Understanding Groth16: A Tutorial Conversation

A dialogue exploring zkSNARKs and the Groth16 proving system

Contents

1 Introduction and Bilinear Pairings

User: In a similar spirit to our recent 'quantum computing for physics graduates', please lead me through with concept check questions, the zkSNARK system Groth16.

Claude: Excellent! Groth16 is a beautiful piece of cryptographic engineering, and your physics/math background will serve you well here. Let me structure this similarly to our quantum computing exploration.

1.1 Starting Point: What Problem Are We Solving?

Imagine you want to prove to me that you know a solution to a sudoku puzzle, *without revealing the solution itself.* More generally: you want to prove you possess some secret information w (the "witness") that satisfies some public computation $C(x, w) = \text{true}$, where x is public input.

A **zkSNARK** achieves this with three remarkable properties:

- **Zero-knowledge**: I learn nothing about w beyond the fact that it exists
- **Succinct**: The proof is tiny (constant size, ∼200 bytes for Groth16) regardless of how complex C is
- **Non-interactive**: You send me one message; no back-and-forth

1.2 The Mathematical Foundation: Bilinear Pairings

Groth16's magic rests on a cryptographic primitive called a **bilinear pairing**. This is where your physics intuition might help.

Consider two groups G_1 and G_2 (written additively), and a target group G_T (written multiplicatively). A pairing is a map:

$$e : G_1 \times G_2 \to G_T$$

with the crucial **bilinearity** property:

$$e(aP, bQ) = e(P, Q)^{ab}$$

for any scalars a, b and group elements $P \in G_1$, $Q \in G_2$.

1.2.1 Concept Check 1

Suppose I give you elements aP and bQ (but you don't know a or b individually). Using the pairing, can you compute $e(P, Q)^{ab}$ without knowing a or b separately?

And a follow-up: if I give you aP, bP, and cP (all in G_1), can you determine whether $c = ab$ using pairings? (Assume you also have access to some generator $Q \in G_2$.)

User: 1. yes, take aP and bQ and calculate the pairing $e(aP, bQ)$ and that is the answer. 2. yes calculate $e(aP, bQ)$ and then calculate $e(cP, Q)$.

Claude: Spot on for the first one.

For the second, you've got exactly the right *idea*, but there's a subtle issue worth flagging: I gave you aP, bP, cP all in G_1. To compute $e(aP, bQ)$, you'd need $bQ \in G_2$, but you only have $bP \in G_1$.

This is actually a crucial point for Groth16: **you can't freely move elements between G_1 and G_2**. There's no efficient way to convert bP into bQ.

So the corrected version: if someone provides you with both $bP \in G_1$ *and* $bQ \in G_2$ (or equivalently aQ), then yes, you can check:

$$e(aP, bQ) \stackrel{?}{=} e(cP, Q)$$

Both sides equal $e(P, Q)^{ab}$ and $e(P, Q)^c$ respectively, so equality holds iff $c = ab$.

This asymmetry between G_1 and G_2 is precisely why Groth16's **trusted setup** must carefully provide the right elements in both groups.

User: sorry i misread. let me try again. you were asking if i have aP, bP and cP can i check if $ab = c$. I didn't read your correct answer. thinking about that, it seems to be the DDH problem. I have a vague memory that if we have pairings, then DDH is *not* hard while CDH is. Is that right? So I think then I would do $e(aP, bP)$ and $e(cP, P)$ and check if they were equal?

Claude: Your cryptographic intuition is exactly right! This is a key insight.

DDH (Decisional Diffie-Hellman): Given (P, aP, bP, cP), decide whether $c = ab$.

CDH (Computational Diffie-Hellman): Given (P, aP, bP), compute abP.

In ordinary discrete-log groups, both are believed hard. But pairings create a "trapdoor" that breaks DDH while leaving CDH intact. This is precisely why pairing-based cryptography is so powerful — you get just enough algebraic structure to do useful things, without breaking everything.

Your solution $e(aP, bP) \stackrel{?}{=} e(cP, P)$ is correct for **symmetric pairings** (where $G_1 = G_2$). These exist but are rarer.

Groth16 typically uses **Type III pairings** (asymmetric, no efficient map between G_1 and G_2), which are more efficient in practice. So you'd need the trusted setup to provide, say, $aP \in G_1$ and $aQ \in G_2$ for the scalars you need to "mix."

This is actually a nice preview of *why* Groth16 needs a structured trusted setup: it must provide carefully chosen elements in both groups so that the verifier can perform exactly the right pairing checks — no more, no less.

Shall we continue to the R1CS concept check, or do you want to dig deeper into pairings first?

User: so i didn't quite understand; is there a solution to your problem as stated for the non-symmetric case? i need to get (ab) in the exponent, but you only gave me aP, bP right?

Claude: You're absolutely right to push back — this is precisely the kind of careful thinking that matters in cryptography.

No, as stated, the problem is unsolvable in the asymmetric case.

If I only give you $aP, bP, cP \in G_1$, you're stuck. The pairing $e(aP, bP)$ isn't even well-typed: e requires one argument from G_1 and one from G_2. You have no way to get ab into an exponent without having at least one of a or b "represented" in G_2.

So for the DDH check to work in Type III pairings, you'd need something like:

- $aP \in G_1$, $bQ \in G_2$, $cP \in G_1$, and the generator $Q \in G_2$

Then you compute $e(aP, bQ) \stackrel{?}{=} e(cP, Q)$.

This is exactly why I mentioned the trusted setup must "carefully provide the right elements in both groups." It's not just convenience — certain verification equations are literally impossible without the setup providing, say, both τP and τQ for the secret scalar τ.

Excellent catch. This asymmetry is a real constraint that shapes the entire structure of Groth16.

Ready to move on to R1CS and why we need intermediate variables?

2 Arithmetization: R1CS

User: yes let's do R1CS carefully. I find this part difficult.

Claude: Good, let's build this carefully from the ground up.

2.1 The Goal

We want to convert an arbitrary computation into a form where:

1. It's purely algebraic (so we can use polynomial/pairing tricks)
2. A verifier can check it efficiently
3. We can later "hide" parts of it using zero-knowledge techniques

2.2 The Core Constraint: One Multiplication Per Equation

The fundamental building block of R1CS is:

$$A \times B = C$$

where A, B, and C are each **linear combinations** of your variables.

Linear means: additions and scalar multiplications only. So A might be something like $3w_1 + 2w_2 - 5$.

