathbf{b} \overset{?}{=} z$ into $\mathbf{a}' \cdot \mathbf{b}' \overset{?}{=} z'$ where $\mathbf{a}'$ and $\mathbf{b}'$ have dimension $\frac{m}{2}$. Prover and Verifier compute new generators $\mathbf{G}'$ and a commitment $\mathbf{A}'$. The Prover computes a new vector $\mathbf{a}'$ such that $\mathbf{A}' = \mathbf{a}'\mathbf{G}'$. Similar for $\mathbf{H}'$, $\mathbf{B}'$ and $\mathbf{b}'$. Because z is public from the beginning, both sides can compute z' such that $\mathbf{a}' \cdot \mathbf{b}' = z'$ implies $\mathbf{a} \cdot \mathbf{b} = z$.

1. $\mathcal{P}, \mathcal{V} : m' = \frac{m}{2}$
2. $\mathcal{P}$:
   - $A_k = \sum_{i=\max(1,1-k)}^{\min(2,2-k)} \mathbf{a}_{i+k}\mathbf{G}_i \in \mathbb{E}$ for $k = -1, 0, 1$
   - $B_k = \sum_{i=\max(1,1-k)}^{\min(2,2-k)} \mathbf{b}_{i+k}\mathbf{H}_i \in \mathbb{E}$ for $k = -1, 0, 1$
   - $z_k = \sum_{i=\max(1,1-k)}^{\min(2,2-k)} \mathbf{a}_i \cdot \mathbf{b}_{i+k}$ for $k = -1, 0, 1$
3. $\mathcal{P} \rightarrow \mathcal{V} : \{A_k\}, \{B_k\}, \{z_k\}$ for $k = -1, 0, 1$
4. $\mathcal{V} : x \leftarrow_{\$} \mathbb{S}$
5. $\mathcal{P} \leftarrow \mathcal{V} : x$
6. $\mathcal{P}, \mathcal{V} :$
   - $A' = \sum_{k=-1}^{1} x^k A_k \in \mathbb{E} \quad \mathbf{G}' = \sum_{i=1}^{2} x^{-i}\mathbf{G}_i \in \mathbb{E}^{m'}$
   - $B' = \sum_{k=-1}^{1} x^{-k} B_k \in \mathbb{E} \quad \mathbf{H}' = \sum_{i=1}^{2} x^{i}\mathbf{H}_i \in \mathbb{E}^{m'}$
   - $z' = \sum_{k=-1}^{1} x^{-k} z_k \in \mathbb{S}$
7. $\mathcal{P} : \mathbf{a}' = \sum_{i=1}^{2} x^i \mathbf{a}_i \in \mathbb{S}^{m'} \quad \mathbf{b}' = \sum_{i=1}^{2} x^{-i} \mathbf{b}_i \in \mathbb{S}^{m'}$

5.2.3 Perfect Completeness

An honest Prover always manages to convince a Verifier because of mathematical equalities. *(I cannot be bothered to write this down. Just show that the Prover's equations imply the Verifier's equations. Most are already the same. I have already spent way too much time on Bootle's protocol :P)*

5.2.4 Knowledge Soundness

We construct $\mathcal{E}$ recursively, like the protocol: Our goal is to recursively extract $\mathbf{a}$ from $\mathbf{a}'$ which we know from the previous step. The extraction of $\mathbf{b}$ from $\mathbf{b}'$ is analogous.

- Base case $m = 2$
  - Run $\mathcal{P}$ normally
  - Read $\mathbf{a}$ in plain and verify $A \overset{?}{=} \mathbf{aG}$
- Reduction
  - Run $\mathcal{P}$ until it sends its messages and make a copy
  - Run the copy 3 times for pairwise distinct values $x \in \mathbb{S}$
  - We know $\mathbf{a}'$ such that $A' = \mathbf{a}'\mathbf{G}'$ from previous step
  - Compute vectors $\mathbf{a}_{k,i}$ through interpolation of polynomial that defines A' (see below)
  - Write A_k as sum of $\mathbf{a}_{k,i}$ and $\mathbf{G}_i$ (see below)
  - Construct $\mathbf{a}$ as components from $\mathbf{a}_{k,i}$ (see below)
  - We have $A = \mathbf{aG}$

As you can see, the recursion starts from the *bottom*, i.e., the base case where every vector is sent in plain. Higher reduction steps rely on the fact that we know $\mathbf{a}'$. To do the full extraction, which starts from the *top*, we must run $\mathcal{P}$ a total of $m < 3^{\log_2 m} < m^2$ times, where $\log_2 m$ is the recursion depth. Thanks to the logarithmic exponent, the total time is polynomial.

Let's do the reduction in detail. We start with what we know: $A' = \mathbf{a}'\mathbf{G}'$.

$$\begin{aligned} x^{-1}A_{-1} + A_0 + xA_1 &= \mathbf{a}'(x^{-1}\mathbf{G}_1 + x^{-2}\mathbf{G}_2) \\ x^{-1}(A_{-1} + xA_0 + x^2A_1) &= x^{-1}\mathbf{a}'\mathbf{G}_1 + x^{-2}\mathbf{a}'\mathbf{G}_2 \\ &= \underbrace{\begin{bmatrix} x^{-1}a'_1 \\ x^{-1}a'_2 \\ x^{-2}a'_1 \\ x^{-2}a'_2 \end{bmatrix}}_{W(x)} \cdot \underbrace{\begin{bmatrix} G_{11} \\ G_{12} \\ G_{21} \\ G_{22} \end{bmatrix}}_{GG} \end{aligned}$$