This is a severe restriction. You get exactly **one** multiplication per constraint equation — the $\times$ between A and B. Everything inside A, B, and C individually must be linear.

2.2.1 Concept Check 3

Which of these could be a single valid R1CS constraint?

(a) $w_1 \times w_2 = w_3$

(b) $w_1 \times w_2 \times w_3 = w_4$

(c) $(w_1 + w_2) \times (w_3 + w_4) = w_5$

(d) $w_1 \times w_1 = w_2$

(e) $(w_1 \times w_2) + w_3 = w_4$

User: a, c and d seem uncontroversially valid, b seems clearly not valid. e is not so obvious to me, it doesn't perfectly fit but it *is* a single multiplication. can we just define C as $w_4 - w_3$ I guess and then it's fine?

Claude: Exactly right. You've understood something important.

The canonical form $A \times B = C$ doesn't require C to be a single variable — it just needs to be a **linear combination** of variables. So for (e):

$$w_1 \times w_2 = w_4 - w_3$$

is perfectly valid with $A = w_1$, $B = w_2$, $C = w_4 - w_3$.

So all of (a), (c), (d), and (e) are valid single constraints. Only (b) fails because it has two multiplications baked in, and there's no algebraic rearrangement that fixes that.

2.3 The Witness Vector

Now let's talk about how these constraints are represented systematically. We collect all our variables into a single vector $\vec{s}$, conventionally structured as:

$$\vec{s} = (1, x_1, x_2, \ldots, x_m, w_1, w_2, \ldots, w_n)$$

- The leading 1 lets us include constants in our linear combinations
- The x_i are **public inputs** (known to verifier)
- The w_j are the **witness** (secret, known only to prover)

Each constraint is then expressed using three vectors $\vec{a}_i$, $\vec{b}_i$, $\vec{c}_i$ such that:

$$(\vec{a}_i \cdot \vec{s}) \times (\vec{b}_i \cdot \vec{s}) = (\vec{c}_i \cdot \vec{s})$$

2.3.1 Concept Check 4

Let's take our earlier example: proving I know w such that $w^3 = x$.

We said we need $\vec{s} = (1, x, w, v)$ where $v = w^2$, with two constraints:

- Constraint 1: $w \times w = v$
- Constraint 2: $w \times v = x$

Can you write out the vectors $\vec{a}_1$, $\vec{b}_1$, $\vec{c}_1$ for the first constraint?

User: $\vec{a}_1 = [0, 0, 1, 0]$, $\vec{b}_1 = [0, 0, 1, 0]$, $\vec{c}_1 = [0, 0, 0, 1]$

Claude: Perfect. Let's just verify:

- $\vec{a}_1 \cdot \vec{s} = 0 \cdot 1 + 0 \cdot x + 1 \cdot w + 0 \cdot v = w$ ✓
- $\vec{b}_1 \cdot \vec{s} = 0 \cdot 1 + 0 \cdot x + 1 \cdot w + 0 \cdot v = w$ ✓
- $\vec{c}_1 \cdot \vec{s} = 0 \cdot 1 + 0 \cdot x + 0 \cdot w + 1 \cdot v = v$ ✓

So the constraint says $w \times w = v$. Exactly right.

2.3.2 Concept Check 5

Now write out $\vec{a}_2$, $\vec{b}_2$, $\vec{c}_2$ for the second constraint: $w \times v = x$.

And then a conceptual question: we now have two constraints represented as vector equations. From the verifier's perspective, what do they know, and what are they convinced of once they've checked these constraints are satisfied?

User: $\vec{a}_2 = [0, 0, 1, 0]$, $\vec{b}_2 = [0, 0, 0, 1]$, $\vec{c}_2 = [0, 1, 0, 0]$. Concept Q: verifier knows x (public). Is convinced that the prover knows (because he has chosen as input and it's valid) a w such that $w^3 = x$.

Claude: Vectors are correct. And yes, you've captured the essence: the verifier knows x, and becomes convinced that the prover possesses some w satisfying the claimed relationship, without learning w itself.

3 From R1CS to Polynomials: The QAP Transform

Here's where things get clever. We have m constraints, each defined by vectors $(\vec{a}_i, \vec{b}_i, \vec{c}_i)$. We want to compress checking all m constraints into checking a **single polynomial equation**.

The trick: instead of thinking of $\vec{a}_i$ as vectors indexed by constraint number i, we **interpolate polynomials** through these values.

For each position j in the witness vector, define polynomials $A_j(x)$, $B_j(x)$, $C_j(x)$ such that:

$$A_j(\omega^i) = (a_i)_j$$

where ω is a primitive m-th root of unity, and $(a_i)_j$ is the j-th component of vector $\vec{a}_i$.

In other words: we pick m special points $\{\omega^1, \omega^2, \ldots, \omega^m\}$, and we build polynomials that "encode" the constraint vectors at these points.

3.0.1 Concept Check 6

If we have m constraints and we're interpolating polynomials through m points, what degree will the polynomials $A_j(x)$ be (at most)?

User: interpolating through m points gives us a polynomial of degree $m - 1$

Claude: Exactly. Lagrange interpolation through m points yields a unique polynomial of degree at most $m - 1$.

3.1 Building the QAP

Now we have polynomials $A_j(x)$, $B_j(x)$, $C_j(x)$ for each variable index j. The prover combines these using their witness vector $\vec{s}$:

$$A(x) = \sum_j s_j \cdot A_j(x)$$
$$B(x) = \sum_j s_j \cdot B_j(x)$$
$$C(x) = \sum_j s_j \cdot C_j(x)$$

Here's the key insight: by construction, when we evaluate at $x = \omega^i$:

$$A(\omega^i) = \sum_j s_j \cdot A_j(\omega^i) = \sum_j s_j \cdot (a_i)_j = \vec{a}_i \cdot \vec{s}$$

So evaluating $A(x)$ at the special points recovers exactly the left-hand side of each constraint!

3.1.1 Concept Check 7

Given this setup, what does the equation $A(\omega^i) \cdot B(\omega^i) = C(\omega^i)$ represent?

And if we want **all** m constraints to hold simultaneously, what can we say about the polynomial $A(x) \cdot B(x) - C(x)$ at the points $\{\omega^1, \omega^2, \ldots, \omega^m\}$?