We rearrange the equations to reveal that the LHS is a quadratic polynomial shifted by x^{-1} and the RHS is the dot product of vectors $W(x)$ and GG. We have three x values for which this equation holds, so we can rewrite it in terms of the Vandermonde matrix. The dot product on the RHS turns into matrix multiplication of $W \coloneqq [W(x_1), W(x_2), W(x_3)]^\top$ and GG.

$$\underbrace{\begin{bmatrix} -x_1^{-1} & 0 & 0 \\ 0 & -x_2^{-1} & 0 \\ 0 & 0 & -x_3^{-1} \end{bmatrix} \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{bmatrix}}_{V\colon\ 3\times 3} \underbrace{\begin{bmatrix} A^{-1} \\ A_0 \\ A_1 \end{bmatrix}}_{AA\colon\ 3\times 1} = \underbrace{W}_{3\times 4} \underbrace{GG}_{4\times 1}$$

This is technically a polynomial over two-dimensional points, but we can solve for each dimension separately. Make sure the dimensions of each matrix matches the matrix multiplication ($[n \times m][m \times o] = [n \times o]$). Now we invert the shifted Vandermonde matrix V to compute AA.

$$AA = V^{-1}(W\ GG) = (\underbrace{V^{-1}W}_{3\times 4})GG$$

Visualising the matrix multiplication on the RHS, the first row of $(V^{-1}W)$ multiplied with GG makes up A_{-1}, the second row makes up A_0 and the third row makes up A_1. We can think of $(V^{-1}W)$ as the following matrix of two-dimensional vectors $\mathbf{a}_{k,i}$ for $k = -1, 0, 1$ and $i = 1, 2$.

$$\begin{bmatrix} \mathbf{a}_{-1,1} & \mathbf{a}_{-1,2} \\ \mathbf{a}_{0,1} & \mathbf{a}_{0,2} \\ \mathbf{a}_{1,1} & \mathbf{a}_{1,2} \end{bmatrix}$$

We rewrite A_k in terms of these vectors $\mathbf{a}_{k,i}$ (whose value we know!).

$$A_k = \sum_{i=1}^{2} \mathbf{a}_{k,i}\mathbf{G}_i \quad \text{for } k = -1, 0, 1$$

We insert this new definition of A_k into the equation $A' = \mathbf{a}'\mathbf{G}'$ and simplify.

$$\sum_{k=-1}^{1} x^k \sum_{i=1}^{2} \mathbf{a}_{k,i}\mathbf{G}_i = \mathbf{a}' \sum_{i=1}^{2} x^{-i}\mathbf{G}_i$$
$$\sum_{i=1}^{\ell} \sum_{k=-1}^{1} x^k \mathbf{a}_{k,i}\mathbf{G}_i = \sum_{i=1}^{2} \mathbf{a}' x^{-i}\mathbf{G}_i$$

Because of binding, the two sums are equal iff each summand is equal.

$$\implies \sum_{k=-1}^{1} x^k \mathbf{a}_{k,i} = \mathbf{a}' x^{-i} \iff \mathbf{a}' = \sum_{k=-1}^{1} x^{k+i}\mathbf{a}_{k,i} \quad \text{for } i = 1, 2$$

This is a really strange equation for $\mathbf{a}'$ because it is *independent* of i! We rewrite this equality as follows:

$$\sum_{k=-1}^{1} x^{k+1}\mathbf{a}_{k,1} = \sum_{k=-1}^{1} x^{k+2}\mathbf{a}_{k,2}$$
$$\mathbf{a}_{-1,1} + x\mathbf{a}_{0,1} + x^2\mathbf{a}_{1,1} = x\mathbf{a}_{-1,2} + x^2\mathbf{a}_{0,2} + x^3\mathbf{a}_{1,2}$$

The above equations hold for random values of x, which restricts $\mathbf{a}_{k,i}$:

$$\mathbf{a}_{-1,1} = \mathbf{0} \quad \mathbf{a}_{0,1} = \mathbf{a}_{-1,2} \quad \mathbf{a}_{1,1} = \mathbf{a}_{0,2} \quad \mathbf{a}_{1,2} = \mathbf{0}$$

This can be restated as follows:

$$\mathbf{a}_{k,i} = \begin{cases} \mathbf{a}_{0,k+i} & \text{if } 1 \le k+i \le 2 \\ \mathbf{0} & \text{otherwise} \end{cases}$$

We define $\mathbf{a}_i := \mathbf{a}_{0,i}$. Remember that we have computed all vectors $\mathbf{a}_{k,i}$ in the above interpolation. We use the definition of $\mathbf{a}'$ from above, remove the zero terms and shift the sum.

$$\mathbf{a}' = \sum_{k=-1}^{1} \mathbf{a}_{k,1}x^{k+1} = \sum_{k=0}^{1} \mathbf{a}_{k+1}x^{k+1} = \sum_{i=1}^{2} \mathbf{a}_i x^i$$

Vector $\mathbf{a}$ is simply $\mathbf{a}_1$ appended with $\mathbf{a}_2$. By $A = A_0 = \mathbf{a}_1\mathbf{G}_1 + \mathbf{a}_2\mathbf{G}_2$ we get $A = \mathbf{a}\mathbf{G}$.

5.2.5 Soundness

We want to show that a valid transcript implies that $\mathbf{x} \cdot \mathbf{y} = z$ holds. Again, the proof is recursive. In particular, we want to show that $\mathbf{a}' \cdot \mathbf{b}' = z'$ implies $\mathbf{a} \cdot \mathbf{b} = z$ for every step.

The base case is trivial because $\mathbf{a} \cdot \mathbf{b} = z$ holds by definition.