User: $A(\omega^i) \cdot B(\omega^i) = C(\omega^i)$ would represent the i-th constraint (i.e. it is true if and only if the constraint is true). so $A(x) \cdot B(x) - C(x)$ would be zero at each of the m m-th roots of unity, which means that .. it should be like $P(x) = x^m - 1$ (since those are the roots of that polynomial).

Claude: You're almost there — just a small clarification needed.

$A(x) \cdot B(x) - C(x)$ isn't *equal* to $x^m - 1$, but rather is **divisible by** $x^m - 1$.

If a polynomial vanishes at all m roots of unity, those roots are roots of the polynomial, which means $(x^m - 1)$ divides it. So we can write:

$$A(x) \cdot B(x) - C(x) = H(x) \cdot Z(x)$$

where $Z(x) = x^m - 1$ is called the **vanishing polynomial**, and $H(x)$ is some quotient polynomial that the prover must compute.

This is the heart of the QAP transformation:

> **All m R1CS constraints are satisfied $\Leftrightarrow$ $Z(x)$ divides $A(x) \cdot B(x) - C(x)$ $\Leftrightarrow$ there exists a polynomial $H(x)$ such that $A(x) \cdot B(x) - C(x) = H(x) \cdot Z(x)$**

3.1.2 Concept Check 8

Suppose the prover tries to cheat: they don't actually know a valid witness, so one of the constraints fails. What happens to the polynomial $A(x) \cdot B(x) - C(x)$ at the corresponding root ω^i? And what does this mean about the divisibility by $Z(x)$?

User: then i guess with overwhelming probability (Schwartz-Zippel or something, handwave), the polynomial does not evaluate to zero at ω^i and so Z does not divide $A(x) \cdot B(x) - C(x)$

Claude: Actually even simpler — no probability needed here!

If the i-th constraint fails, then by construction:

$$A(\omega^i) \cdot B(\omega^i) - C(\omega^i) = (\vec{a}_i \cdot \vec{s}) \cdot (\vec{b}_i \cdot \vec{s}) - (\vec{c}_i \cdot \vec{s}) \neq 0$$

This is deterministic. The polynomial evaluations at the roots of unity *are* the constraint checks, by how we built the polynomials. So ω^i is definitely not a root, so $Z(x)$ definitely doesn't divide $A(x) \cdot B(x) - C(x)$, so no valid $H(x)$ exists.

Schwartz-Zippel comes in at the *next* step, which is exactly where we're headed...

4 The Verification Problem

So we've reduced "check all constraints" to "check that $Z(x)$ divides $A(x) \cdot B(x) - C(x)$".

But there's a problem: these polynomials have degree $O(m)$, where m is the number of constraints. For a real computation, m might be millions. We can't have the verifier check this equation at every point, or even receive the full polynomials — that would be linear in the computation size.

The solution: **check at a single random point** τ.

If $A(\tau) \cdot B(\tau) - C(\tau) = H(\tau) \cdot Z(\tau)$ for a randomly chosen τ, then by Schwartz-Zippel, either the polynomial equation holds everywhere, or we got extremely unlucky (probability $\approx$ degree/|field|, which is negligible).

4.0.1 Concept Check 9

But wait — if the verifier picks τ and tells it to the prover, what could a cheating prover do? Why is this a problem?

User: i'm trying to establish the exact sequence of events. the prover and verifier have agreed in advance what's to be proven (such as "i know w s.t. $w^3 = x$", with x public). this defines the $\vec{a}$, $\vec{b}$, $\vec{c}$ vectors for the m constraints (in that case only 2, but say m). the polynomials $A_j(x)$, $B_j(x)$, $C_j(x)$ are then directly constructible by interpolation. This is public to both sides. It's just the vector $\vec{s}$ that's secret (technically just some components of the $\vec{s}$-vector), and to get $A(x)$, $B(x)$, $C(x)$, you need to know those secret values. So I guess that if τ is told to the prover, they can calculate $H(\tau)Z(\tau)$ and then "doctor" the LHS by editing $\vec{s}$-vector to make $AB - C$ equal that value. Somehow?

Claude: You've got exactly the right picture of what's public vs secret, and your intuition about "doctoring" is on the right track. Let me make it precise.

If the prover knows τ in advance, they can work backwards. They don't need a valid witness at all. They can:

1. Pick arbitrary values for the secret components of $\vec{s}$ (not satisfying the constraints)
2. Compute $A(\tau)$, $B(\tau)$, $C(\tau)$ using these garbage values
3. Compute $H(\tau) = (A(\tau) \cdot B(\tau) - C(\tau))/Z(\tau)$

As long as $Z(\tau) \neq 0$, this gives them a value $H(\tau)$ that satisfies the check — even though no valid $H(x)$ polynomial exists!

The verifier only sees the *evaluated values* $A(\tau)$, $B(\tau)$, $C(\tau)$, $H(\tau)$. They can confirm the equation holds at τ, but this tells them nothing about whether the underlying polynomial identity holds everywhere.

4.1 The Solution: Commit First, Challenge Later (or: Trusted Setup)

There are two ways out:

Interactive (Fiat-Shamir): Prover commits to the polynomials first (via a hash), *then* τ is derived from the commitment. Prover can't cheat because they didn't know τ when constructing the polynomials.

Non-interactive (Trusted Setup): Someone generates τ in advance and then *destroys it.* They publish only "encrypted" versions like τP, $\tau^2 P$, etc. in the group. The prover can evaluate polynomials "in the exponent" without ever learning τ.

Groth16 takes the second approach.

4.1.1 Concept Check 10

Suppose the trusted setup publishes $\{P, \tau P, \tau^2 P, \ldots, \tau^n P\}$ in group G_1. If my polynomial is $f(x) = 3 + 2x + 5x^2$, can I compute $f(\tau)P$ without knowing τ? How?

User: Wait, before we continue, I'm a little hazy about H. IIUC we use $AB-C = HZ$ evaluated at τ as a proxy for checking that this is an identity, which we count as being significant because it proves that Z divides $AB - C$, right? So it's just the existence of H that is significant, rather than what H is, right? So does the prover just provide H as part of the proof, or ..?

Claude: Yes, this is an important point to be clear on. Let me lay it out.

You're right: the *existence* of $H(x)$ as a polynomial is what proves that the constraints are satisfied. If even one constraint fails, $A(x) \cdot B(x) - C(x)$ doesn't vanish at all roots of unity, so the division by $Z(x)$ leaves a remainder — $H(x)$ doesn't exist as a polynomial.

So here's what happens:

Honest prover:

1. Knows valid witness $\vec{s}$
2. Constructs $A(x)$, $B(x)$, $C(x)$
3. Computes $H(x) = (A(x) \cdot B(x) - C(x))/Z(x)$ — division is exact, no remainder
4. Provides $H(\tau)$ (in committed/encrypted form) as part of the proof

Cheating prover:

1. Doesn't know valid witness
2. Constructs garbage $A(x)$, $B(x)$, $C(x)$
3. Tries to divide... but there's a remainder. No polynomial $H(x)$ exists.
4. They could still *fabricate* some value and call it $H(\tau)$, but...

And here's where your earlier concern comes back: if they knew τ, they could fabricate. But they don't — they only have $\{P, \tau P, \tau^2 P, \ldots\}$.

4.1.2 Concept Check 10 (revisited)

This leads directly to the question I asked: given $\{P, \tau P, \tau^2 P, \ldots, \tau^n P\}$ and a polynomial $f(x) = 3 + 2x + 5x^2$, can you compute $f(\tau)P$ without knowing τ?

This is exactly what the prover must do with $H(x)$.

User: yes this is a trivial consequence of the homomorphism of EC point addition. To get $f(\tau)P$ I just do $3P + 2(\tau P) + 5(\tau^2 P)$ and that's $f(\tau)P$. This of course depends on the assumption that knowing $\{P, \tau P, \tau^2 P, \ldots, \tau^n P\}$ doesn't reveal info about τ which is presumably a nontrivially stronger assumption than ECDLP.

Claude: Exactly right on the computation.

Your point about security assumptions is astute. You're correct that Groth16's security relies on assumptions stronger than plain ECDLP. The main ones are:

- **q-SDH (q-Strong Diffie-Hellman):** Roughly, given $\{P, \tau P, \tau^2 P, \ldots, \tau^n P\}$, you can't produce a pair $(c, \frac{1}{\tau+c} \cdot P)$ for any c of your choosing.

- **Knowledge of Exponent (KEA):** If you output a pair $(Q, \tau Q)$ that's consistent with the pairing check $e(Q, \tau P) = e(\tau Q, P)$, you must "know" the linear combination of the setup elements you used to construct it.

These are indeed stronger than ECDLP, though they're well-studied and believed to hold in standard elliptic curve groups. Groth16 is proven secure in the **Generic Group Model**, which is a stronger idealization.

For our purposes: the key point is that the prover can evaluate polynomials at τ "in the exponent" but can't do anything else clever with it.

5 Putting It Together: What Goes Into a Groth16 Proof?

So the prover must provide, in committed form:

- $A(\tau)$, $B(\tau)$, $C(\tau)$ — derived from their witness
- $H(\tau)$ — the quotient polynomial

And the verifier checks $A(\tau) \cdot B(\tau) - C(\tau) = H(\tau) \cdot Z(\tau)$ "in the exponent" using pairings.

5.0.1 Concept Check 11

Here's a potential problem. $A(x)$, $B(x)$, $C(x)$ each have the form:

$$A(x) = \sum_j s_j \cdot A_j(x)$$

Some of the s_j are public (the 1 and the public inputs x), but some are secret (the witness w). The verifier needs to incorporate the public inputs into the check.

How might we split things so the verifier can "add in" the public input contribution themselves, while the prover only provides the secret witness contribution?

User: hmm. for each index j which is secret (so in our earlier toy example, that was 'w' at index 2), we need to give the value $s_2 \cdot A_2(\tau)$ so it can be added in to the sum to get the overall $A(\tau)$? But no, that seems not to work, since the verifier can calculate $A_2(\tau)$ (A_2 is known) and just divide to get the secret. Do we have to add some blinding then?

Claude: You're on the right track, but there's a key point to remember: the verifier never sees $A(\tau)$ as a field element. They see $A(\tau) \cdot P$ as a group element!

So the split works like this:

Trusted setup provides (among other things):

- For each *public* index j: the value $A_j(\tau) \cdot P$ in the group
- Similar for B_j, C_j

Prover provides:

- A single group element representing $\sum_{\text{secret } j} s_j \cdot A_j(\tau) \cdot P$

Verifier computes:

- Their own contribution: $\sum_{\text{public } j} s_j \cdot (A_j(\tau) \cdot P)$ — they know s_j and have $A_j(\tau) \cdot P$ from setup

- Adds (in the group) the prover's contribution
- Result: $A(\tau) \cdot P$

Now, can the verifier extract individual secret s_j values? They'd need to:

1. Isolate one term $s_j \cdot A_j(\tau) \cdot P$ from the sum (they can't — they only get the combined sum)
2. Even if they could, extract s_j from $s_j \cdot A_j(\tau) \cdot P$ (that's discrete log)

5.0.2 Concept Check 12

But you raised blinding, and that *is* still needed. Here's why: imagine a circuit with only one secret witness variable w. The prover's contribution would just be $w \cdot A_2(\tau) \cdot P$. The verifier can't compute w directly, but they could try different guesses for w and check which one matches.

So: we need randomization. In Groth16, the prover adds random multiples of δ (another secret from the trusted setup) to blind things.

Before we get into that detail — does the public/private split make sense now?

User: yes

6 The Groth16 Proof Structure

The trusted setup generates several secret scalars (the "toxic waste"):

- τ — the evaluation point we've discussed
- α, β — enforce structural constraints (more on this shortly)
- γ, δ — enable the public/private split and blinding

The final proof is remarkably compact: just **three group elements**:

- $A \in G_1$
- $B \in G_2$
- $C \in G_1$

And verification is a **single pairing equation**.

6.1 Why Do We Need α and β?

We might think we're done: prover commits to $A(\tau)$, $B(\tau)$, $C(\tau)$, $H(\tau)$, verifier checks $A(\tau) \cdot B(\tau) - C(\tau) = H(\tau) \cdot Z(\tau)$ via pairings.

But there's a subtle attack.

6.1.1 Concept Check 13

The prover is supposed to construct $A(x)$ as $\sum_j s_j \cdot A_j(x)$ — a specific linear combination of the setup polynomials, determined by their witness.

But what if the prover ignores this structure entirely? They have access to $\{P, \tau P, \tau^2 P, \ldots\}$ from the setup. What stops them from constructing some arbitrary polynomial $A'(x)$ that isn't a valid linear combination of the $A_j(x)$ polynomials, and using $A'(\tau) \cdot P$ in their proof?