Let us turn to the reduction step. From knowledge soundness, we know the openings of $\mathbf{a}'$ and $\mathbf{b}'$. From the previous step, we know that their dot product is equal to z'. The opening of z' is known to the Verifier.

$$\mathbf{a}' \cdot \mathbf{b}' = z'$$
$$\sum_{i=1}^{2} \mathbf{a}_i x^i + \sum_{i=1}^{2} \mathbf{b}_i x^{-i} = \sum_{k=-1}^{1} z_k x^{-k}$$
$$(\mathbf{a}_1 x + \mathbf{a}_2 x^2) \cdot (\mathbf{b}_1 x^{-1} + \mathbf{b}_2 x^{-2}) = xz_{-1} + z_0 + x^{-1}z_1$$
$$\mathbf{a}_1 \cdot \mathbf{b}_1 + \mathbf{a}_1 \cdot \mathbf{b}_2 x^{-1} + \mathbf{a}_2 \cdot \mathbf{b}_1 x + \mathbf{a}_2 \cdot \mathbf{b}_2 = \dots$$
$$x(\mathbf{a}_2 \cdot \mathbf{b}_1) + (\mathbf{a}_1 \cdot \mathbf{b}_1 + \mathbf{a}_2 \cdot \mathbf{b}_2) + x^{-1}(\mathbf{a}_1 \cdot \mathbf{b}_2) = \dots$$

This equation holds for at least three random values of x. Therefore the coefficients of each power of x must be equal, which gives us what we wanted.

$$\mathbf{a}_2 \cdot \mathbf{b}_1 = z_{-1} \quad \mathbf{a}_1 \cdot \mathbf{b}_1 + \mathbf{a}_2 \cdot \mathbf{b}_2 = z_0 = z \quad \mathbf{a}_1 \cdot \mathbf{b}_2 = z_1$$

6 Bulletproofs

A *bulletproof* consists of multiple elements: an inner product argument of logarithmic size and a range proof that makes use of the former. To recapitulate, in an inner product argument, the Prover claims to know two vectors $\mathbf{x}$ and $\mathbf{y}$ such $\mathbf{x}$ and $\mathbf{y}$ are openings of given commitments and such that their inner product $\mathbf{x} \cdot \mathbf{y}$ is equal to a given scalar z. In a *range proof* on the other hand, the Prover claims to know a scalar opening to a given commitment that is inside a given range. We will look into how such a statement can be phrased in terms of inner products below.

6.1 Logarithmic Inner Product

Prover: "I know an opening $\mathbf{a}$ and $\mathbf{b}$ to C such that $\mathbf{a} \cdot \mathbf{b} = z$."

6.1.1 Base Case

1. $\mathcal{P} \rightarrow \mathcal{V} : a, b$
2. $\mathcal{V} : z = a \cdot b \in \mathbb{S} \quad C \overset{?}{=} aG + bH + zI$

6.1.2 Reduction

1. $\mathcal{P}, \mathcal{V} : m' = \frac{m}{2}$
2. $\mathcal{P}$:
   - $z_L = \mathbf{a}_{\blacktriangleleft} \cdot \mathbf{b}_{\blacktriangleright} \in \mathbb{S}$
   - $z_R = \mathbf{a}_{\blacktriangleright} \cdot \mathbf{b}_{\blacktriangleleft} \in \mathbb{S}$
   - $L = \mathbf{a}_{\blacktriangleleft}\mathbf{G}_{\blacktriangleright} + \mathbf{b}_{\blacktriangleright}\mathbf{H}_{\blacktriangleleft} + z_L I \in \mathbb{E}$
   - $R = \mathbf{a}_{\blacktriangleright}\mathbf{G}_{\blacktriangleleft} + \mathbf{b}_{\blacktriangleleft}\mathbf{H}_{\blacktriangleright} + z_R I \in \mathbb{E}$
3. $\mathcal{P} \rightarrow \mathcal{V} : L, R$
4. $\mathcal{V} : x \leftarrow_{\$} \mathbb{S}$

5. $\mathcal{P} \leftarrow \mathcal{V} : x$

6. $\mathcal{P}, \mathcal{V} :$

- $\mathbf{G}' = x^{-1}\mathbf{G}_{\blacktriangleleft} + x\mathbf{G}_{\blacktriangleright} \in \mathbb{E}^{m'}$
- $\mathbf{H}' = x\mathbf{H}_{\blacktriangleleft} + x^{-1}\mathbf{H}_{\blacktriangleright} \in \mathbb{E}^{m'}$
- $C' = C + x^2 L + x^{-2} R \in \mathbb{E}$

7. $\mathcal{P} :$

- $\mathbf{a}' = x\mathbf{a}_{\blacktriangleleft} + x^{-1}\mathbf{a}_{\blacktriangleright} \in \mathbb{S}^{m'}$
- $\mathbf{b}' = x^{-1}\mathbf{b}_{\blacktriangleleft} + x\mathbf{b}_{\blacktriangleright} \in \mathbb{S}^{m'}$

6.1.3 Perfect Completeness

An honest Prover always manages to convince a Verifier because of mathematical equalities. *As is tradition at this point, we skip the proofs.*

6.1.4 Knowledge Soundness and Soundness

We show that an algorithm can extract the Prover's secret in polynomial time and that a valid transcript implies that $\mathbf{x} \cdot \mathbf{y} = z$ holds. The procedure is as in Bootle's protocol: We construct a recursive algorithm that extracts step i if step $i + 1$ has already been extracted, starting at the last step. Every step i requires a finite number of instances of the next step $i + 1$. To extract the secret of the first step requires exponentially many recursive calls in a tree of logarithmic depth. That's polynomially many calls and our polynomial extractor algorithm.