User: i'm struggling here. The prover needs to make this true: $A'(\tau)\cdot B(\tau)-C(\tau) = H(\tau)\cdot Z(\tau)$ but they only have access to $\{P, \tau P, \tau^2 P, \ldots\}$. The idea of providing that instead of τ directly, is to force them to construct a polynomial "on the curve" instead of scalar values (the way you put it was they provide a *commitment* to $A(\tau)$, by which you mean $A(\tau)P$ instead of $A(\tau)$ itself). But I'm already getting lost: if the LHS is $A(\tau)B(\tau)$ and on the RHS there is $H(\tau)Z(\tau)$ then I'm confused as to how you convert these commitments into an equation check $A(\tau)\cdot B(\tau) - C(\tau) = H(\tau)\cdot Z(\tau)$, since you can't do $A(\tau)P \cdot B(\tau)P$ (there is no multiplication on curve points). Ah, is that where pairings comes in?

Claude: Exactly — this is precisely where pairings come in!

You're right: given $A(\tau)\cdot P$ and $B(\tau)\cdot P$, you cannot compute $A(\tau)\cdot B(\tau)\cdot P$. There's no multiplication on curve points.

But recall the pairing property:

$$e(aP, bQ) = e(P, Q)^{ab}$$

So if we have $A(\tau)\cdot P \in G_1$ and $B(\tau)\cdot Q \in G_2$, then:

$$e(A(\tau)\cdot P, B(\tau)\cdot Q) = e(P, Q)^{A(\tau)\cdot B(\tau)}$$

The multiplication happens "in the exponent" of the target group G_T.

6.2 The Verification Equation (Simplified)

So the verifier checks something like:

$$e(A, B) = e(C, P)\cdot e(H, Z(\tau)\cdot P)$$

where I'm being sloppy with notation, but the point is:

- Left side: pairing "multiplies" $A(\tau)$ and $B(\tau)$ in the exponent
- Right side: combines $C(\tau)$ and $H(\tau)\cdot Z(\tau)$ in the exponent
- Equality in G_T means the exponents match

This is why B lives in G_2 while A and C live in G_1 — we need one element from each group to feed into the pairing.

User: Before we get to the question, I still have confusion, specifically on your notation. You wrote: $e(A, B) = e(C, P)\cdot e(H, Z(\tau)\cdot P)$ but (1) isn't Q the generator of G_2 so why do you have P in $e(C, P)$ and (2) when you wrote $e(A, B)$ did you mean $e(A(\tau)\cdot P, B(\tau)\cdot Q)$? and (3) is $e(H, Z(\tau)\cdot P)$ actually the correct notation?

Claude: You're absolutely right to push back — I was being sloppy and it's confusing. Let me be precise.

Notation clarification:

- P is the generator of G_1
- Q is the generator of G_2
- A pairing requires one element from each group: $e(\cdot,\cdot) : G_1 \times G_2 \to G_T$

So:

(1) You're right: $e(C, P)$ is nonsense if both are in G_1. It should involve Q.

(2) Yes, when I wrote $e(A, B)$, I was conflating the proof elements with the polynomials. To be precise: the proof element $A \in G_1$ encodes something like $A(\tau) \cdot P$, and proof element $B \in G_2$ encodes something like $B(\tau) \cdot Q$. So $e(A, B)$ as a pairing is well-typed.

(3) $e(H, Z(\tau) \cdot P)$ is wrong — both arguments would be in G_1. It should be something like $e(H(\tau) \cdot P, Z(\tau) \cdot Q)$, which requires the setup to provide $Z(\tau) \cdot Q \in G_2$.

A cleaner version of the check:

We want to verify $A(\tau) \cdot B(\tau) = C(\tau) + H(\tau) \cdot Z(\tau)$ in the exponent.

Given:

- $A(\tau) \cdot P \in G_1$
- $B(\tau) \cdot Q \in G_2$
- $C(\tau) \cdot P \in G_1$
- $H(\tau) \cdot P \in G_1$
- $Z(\tau) \cdot Q \in G_2$ (from setup, since $Z(\tau)$ is public)

The verifier checks:

$$e(A(\tau) \cdot P,\ B(\tau) \cdot Q) = e(C(\tau) \cdot P + H(\tau) \cdot Z(\tau) \cdot P,\ Q)$$

Left side: $e(P, Q)^{A(\tau) \cdot B(\tau)}$

Right side: $e(P, Q)^{C(\tau) + H(\tau) \cdot Z(\tau)}$

Equality holds iff $A(\tau) \cdot B(\tau) = C(\tau) + H(\tau) \cdot Z(\tau)$.

Actually, we can split the right side differently:

$$e(A(\tau) \cdot P,\ B(\tau) \cdot Q) = e(C(\tau) \cdot P,\ Q) \cdot e(H(\tau) \cdot P,\ Z(\tau) \cdot Q)$$

This uses bilinearity: $e(X + Y, Q) = e(X, Q) \cdot e(Y, Q)$.

Does this clarify the structure? Then we can return to concept check 13.

User: yes. back to Q13. rewrite it if necessary, otherwise I'll just keep thinking.

Claude: The question stands as is. Take your time.

User: Can we somehow make $A(\tau) \cdot B(\tau) = C(\tau) + H(\tau) \cdot Z(\tau)$ be true by choosing adversarially A and B such that there is cancellation? Like e.g. making $A = HZ - C/B$; here, division is possible because they're polynomials I'm choosing, and this turns it into an identity that is true for all inputs, so it's true for τ even though I don't know τ.

Claude: Yes! This is exactly the attack.

If there's no structural constraint, the prover can simply:

1. Pick arbitrary polynomials $A'(x)$, $B'(x)$
2. Compute $A'(x) \cdot B'(x)$, divide by $Z(x)$, get quotient $H'(x)$ and remainder $C'(x)$

3. Or even simpler: set $C'(x) = A'(x) \cdot B'(x) - H'(x) \cdot Z(x)$ for any $H'(x)$ you like

The equation $A'(\tau) \cdot B'(\tau) = C'(\tau) + H'(\tau) \cdot Z(\tau)$ holds because the underlying polynomial identity holds everywhere — no knowledge of τ required.

This "proof" is meaningless: it has nothing to do with any witness or any R1CS constraints.

6.3 The Fix: α and β

We need to force the prover to use the *specific* polynomials $A_j(x)$, $B_j(x)$, $C_j(x)$ from the constraint system.