- Base case $m = 1$
  - Read a, b in plain, compute z
  - $C = aG + bH + zI$ and $a \cdot b = z$ hold by definition
- Reduction
  - Run $\mathcal{P}$ until it sends L, R and make a copy
  - Run the copy four times for pairwise distinct values $x \in \mathbb{S}$
  - We know $\mathbf{a}'_i$, $\mathbf{b}'_i$, z'_i for $1 \leq i \leq 4$ from previous step such that $C'_i = \mathbf{a}'_i\mathbf{G}'_i + \mathbf{b}_i\mathbf{H}'_i + z'_i I$ and $\mathbf{a}'_i \cdot \mathbf{b}'_i = z_i$ hold
  - Compute $\mathbf{a}$ and $\mathbf{b}$ by solving system of linear equations

– We have $C = \mathbf{a}\mathbf{G} + \mathbf{b}\mathbf{H} + zI$ and $\mathbf{a} \cdot \mathbf{b} = z$

We know the following. All these values are known.

$$\begin{aligned} C_i' &= \mathbf{a}_i'\mathbf{G}_i' + \mathbf{b}_i'\mathbf{H}_i' + z_i'I \\ &= \mathbf{a}_i'(x_i^{-1}\mathbf{G}_\blacktriangleleft + x_i\mathbf{G}_\blacktriangleright) + \mathbf{b}_i'(x_i\mathbf{H}_\blacktriangleleft + x_i^{-1}\mathbf{H}_\blacktriangleright) + (\mathbf{a}_i' \cdot \mathbf{b}_i')I \end{aligned}$$

We want to find an opening of C_i' of the following form:

$$\begin{aligned} C_i' &= C + x_i^2 L + x_i^{-2} R \\ &= \mathbf{a}\mathbf{G} + \mathbf{b}\mathbf{H} + x_i^2(\mathbf{a}_L\mathbf{G} + \mathbf{b}_L\mathbf{H} + z_LI) + x_i^{-2}(\mathbf{a}_R\mathbf{G} + \mathbf{b}_R\mathbf{H} + z_RI) \\ &= (\mathbf{a} + x^2\mathbf{a}_L + x^{-2}\mathbf{a}_R)\mathbf{G} + (\mathbf{b} + x^2\mathbf{b}_L + x^{-2}\mathbf{b}_R)\mathbf{H} + (z + x^2z_L + x^{-2}z_R)I \end{aligned}$$

Because of binding, the terms of the same generator vector must be equal:

$$\begin{aligned} \mathbf{a}_i'(x_i^{-1}\mathbf{G}_\blacktriangleleft + x_i\mathbf{G}_\blacktriangleright) &= (\mathbf{a} + x_i^2\mathbf{a}_L + x_i^{-2}\mathbf{a}_R)\mathbf{G} \\ \mathbf{b}_i'(x_i\mathbf{H}_\blacktriangleleft + x_i^{-1}\mathbf{H}_\blacktriangleright) &= (\mathbf{b} + x_i^2\mathbf{b}_L + x_i^{-2}\mathbf{b}_R)\mathbf{H} \\ (\mathbf{a}_i' \cdot \mathbf{b}_i')I &= (z + x_i^2z_L + x_i^{-2}z_R)I \end{aligned}$$

Because binding holds for each component of these generator vectors, the ◀ parts on the LHS must be equal those on the RHS, and similar for ▶ parts.

$$\begin{aligned} \mathbf{a}_i'x_i^{-1} &= \mathbf{a}_\blacktriangleleft + x_i^2\mathbf{a}_{L,\blacktriangleleft} + x_i^{-2}\mathbf{a}_{R,\blacktriangleleft} \\ \mathbf{a}_i'x_i &= \mathbf{a}_\blacktriangleright + x_i^2\mathbf{a}_{L,\blacktriangleright} + x_i^{-2}\mathbf{a}_{R,\blacktriangleright} \\ \mathbf{b}_i'x_i &= \mathbf{b}_\blacktriangleleft + x_i^2\mathbf{b}_{L,\blacktriangleleft} + x_i^{-2}\mathbf{b}_{R,\blacktriangleleft} \\ \mathbf{b}_i'x_i^{-1} &= \mathbf{b}_\blacktriangleright + x_i^2\mathbf{b}_{L,\blacktriangleright} + x_i^{-2}\mathbf{b}_{R,\blacktriangleright} \end{aligned}$$

These equations have the form $\alpha = x + \beta y + \gamma z$, where α, β, γ are known values. We can solve for x, y, z by providing three values α, β, γ, which we have thanks to three values x_i. At this point we have computed $\mathbf{a}$, $\mathbf{a}_L$, $\mathbf{a}_R$, $\mathbf{b}$, $\mathbf{b}_L$, $\mathbf{b}_R$, z, z_L and z_R. What remains to show is that these vectors have the right shape. Using the above equations, we multiply by x_i and x_i^{-1} to obtain two equations for $\mathbf{a}_i'$ respectively. One equation minus the other equals zero.

$$x_i'(\mathbf{a}_\blacktriangleleft + x_i^2\mathbf{a}_{L,\blacktriangleleft} + x_i^{-2}\mathbf{a}_{R,\blacktriangleleft}) - x_i^{-1}(\mathbf{a}_\blacktriangleright + x_i^2\mathbf{a}_{L,\blacktriangleright} + x_i^{-2}\mathbf{a}_{R,\blacktriangleright}) = 0$$