The trusted setup introduces secrets α and β, and provides:

- $\alpha P \in G_1$
- $\beta Q \in G_2$
- For each j: the element $(\beta \cdot A_j(\tau) + \alpha \cdot B_j(\tau) + C_j(\tau)) \cdot P$

The prover must construct their proof using these "entangled" elements. The verification equation then includes terms like $e(A, \beta Q)$ and $e(\alpha P, B)$ that enforce consistency.

6.3.1 Concept Check 14

Here's the intuition: suppose the verification checks (among other things):

$$e(A, \beta Q) \cdot e(\alpha P, B) = e(\text{something well-formed}, Q)$$

If the prover constructs A honestly as $\sum_j s_j \cdot A_j(\tau) \cdot P$, what does $e(A, \beta Q)$ evaluate to in terms of the exponent?

And why can't a cheating prover who constructs $A'(\tau) \cdot P$ from arbitrary coefficients satisfy this check?

User: First part: the exponent gets from $e(A, \beta Q)$ the term $\sum_j s_j \cdot \beta A_j(\tau)$ .. but to relate that to the element $(\beta \cdot A_j(\tau) + \alpha \cdot B_j(\tau) + C_j(\tau)) \cdot P$ you'd have to combine it with the other terms, right?

Claude: Exactly right. Let me show how the combination works.

If the prover honestly computes:

- $A = \sum_j s_j \cdot A_j(\tau) \cdot P$
- $B = \sum_j s_j \cdot B_j(\tau) \cdot Q$

Then:

- $e(A, \beta Q)$ gives exponent $\beta \cdot \sum_j s_j \cdot A_j(\tau) = \beta \cdot A(\tau)$
- $e(\alpha P, B)$ gives exponent $\alpha \cdot \sum_j s_j \cdot B_j(\tau) = \alpha \cdot B(\tau)$

Combined: $\beta \cdot A(\tau) + \alpha \cdot B(\tau)$

Now the setup also provides the combined elements $(\beta \cdot A_j(\tau) + \alpha \cdot B_j(\tau) + C_j(\tau)) \cdot P$ for each j.

The prover can compute:

$$\sum_j s_j \cdot (\beta \cdot A_j(\tau) + \alpha \cdot B_j(\tau) + C_j(\tau)) \cdot P = (\beta \cdot A(\tau) + \alpha \cdot B(\tau) + C(\tau)) \cdot P$$

So the verification can check that $e(A, \beta Q) \cdot e(\alpha P, B)$ (giving $\beta \cdot A(\tau) + \alpha \cdot B(\tau)$ in exponent) is consistent with the prover's combined element (which has $\beta \cdot A(\tau) + \alpha \cdot B(\tau) + C(\tau)$).

6.3.2 Why Can't a Cheater Fake This?

Suppose the cheater constructs $A' = A'(\tau) \cdot P$ from arbitrary polynomial coefficients, not using the $A_j(\tau) \cdot P$ elements.

They need $e(A', \beta Q)$ to give the right contribution. That means they need $\beta \cdot A'(\tau)$ in the exponent.

But the only way to get β into an exponent is by pairing with βQ — and they don't control what $A'(\tau)$ pairs with β to produce. They'd need their arbitrary $A'(\tau)$ to somehow match $\sum_j s_j \cdot A_j(\tau)$ for some valid witness.

They can't "mix in" extra β from elsewhere because they only have $\beta Q \in G_2$, not β as a scalar or $\beta P \in G_1$.

6.3.3 Concept Check 15

This is subtle, so let me ask it directly: the cheater has access to $\{\tau^i P\}$ and can construct any polynomial evaluation $f(\tau) \cdot P$. They also have βQ. Why can't they compute $\beta \cdot f(\tau) \cdot P$ for their arbitrary f?

User: Because they don't have β? β is secret, only βQ is known and that would be a DLOG break.

Claude: Exactly. They have $\beta Q \in G_2$, but to compute $\beta \cdot f(\tau) \cdot P \in G_1$ they'd need either:

- β as a scalar (requires breaking DLOG on βQ), or
- $\beta P \in G_1$ (not provided by setup — this is deliberate)

The asymmetry is crucial: β only appears in G_2 via βQ, so the only way to get β into an exponent alongside G_1 elements is through a pairing. And pairings only give you products of what you already have.

This is the "knowledge binding" — the algebraic structure forces the prover to use the provided elements in the intended way.

7 The Full Picture: γ and δ

We've covered:

- τ — the secret evaluation point
- α, β — enforce that prover uses the correct polynomial structure

Now γ and δ handle:

- **Public/private split**: separating public inputs from secret witness
- **Zero-knowledge**: randomizing the proof so it reveals nothing about the witness

The setup provides:

- Public input terms divided by γ: things like $(\beta \cdot A_j(\tau) + \alpha \cdot B_j(\tau) + C_j(\tau))/\gamma \cdot P$ for public indices

- Witness terms divided by δ: same form but with δ for secret indices
- γQ and δQ in G_2

The prover adds random multiples of δ to blind the proof.

7.0.1 Concept Check 16

Recall our earlier concern: if the circuit has just one secret variable w, the prover's contribution is essentially $w \cdot (\text{stuff}) \cdot P$, and a verifier could try guessing w.

If the prover adds a random $r \cdot \delta \cdot P$ to their proof element, why does this prevent the guessing attack? And why doesn't it break the verification equation?

User: wait, what is "r" in that last expression? I don't see it referred to earlier.

Claude: You're right, I introduced it without explanation.

r is a random scalar that the **prover** chooses freshly when generating the proof. It's not from the trusted setup — it's the prover's own randomness.

So the prover picks random r (and actually also a second random value s), and uses these to randomize the proof elements. The setup provides $\delta P \in G_1$ and $\delta Q \in G_2$ so the prover can compute things like $r \cdot \delta P$ without knowing δ itself.

Now the question: why does adding $r \cdot \delta P$ to a proof element prevent guessing attacks, and why doesn't it break verification?

User: Not very sure. Here's what I have: if as prover I have just w at index j, I have to provide $w \cdot A_j(\tau)P$ and as you say, since the verifier can calculate $A_j(\tau)P$ he can guess values of w until he matches my message. If I "add $r \cdot \delta P$" then I am sending $w \cdot A_j(\tau)P + r \cdot \delta P$ which clearly blinds it but I don't really see how the verifier can check; he has δP but i don't see how he deals with the r, unless somehow the prover commits to it (e.g. $r \cdot Q$, and then the pairing can use it to verify? ....)

Claude: You're on the right track. The key insight is that r appears in *multiple* proof elements in a coordinated way, so that when the pairing equation is checked, the random terms cancel.

Here's the structure. The proof is three elements (A, B, C), and the prover uses *two* random values r and s:

Proof element A (in G_1):

$$A = \alpha P + A(\tau)P + r \cdot \delta P$$

Proof element B (in G_2):

$$B = \beta Q + B(\tau)Q + s \cdot \delta Q$$

Proof element C (in G_1):

$$C = (\text{witness terms}) + s \cdot A + r \cdot B' - rs \cdot \delta P$$

where B' is the G_1 version of the B polynomial part.

7.0.2 Concept Check 17

When we compute $e(A, B)$, we're multiplying (in the exponent):

$$(\alpha + A(\tau) + r\delta) \cdot (\beta + B(\tau) + s\delta)$$

Expand this product. Which terms involve the randomness r and s?

And then: looking at the structure of C, can you see how those random terms might cancel when we check $e(A, B) = e(C, \ldots) \cdot$ (other pairings)?

User: There are 5 terms which involve the randomness r and s. They are: $\delta(r\beta + rB + sA + \alpha s + rs\delta)$ where I have factored out δ and dropped the tau dependency for brevity. I can't easily see how this cancels when combined with C.

Claude: Your expansion is correct. Let me write it out fully:

$$\begin{aligned}&(\alpha + A(\tau) + r\delta)(\beta + B(\tau) + s\delta)\\ &= \alpha\beta + \alpha B(\tau) + A(\tau)\beta + A(\tau)B(\tau) + \delta(\alpha s + sA(\tau) + r\beta + rB(\tau) + rs\delta)\end{aligned}$$

The "clean" terms are: $\alpha\beta + \alpha B(\tau) + A(\tau)\beta + A(\tau)B(\tau)$

The "random" terms are: $\delta(\alpha s + sA(\tau) + r\beta + rB(\tau) + rs\delta)$

Now, the Groth16 verification equation has the form:

$$e(A, B) = e(\alpha P, \beta Q) \cdot e(L, \gamma Q) \cdot e(C, \delta Q)$$

where L is the public input contribution.

The key: $e(C, \delta Q)$ produces $C_{\text{exponent}} \cdot \delta$ in the exponent of G_T.

So if the prover constructs C such that its exponent equals:

$$C_{\text{exponent}} = \frac{\alpha B(\tau) + A(\tau)\beta + A(\tau)B(\tau) - C(\tau)}{\delta} + (\alpha s + sA(\tau) + r\beta + rB(\tau) + rs\delta)$$

then $e(C, \delta Q)$ contributes exactly $\alpha B(\tau) + A(\tau)\beta + A(\tau)B(\tau) - C(\tau) + \delta$(random terms).

Combined with $e(\alpha P, \beta Q) = \alpha\beta$, the verification equation balances.

The crucial point: the division by δ in the first part of C_{exponent} is only possible because the **trusted setup** provides elements like $(\beta \cdot A_j(\tau) + \alpha \cdot B_j(\tau) + C_j(\tau))/\delta \cdot P$. The prover never knows δ — they just combine these pre-divided elements.

The random terms $(\alpha s + sA(\tau) + \ldots)$ don't need division by δ because they already have a δ factor that will cancel when paired with δQ.

7.0.3 Concept Check 18

The prover can add the random terms because they're constructed from:

- $s \cdot A$ where A (the proof element) has exponent $\alpha + A(\tau) + r\delta$
- $r \cdot (B(\tau)P)$ from the setup
- $-rs \cdot \delta P$ from the setup

They never need δ as a scalar. Does it make sense why each of these is computable by the prover?

User: Mostly, kinda. Just to clarify: the prover chooses r and s but doesn't have to communicate even a commitment to them, right? Because they cancel within verification? Because he's added the necessary expressions involving them into his C? Btw at this point can you summarise again the roles of each of alpha, beta, gamma, delta as it's hard to keep track.

Claude: Correct. The prover chooses r and s fresh, uses them to construct A, B, C, and never communicates them. The verifier just checks the pairing equation — the random terms cancel internally without the verifier ever knowing r or s.

This is what makes it zero-knowledge: each proof is randomized differently, so two proofs for the same witness look completely unrelated.

7.1 Summary of the Secret Scalars

τ **(tau)** — *Evaluation point*

- Polynomials are evaluated at τ to compress constraint checking
- Prover only has $\{P, \tau P, \tau^2 P, \ldots\}$, never τ itself
- This forces honest polynomial construction (can't "work backwards" from a known point)

α, β **(alpha, beta)** — *Structural enforcement*

- Force the prover to use the specific $A_j(x)$, $B_j(x)$, $C_j(x)$ from the R1CS
- Setup provides "entangled" elements like $(\beta \cdot A_j(\tau) + \alpha \cdot B_j(\tau) + C_j(\tau)) \cdot P$
- Verification checks $e(A, \beta Q) \cdot e(\alpha P, B)$ to ensure consistency
- Without these, prover could use arbitrary polynomials satisfying $AB - C = HZ$

γ **(gamma)** — *Public input separation*

- Public input terms are provided as $(\ldots)/\gamma \cdot P$
- Verification includes $e(L, \gamma Q)$ where L is the public input contribution
- Allows verifier to "plug in" public inputs themselves

δ **(delta)** — *Witness separation and zero-knowledge*

- Secret witness terms are provided as $(\ldots)/\delta \cdot P$
- Prover's randomness r, s multiply δ terms
- Verification includes $e(C, \delta Q)$
- Enables blinding without breaking the equation

8 Security: What If the Toxic Waste Leaks?

8.0.1 Concept Check 19

Here's a unifying question. The trusted setup provides many elements, but after setup is complete, the secrets $\tau, \alpha, \beta, \gamma, \delta$ are destroyed ("toxic waste").

Why is it catastrophic if someone retains δ? What could they do?

User: If someone retains delta, I think it specifically removes the ZK property .. the trusted setup issues terms like $(\beta \cdot A_j(\tau) + \alpha \cdot B_j(\tau) + C_j(\tau))/\delta \cdot P$, so if you multiply them by delta you could perhaps guess witness terms from things like $w \cdot A_j(\tau)P$ in the way we earlier described ... sorry I'm not quite there, this seems very complex.

Claude: You're right that it's complex. Let me reframe — your intuition is pointing somewhere, but the main catastrophe is actually different.