We multiply both sides by x_i^3 to arrive at a polynomial of degree four defined over x_i^2:

$$\begin{aligned} (x_i^2)^2\mathbf{a}_\blacktriangleleft + (x_i^2)^4\mathbf{a}_{L,\blacktriangleleft} + (x_i^2)\mathbf{a}_{R,\blacktriangleleft} - (x_i^2)\mathbf{a}_\blacktriangleright - (x_i^2)^2\mathbf{a}_{L,\blacktriangleright} - \mathbf{a}_{R,\blacktriangleright} &= 0 \\ (x_i^2)^4\mathbf{a}_{L,\blacktriangleleft} + (x_i^2)^2(\mathbf{a}_\blacktriangleleft - \mathbf{a}_{L,\blacktriangleright}) + (x_i^2)(\mathbf{a}_{R,\blacktriangleleft} - \mathbf{a}_\blacktriangleright) - \mathbf{a}_{R,\blacktriangleright} &= 0 \end{aligned}$$

This polynomial is zero at four values of x_i, so its coefficients must be zero. We find that the $\blacktriangleleft$ side of $\mathbf{a}_L$ and the $\blacktriangleright$ side of $\mathbf{a}_R$ are zero, as required. Also $\mathbf{a}_L$ and $\mathbf{a}_R$ copy from $\mathbf{a}$.

$$\mathbf{a}_{L,\blacktriangleleft} = \mathbf{a}_{R,\blacktriangleright} = \mathbf{0} \quad \mathbf{a}_{\blacktriangleleft} = \mathbf{a}_{L,\blacktriangleright} \quad \mathbf{a}_{R,\blacktriangleleft} = \mathbf{a}_{\blacktriangleright}$$

A similar analysis can be done for $\mathbf{b}$, $\mathbf{b}_L$ and $\mathbf{b}_R$. This establishes knowledge soundness. What remains to show is that z is a sound dot product of $\mathbf{a}$ and $\mathbf{b}$. Using the above equalities, we rewrite $\mathbf{a}'_i$ and $\mathbf{b}'_i$ in terms of $\mathbf{a}$ and $\mathbf{b}$.

$$\begin{aligned} x_i^{-1}\mathbf{a}'_i &= \mathbf{a}_{\blacktriangleleft} + x_i^2\mathbf{0} + x_i^{-2}\mathbf{a}_{\blacktriangleright} \\ \mathbf{a}'_i &= x_i\mathbf{a}_{\blacktriangleleft} + x_i^{-1}\mathbf{a}_{\blacktriangleright} \\ \mathbf{b}'_i &= x_i^{-1}\mathbf{b}_{\blacktriangleleft} + x_i\mathbf{b}_{\blacktriangleright} \end{aligned}$$

We have an equation for $\mathbf{a}'_i \cdot \mathbf{b}'_i$ from above. We substitute the new definitions of $\mathbf{a}'_i$ and $\mathbf{b}'_i$ and simplify.

$$\begin{aligned} z + x_i^2 z_L + x_i^{-2} z_R &= \mathbf{a}'_i \cdot \mathbf{b}'_i \\ &= (x_i\mathbf{a}_{\blacktriangleleft} + x_i^{-1}\mathbf{a}_{\blacktriangleright}) \cdot (x_i^{-1}\mathbf{b}_{\blacktriangleleft} + x_i\mathbf{b}_{\blacktriangleright}) \\ &= \mathbf{a}_{\blacktriangleleft} \cdot \mathbf{b}_{\blacktriangleleft} + x_i^2\mathbf{a}_{\blacktriangleleft} \cdot \mathbf{b}_{\blacktriangleright} + x_i^{-2}\mathbf{a}_{\blacktriangleright} \cdot \mathbf{b}_{\blacktriangleleft} + \mathbf{a}_{\blacktriangleright} \cdot \mathbf{b}_{\blacktriangleright} \\ &= \mathbf{a} \cdot \mathbf{b} + x_i^2\mathbf{a}_{\blacktriangleleft} \cdot \mathbf{b}_{\blacktriangleright} + x_i^{-2}\mathbf{a}_{\blacktriangleright} \cdot \mathbf{b}_{\blacktriangleleft} \end{aligned}$$

This equation holds for (at least) one random x value in $\mathbb{S}$. By the Schwartz-Zippel lemma, the coefficients are equal with probability greater equal $1 - \frac{2}{n} \approx 1$. This establishes soundness.

$$z = \mathbf{a} \cdot \mathbf{b} \quad z_L = \mathbf{a}_{\blacktriangleleft} \cdot \mathbf{b}_{\blacktriangleright} \quad z_R = \mathbf{a}_{\blacktriangleright} \cdot \mathbf{b}_{\blacktriangleleft}$$

6.2 Linear Range Proof

We want to prove that a Pedersen commitment has an opening that is inside a given range. This is useful for confidential transactions, where the amounts of spent coins are hidden in commitments, but we still need to verify that the number of coins going out of the transaction is less equal the number of coins going in (lest we have inflation). Such a commitment has the following form, where $\gamma \in \mathbb{S}$ is the blinding factor and $v \in \mathbb{S}$ is the value.

$$V = \gamma H + vG$$

The Prover claims "I know an opening v of V such that $v \in [0, 2^m)$". This is well and good, but we need to transform this statement to make it work in an interactive proof. We start with the original statement:

$$v \in [0, 2^m)$$

This holds iff we can represent v in a bit vector of size m, which we call $\mathbf{a}_L$. To make sure that $\mathbf{a}_L$ is indeed a bit vector (with components 0 and 1), we introduce another bit vector $\mathbf{a}_R$ that is $\mathbf{a}_L$ minus one, and require that their component-wise product is zero. You can check for yourself that this is equivalent to v being in $[0, 2^m)$.

$$\mathbf{a}_L \cdot \mathbf{2}^m = v \quad \mathbf{a}_R = \mathbf{a}_L - \mathbf{1} \quad \mathbf{a}_L \circ \mathbf{a}_R = \mathbf{0}$$