8.1 The Real Danger: Forgery, Not De-anonymization

If someone retains δ, they can **forge proofs for false statements**.

Here's why. The honest prover constructs C using pre-computed elements from setup:

$$\frac{\beta \cdot A_j(\tau) + \alpha \cdot B_j(\tau) + C_j(\tau)}{\delta} \cdot P$$

They can only form linear combinations of these — they're "locked in" to using valid witness values because they don't know δ.

But if you know δ as a scalar:

1. You can compute any polynomial $f(\tau) \cdot P$ using $\{P, \tau P, \tau^2 P, \ldots\}$
2. You can divide by δ yourself: compute $(1/\delta) \cdot f(\tau) \cdot P$
3. You can construct a valid-looking C for a **non-existent witness**

You can prove "I know w such that $w^3 = x$" without knowing any such w.

8.2 ZK vs Soundness

There are two distinct security properties:

Zero-knowledge: Verifier learns nothing about the witness from the proof. → Protected by prover's randomness r, s. The verifier doesn't have δ anyway.

Soundness: Cheater can't prove false statements. → Protected by $\tau, \alpha, \beta, \gamma, \delta$ being unknown. If any leak, forgery may be possible.

Retaining δ breaks **soundness**, not zero-knowledge.

8.2.1 Concept Check 20

Similarly: what goes wrong if someone retains τ?

User: I suspect the tau case is easier to follow/more direct. Tau was introduced as an "unexpected" sampling of the polynomials, such that the prover cannot choose random polynomials that happen to fit $AB - C = HZ$ at that particular point. A more natural way of doing that in similar but simpler cryptosystems is to just use the Fiat Shamir transform, but presumably here that is not viable since we need a bunch of conditions and not only this one. But indeed if you knew tau, you could do this and make completely invalid proofs.

Claude: Exactly right.

Knowing τ is the most direct attack. You can:

1. Pick garbage values for your "witness"

2. Compute $A(\tau)$, $B(\tau)$, $C(\tau)$ as field elements (not group elements)
3. Compute $H(\tau) = (A(\tau) \cdot B(\tau) - C(\tau))/Z(\tau)$ directly
4. Construct the proof elements to satisfy the verification equation

The polynomial identity $A(x) \cdot B(x) - C(x) = H(x) \cdot Z(x)$ doesn't hold — but who cares? You only needed it to hold at τ, and you made sure it does.

Your Fiat-Shamir intuition is sharp. In systems like **STARKs** or **Plonk**, the prover first commits to their polynomials (e.g., via Merkle trees or polynomial commitments), and then τ is derived by hashing the commitment. The prover can't cheat because they didn't know τ when they committed.

Groth16 takes the trusted setup route instead because:

- It gives **constant-size proofs** (just 3 group elements, ~200 bytes)
- Verification is **one pairing equation** (very fast)
- The cost: you need a ceremony to generate and destroy the toxic waste

Other systems avoid trusted setup but pay in proof size or verification time.

We've covered the core of Groth16. Where would you like to go next?

- **Worked example**: trace through a tiny circuit end-to-end
- **Trusted setup ceremonies**: how do you actually generate and destroy toxic waste with multiple parties?
- **Comparison**: how do STARKs/Plonk differ architecturally?
- **Applications**: how does this get used in practice (e.g., Zcash, rollups)?

User: I didn't understand the attack from knowing delta. Can you work through it in a bit more detail.

Claude: Let me work through a concrete example.

8.3 Setup Elements for the H Term

Recall the verification needs to check that $A(\tau) \cdot B(\tau) - C(\tau) = H(\tau) \cdot Z(\tau)$.

The $H(\tau) \cdot Z(\tau)/\delta$ contribution to C comes from setup elements. Specifically, the setup provides:

$$\left(\frac{Z(\tau) \cdot \tau^i}{\delta}\right) \cdot P \quad \text{for } i = 0, 1, \ldots, d$$

where d is the maximum degree of H.

An honest prover with $H(x) = h_0 + h_1 x + h_2 x^2 + \ldots$ computes:

$$\sum_i h_i \cdot \left(\frac{Z(\tau) \cdot \tau^i}{\delta}\right) \cdot P = \frac{H(\tau) \cdot Z(\tau)}{\delta} \cdot P$$

8.4 Why the Honest Prover is Constrained

The honest prover can only construct $(f(\tau)/\delta) \cdot P$ for polynomials of the form $f(x) = H(x) \cdot Z(x)$.

Why? Because every setup element they have access to contains $Z(\tau)$ in the numerator. Any linear combination of $\{Z(\tau) \cdot \tau^i/\delta \cdot P\}$ gives $(H(\tau) \cdot Z(\tau)/\delta) \cdot P$ for some H.

They cannot construct $(g(\tau)/\delta) \cdot P$ for arbitrary $g(\tau)$ that doesn't factor through $Z(\tau)$.

8.5 The Attack with Known δ

If you know δ as a scalar, you have:

1. $\{P, \tau P, \tau^2 P, \ldots\}$ from setup — lets you compute $f(\tau) \cdot P$ for any polynomial f
2. δ as a scalar — lets you compute $(1/\delta) \cdot Q$ for any group element Q

Combined: you can compute $(f(\tau)/\delta) \cdot P$ for **any** polynomial $f(x)$, not just those divisible by $Z(x)$.

8.6 Concrete Forgery

Say you want to prove "I know w such that $w^3 = 7$" but no such w exists in your field.

Honest attempt (fails):

1. You try some garbage witness
2. You compute $A(\tau) \cdot B(\tau) - C(\tau)$ — call it $E(\tau)$ for "error"
3. You try to divide: $H(x) = E(x)/Z(x)$ — but there's a remainder
4. No polynomial $H(x)$ exists, so you can't construct $(H(\tau) \cdot Z(\tau)/\delta) \cdot P$
5. Proof fails

With known δ:

1. Use same garbage witness, get same error $E(\tau)$
2. Compute $E(\tau) \cdot P$ using $\{\tau^i P\}$
3. Compute $(E(\tau)/\delta) \cdot P = (1/\delta) \cdot (E(\tau) \cdot P)$ — you know δ, so this is just scalar multiplication
4. Use this as your "H contribution" in C
5. Verification passes because the equation balances at τ

You never needed $E(x)$ to be divisible by $Z(x)$. You just needed $E(\tau)/\delta$ in the group, and knowing δ gave you that directly.

Does this make the attack clearer?