Next, we phrase these conditions in terms of inner products using challenge $y \in \mathbb{S}$. The above holds iff the below holds for all challenges $y \in \mathbb{S}$.

$$\mathbf{a}_L \cdot \mathbf{2}^m = v \quad (\mathbf{a}_L - \mathbf{1}^m - \mathbf{a}_R) \cdot \mathbf{y}^m = 0 \quad (\mathbf{a}_L \circ \mathbf{a}_R) \cdot \mathbf{y}^m = 0$$

Now, we combine the inner products into a single equation using another challenge $z \in \mathbb{S}$. The above holds iff the below holds for all challenges $z \in \mathbb{S}$.

$$z^2(\mathbf{a}_L \cdot \mathbf{2}^m) + z((\mathbf{a}_L - \mathbf{1}^m - \mathbf{a}_R) \cdot \mathbf{y}^m) + (\mathbf{a}_L \circ \mathbf{a}_R) \cdot \mathbf{y}^m = z^2 v$$

Finally, we go through a bunch of mathematical transformations that I will skip here to arrive at a single inner product. We give each term a name to better keep track of it.

$$\underbrace{(\mathbf{a}_L - z\mathbf{1}^m)}_{=:\boldsymbol{\ell}} \cdot \underbrace{(\mathbf{y}^m \circ (\mathbf{a}_R + z\mathbf{1}^m) + z^2\mathbf{2}^m)}_{=:\mathbf{r}} = \underbrace{z^2 v + \delta(y,z)}_{=:t}$$
$$(z - z^2)(\mathbf{1}^m \cdot \mathbf{y}^m) - z^3(\mathbf{1}^m \cdot \mathbf{2}^m) =: \delta(y,z)$$

There is another problem: The vectors $\mathbf{a}_L$ and $\mathbf{a}_R$ leak the value of v! We need this protocol to be zero-knowledge, so we commit to these vectors in a Pedersen commitment. To transform $\boldsymbol{\ell}$ and $\mathbf{r}$, we introduce blinding vectors $\mathbf{s}_L$ and $\mathbf{s}_R$ that we also commit to.

$$A = \alpha H + \mathbf{a}_L\mathbf{G} + \mathbf{a}_R\mathbf{H} \quad S = \rho H + \mathbf{s}_L\mathbf{G} + \mathbf{s}_R\mathbf{H} \quad \text{where } \mathbf{s}_L, \mathbf{s}_R \leftarrow\!\$\, \mathbb{S}^m$$

We incorporate $\mathbf{s}_L$ and $\mathbf{s}_R$ into $\boldsymbol{\ell}$ and $\mathbf{r}$ in a clever way to obtain $\boldsymbol{\ell}(x)$ and $\mathbf{r}(x)$, respectively. The useful property of these functions is that they contain the original vectors $\boldsymbol{\ell}$ and $\mathbf{r}$, respectively, plus some function of x.

$$\boldsymbol{\ell}(x) \coloneqq (\mathbf{a}_L - z\mathbf{1}^m) + \mathbf{s}_L x = \ell + \mathbf{s}_L x = \boldsymbol{\ell} + \boldsymbol{\ell}'(x)$$
$$\mathbf{r}(x) \coloneqq \mathbf{y}^m \circ (\mathbf{a}_R + z\mathbf{1}^m + \mathbf{s}_R x) + z^2\mathbf{2}^m = r + \mathbf{y}^m \circ \mathbf{s}_R x = \mathbf{r} + \mathbf{r}'(x)$$

Using these definitions and the law of distributivity, we can see that the dot product of $\boldsymbol{\ell}(x)$ and $\mathbf{r}(x)$ is the dot product of $\boldsymbol{\ell}$ and $\mathbf{r}$ plus two coefficients times a power of x.

$$\boldsymbol{\ell}(x) \cdot \mathbf{r}(x) = \boldsymbol{\ell} \cdot \mathbf{r} + \underbrace{\boldsymbol{\ell} \cdot \mathbf{r}'(x) + \boldsymbol{\ell}'(x) \cdot \mathbf{r}}_{:=t_1 x} + \underbrace{\boldsymbol{\ell}'(x) \cdot \mathbf{r}'(x)}_{:=t_2 x^2}$$
$$t_1 := (\mathbf{a}_L - z\mathbf{1}^m) \cdot (\mathbf{y}^m \circ \mathbf{s}_R) + \mathbf{s}_L \cdot (\mathbf{y}^m \circ (\mathbf{a}_R + z\mathbf{1}^m) + z^2\mathbf{2}^m)$$
$$t_2 := \mathbf{s}_L \cdot (\mathbf{y}^m \circ \mathbf{s}_R)$$

Make sure to distinguish which equalities hold by definition and which equalities imply our original statement. The magic happens here: If $\boldsymbol{\ell} \cdot \mathbf{r} = t$, then v is in the range. In other words, the following statement holds for all challenges $x, y, z \in \mathbb{S}$ iff $v \in [0, 2^m)$.

$$\boldsymbol{\ell}(x) \cdot \mathbf{r}(x) = t(x) := t + t_1 x + t_2 x^2$$

6.2.1 Protocol

1. $\mathcal{P}$:
   - $A = \alpha H + \mathbf{a}_L \mathbf{G} + \mathbf{a}_R \mathbf{H} \in \mathbb{E}$ where $\alpha \leftarrow\!\$\, \mathbb{S}$
   - $S = \rho H + \mathbf{s}_L \mathbf{G} + \mathbf{s}_R \mathbf{H} \in \mathbb{E}$ where $\rho \leftarrow\!\$\, \mathbb{S}, \mathbf{s}_L, \mathbf{s}_R \leftarrow\!\$\, \mathbb{S}^m$
2. $\mathcal{P} \rightarrow \mathcal{V} : A, S$
3. $\mathcal{V} : y, z \leftarrow\!\$\, \mathbb{S}$
4. $\mathcal{P} \leftarrow \mathcal{V} : y, z$
5. $\mathcal{P}$:
   - $T_1 = \tau_1 H + t_1 G \in \mathbb{E}$ where $\tau_1 \leftarrow\!\$\, \mathbb{S}$
   - $T_2 = \tau_2 H + t_2 G \in \mathbb{E}$ where $\tau_2 \leftarrow\!\$\, \mathbb{S}$
6. $\mathcal{P} \rightarrow \mathcal{V} : T_1, T_2$
7. $\mathcal{V} : x \leftarrow\!\$\, \mathbb{S}$
8. $\mathcal{P} \leftarrow \mathcal{V} : x$
9. $\mathcal{P}$:
   - $\tau_x = \tau_2 x^2 + \tau_1 x + z^2 \gamma \in \mathbb{S}$
   - $\mu = \alpha + \rho x \in \mathbb{S}$
   - $\hat{\boldsymbol{\ell}} = \boldsymbol{\ell}(x) \in \mathbb{S}^m$

- $\hat{\mathbf{r}} = \mathbf{r}(x) \in \mathbb{S}^m$
- $\hat{t} = \hat{\boldsymbol{\ell}} \cdot \hat{\mathbf{r}} \in \mathbb{S}$

10. $\mathcal{P} \to \mathcal{V} : \tau_x, \mu, \hat{t}, \hat{\boldsymbol{\ell}}, \hat{\mathbf{r}}$

11. $\mathcal{V}:$

- $\tau_x H + \hat{t} G \stackrel{?}{=} z^2 V + \delta(y,z) G + x T_1 + x^2 T_2$
- $\mathbf{H}' = \mathbf{y}^{-m} \circ \mathbf{H} \in \mathbb{E}^m$
- $P = A + xS - z\mathbf{G} + (z\mathbf{y}^m + z^2 \mathbf{2}^m)\mathbf{H}' \in \mathbb{E}$
- $P \stackrel{?}{=} \mu H + \hat{\boldsymbol{\ell}} \mathbf{G} + \hat{\mathbf{r}} \mathbf{H}'$
- $\hat{t} \stackrel{?}{=} \hat{\boldsymbol{\ell}} \cdot \hat{\mathbf{r}}$

6.2.2 Perfect Completeness

Skipped.

6.2.3 Knowledge Soundness

We have to show that a successful interaction with a prover implies knowledge of an opening v of V. While we are at it, we also compute all other values. We construct an extractor as follows:

1. Run $\mathcal{P}$ until it sends T_1, T_2 and make a copy
2. Run copy three times for pairwise distinct values $x \in \mathbb{S}$
3. Compute all values using binding and systems of linear equations

A valid transcript gives us the following equality for $i = 1, 2, 3$. Make sure to distinguish values which we already know $(x_i, y, z, \ldots)$ and values which we want to compute $(\gamma, v, \tau_1, \ldots)$.

$$\begin{aligned}\tau_{x_i} H + \hat{t} G &= z^2 V + \delta(y,z) G + x_i T_1 + x_i^2 T_2 \\ &= z^2(\gamma H + vG) + \delta(y,z) G + x_i(\tau_1 H + t_1 G) + x_i^2(\tau_2 H + t_2 G)\end{aligned}$$

The coefficients of each generator must be equal. Each equation has three unknowns and holds for three values of x_i, so we can solve it.

$$\begin{aligned}\tau_{x_i} &= z^2 \gamma + x_i \tau_1 + x_i^2 \tau_2 && \text{solve } \gamma, \tau_1, \tau_2 \\ \hat{t} &= z^2 v + \delta(y,z) + x_i t_1 + x_i^2 t_2 && \text{solve } v, t_1, t_2\end{aligned}$$

A valid transcript also gives us the following for $i = 1, 2$.

$$\begin{aligned} A + x_i S - z\mathbf{G} + (z\mathbf{y}^m + z^2\mathbf{2}^m)\mathbf{H}' &= \mu H + \hat{\boldsymbol{\ell}}_i \mathbf{G} + \hat{\mathbf{r}}_i \mathbf{H}' \\ A &= \alpha H + \mathbf{a}_L \mathbf{G} + \mathbf{a}_R \mathbf{H} \\ S &= \rho H + \mathbf{s}_L \mathbf{G} + \mathbf{s}_R \mathbf{H} \end{aligned}$$

Again, the coefficients of each generator must be equal. We solve these equations with two unknowns via two values of x_i.

$$\begin{aligned} \alpha + x_i \rho &= \mu && \text{solve } \alpha, \rho \\ \mathbf{a}_L + x_i \mathbf{s}_L - z &= \hat{\boldsymbol{\ell}}_i && \text{solve } \mathbf{a}_L, \mathbf{s}_L \\ \mathbf{a}_R + x_i \mathbf{s}_R + (z\mathbf{y}^m + z^2\mathbf{2}^m) \circ \mathbf{y}^{-m} &= \hat{\mathbf{r}}_i \circ \mathbf{y}^{-m} \\ (\mathbf{a}_R + x_i \mathbf{s}_R + z) \circ \mathbf{y}^m + z^2\mathbf{2}^m &= \hat{\mathbf{r}}_i && \text{solve } \mathbf{a}_R, \mathbf{s}_R \end{aligned}$$

6.2.4 Soundness

We have to show that a successful interaction with a prover implies $v \in [0, 2^m)$. This involves reversing the process that let us to construct $\boldsymbol{\ell}(x)$, $\mathbf{r}(x)$ and $t(x)$ in the first place.

Looking at the equations for $\hat{\boldsymbol{\ell}}_i$ and $\hat{\mathbf{r}}_i$ from knowledge soundness, we can see that $\hat{\boldsymbol{\ell}}_i = \boldsymbol{\ell}(x_i)$ and $\hat{\mathbf{r}} = \mathbf{r}(x_i)$ holds for $i = 1, 2, 3$. The valid transcript gives us $\hat{\boldsymbol{\ell}}_i \cdot \hat{\mathbf{r}}_i = \hat{t}_i$. Combining these facts gives us the following:

$$\boldsymbol{\ell}(x_i) \cdot \mathbf{r}(x_i) = t + x_i t_1 + x_i^2 t_2$$

This equates two quadratic polynomials *(see definitions)* over three values of x_i. The coefficients of each power of x_i must necessarily be equal:

$$\boldsymbol{\ell} \cdot \mathbf{r} = t \quad \boldsymbol{\ell} \cdot \mathbf{r}'(x_i) + \boldsymbol{\ell}'(x_i) \cdot \mathbf{r} = x_i t_1 \quad \boldsymbol{\ell}'(x_i) \cdot \mathbf{r}'(x_i) = x_i t_2$$

Taking the leftmost equation, we reverse the transformations we did at the beginning of this section to arrive at:

$$z^2(\mathbf{a}_L \cdot \mathbf{2}^m) + z((\mathbf{a}_L - \mathbf{1}^m - \mathbf{a}_R) \cdot \mathbf{y}^m) + (\mathbf{a}_L \circ \mathbf{a}_R) \cdot \mathbf{y}^m = z^2 v$$

This is a bivariate polynomial over y and z of degree $m + 1$, which no longer includes any x. So far, we have only used one combination of y, z. If we want, we can run the Prover for $m + 2$ pairwise distinct combinations of y, z to obtain the above equality $m + 2$ times. Again, the coefficients of each power of y, z must be equal *(including the zeroes on the RHS).*

$$\mathbf{a}_L \cdot \mathbf{2}^m = v \quad \mathbf{a}_L - \mathbf{1}^m - \mathbf{a}_R = 0 \quad \mathbf{a}_L \circ \mathbf{a}_R = 0$$

As we have discussed at the beginning, these equations imply that $\mathbf{a}_L$ is a bit representation of v. This obviously puts v in the range $[0, 2^m)$ since $\mathbf{a}_L$ has size m.

6.2.5 Honest-Verifier Perfect ZK

Finally another zero-knowledge proof *(of a zero-knowledge proof (?))*. Jokes aside, let's get started.

We construct a simulator for an honest verifier as follows:

1. $A, T_2 \leftarrow\$\, \mathbb{E} \quad \tau_x, \mu, x, y, z, \hat{t} \leftarrow\$\, \mathbb{S}$
2. $T_1 = x^{-1}((\hat{t} - \delta(y,z))G - z^2 V - x^2 T_2 + \tau_x H)$
3. Compute $\hat{\boldsymbol{\ell}}, \hat{\mathbf{r}} \in \mathbb{S}^m$ such that $\hat{\boldsymbol{\ell}} \cdot \hat{\mathbf{r}} = \hat{t}$ holds. This works for any $\hat{t} \in \mathbb{S}$.
4. $S = x^{-1}(\mu H + z\mathbf{G} + \hat{\boldsymbol{\ell}}\mathbf{G} + (\hat{\mathbf{r}} - z\mathbf{y}^m - z^2\mathbf{2}^m)\mathbf{H}' - A)$

6.3 Logarithmic Range Proof

6.3.1 Protocol

10. $\mathcal{P} \rightarrow \mathcal{V} : \tau_x, \mu, \hat{t}$
11. $\mathcal{P}, \mathcal{V}:$
    - $\mathbf{H}' = \mathbf{y}^{-m} \circ \mathbf{H} \in \mathbb{E}^m$
    - $P = A + xS - z\mathbf{G} + (z\mathbf{y}^m + z^2\mathbf{2}^m)\mathbf{H}' \in \mathbb{E}$
12. $\mathcal{V} : \tau_x H + \hat{t}G \overset{?}{=} z^2 V + \delta(y,z)G + xT_1 + x^2 T_2$
13. $\mathcal{P}$: Prove knowledge of $\hat{\boldsymbol{\ell}}, \hat{\mathbf{r}}$ such that $(P - \mu H) = \hat{\boldsymbol{\ell}}\mathbf{G} + \hat{\mathbf{r}}\mathbf{H}'$ and $\hat{\boldsymbol{\ell}} \cdot \hat{\mathbf{r}} = \hat{t}$

6.3.2 Perfect Completeness

Same as for the linear range proof. The conditions that make up a valid transcript have not changed and we have merely outsourced the proof of the inner product, which is complete.

6.3.3 Knowledge Soundness and Soundness

Surprisingly, the same soundness proofs as for the linear case work for the logarithmic case. We have to make two adjustments: Because the inner product proof is knowledge-sound, the Extractor obtains the openings $\hat{\boldsymbol{\ell}}$ and $\hat{\mathbf{r}}$ from the Prover. Second, because the inner product proof is sound, we get $P - \mu H = \hat{\boldsymbol{\ell}}\mathbf{G} + \hat{\mathbf{r}}\mathbf{H}'$, equivalent to $P = \mu H + \hat{\boldsymbol{\ell}}\mathbf{G} + \hat{\mathbf{r}}\mathbf{H}'$, and $\hat{\boldsymbol{\ell}} \cdot \hat{\mathbf{r}} = \hat{t}$. This is all we need; the rest of the proof is exactly as for the linear case. Check for yourself.