Ideas behind Bulletproofs

August 28, 2025

1 Overview

1.1 Steps

- Background: what is a ZKP and what is a POK?
- NUMS generators
- Committing to a vector
- Committing to two vectors and an inner product
- The Bulletproofs IPA algorithm
- Using the IPA to prove that a value is in range
- Halo and amortizing verification
- Using an IPA as a Polynomial Commitment Scheme
- Further areas of interest

1.2 Notation

I am going to use additive notation in this document, so that a public key curve point P is related to the corresponding private key x with $P = xG$, and further I'm going to assume only using elliptic curve groups, so capital letters Q always refer to elliptic curve points and lower case letters q always refer to scalars in Z_p where p is the order of the group (assume p is prime, though that won't be relevant in this study).

Translating between additive and multiplicative notation, as you will have to do in reading the papers, is a useful skill to pick up.

1.2.1 Relations and Languages

This is useful to know for reading academic papers going forward, if nothing else. A **relation**, more specifically a binary relation, is written as $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^*$ whose output value is true or false (or 0/1); not a very interesting definition in itself, but taken as the basis for defining what's called a **language**, which is some (often very large) set of *statements*, call them x, think of them as the first of the two strings in the above $\mathcal{R}$, which can be proven to be *in the language*; we

state this formally as: $\mathcal{L} = \{x \mid \exists\ w \text{ s.t. } \mathcal{R}(x, w) = 1\}$. Here w is the **witness** because it attests to the truthiness of the claim that the statement x is in the language $\mathcal{L}$. Note w doesn't always have to be secret. Note also that for obvious reasons we care if $\mathcal{R}$ is efficiently computable.

1.2.2 Reader prompts

In the remainder I'll use Question for things you should answer quickly in a sentence or equation or two, and Task for something meatier which might require writing out a large(r) number of steps. Underline is your cue to do something other than just read references.

2 ZKP fundamentals

2.1 What is a ZKP and what is POK?

A little side discursion before we get going, because we are going to be examining instantiations of both of these ideas; they are connected, but quite distinct.

> Task: For a truly excellent introduction to understanding ZKP fundamentals, please watch the first hour (approximately) of this video presentation by Shaffi Goldwasser. Hat tip Jonas Nick for suggesting this particular video! While this is a long time, it will allow you to skip or skim the remainder of this section :)

Proofs of Knowledge are proofs of knowing a witness w for the fact that $x \in \mathcal{L}$. You will often see "arguments" of knowledge instead, which is a term of art just meaning that the "proof" is computational, that is, it is possible to produce an argument of knowledge without actually knowing the witness, but we argue that the probability of achieving this is negligible given certain assumptions about the computational power of the adversarial prover. In the remainder, I'll always use "POK" even where "AOK" would be technically correct.

Zero Knowledge Proofs (or ZKPs) are proofs for which three specific properties hold:

- Completeness
- Soundness
- Zero-knowledgeness

Let's speak crudely. Consider that there are two parties, the Prover and the Verifier. From the point of view of the Prover:

- **COMPLETENESS**: If you're honest, which is to say, the statement is true, you *can* prove it.
- **SOUNDNESS**: If you're not honest, which is to say, the statement is false, you *cannot* produce a proof which the verifier accepts.

- **ZERO-KNOWLEDGENESS**: If you complete the protocol honestly, you do not leak any additional information to the verifier except that the statement is true.

The COMPLETENESS property should always be trivial: the protocol is nonsense if an honest prover can't produce a proof that verifies. Papers will always briefly cover this though, to allow the reader to understand how the verification algorithm works.

The SOUNDNESS property is related to, but not the same as, what we've been seeing already in this course: how "reductions" can essentially isolate the prover, here treated like an adversary, and from its ability to create a validating proof, extract some other useful thing. *Often* it is able to extract a secret witness (such as a private key); in that case we are in fact dealing with a ZKP which is also a POK (often you'll see ZkPoK as the shorthand).

But in less common scenarios, this is not the case: I expand on this briefly below:

2.1.1 Are ZKPs also POKs?

Do POKs always accompany ZKPs?
Question: Consider that the Prover is treated as an algorithm, or a computer program. What could it possibly mean for an algorithm to "know" something? Can you come up with a definition that might make sense? (Answer is spoiled below!)

If the protocol *is* a POK, it must be accompanied by an algorithm called an **extractor**. A large majority of examples you might have heard about, of ZK, obviously require the prover to *know* a secret witness. A Schnorr identification protocol has a zero-knowledgeness property (we'll talk about why in a moment), and it is a proof of knowledge (though, with a caveat; do you see why?). But proving it sound, means exactly constructing an extractor for the witness (the dlog or private key). Other typical examples might be: proof that a graph can be 3-colored, which is done by actually *knowing* a valid 3-coloring, or a similar case might be solving a Sudoku. In all these cases proving soundness means demonstrating an extractor algorithm to extract the secret witness.

But you can imagine special types of ZKPs that do not require *knowledge* of a witness. In this case soundness proof must be based on a (computational, at least) impossibility result. A concrete example is certain RSA accumulator schemes, in which you can make a ZKP of membership of a group, but the witness for the truth of the statement is exponentially large. So instead of proving soundness by creating an extractor, you prove that creating a validating proof of a false statement would imply the ability to factor the RSA modulus $n = pq$ (whereas in these cases, "hidden order" groups, this is not computationally feasible).

Remember: if you have a POK, you have soundness (using the extractor). But if your ZKP has soundness, it doesn't necessarily imply there is any POK, or extractor.

Finally, let's talk about the third property of ZKPs, ZERO-KNOWLEDGENESS. This is the genius insight, and one that has been rather spoiled for you; you've used it already! Taking a list the public transcripts of the conversation between the Prover and Verifier, in successful runs. We define an algorithm called a **simulator**: its task is to output a list of transcripts that are *statistically indistinguishable* from the previously mentioned list of *real* transcripts for successful runs. If it can do this, it follows logically that the Verifier has learned nothing from the successful runs, other than that the statement is true.

The ability to create such transcripts might seem like magic, but it depends on a crucial difference between the simulation scenario, and the real proof scenario: **interactivity**. The Prover is forced to take some (usually random) input from the verifier during the protocol execution, but the simulator has no such limitation, and it exploits this to achieve its goal. In the case of a Schnorr identification protocol, this means that the Simulator *first* generates the challenge value e, and only then chooses/calculates the response value s and then the commitment R. By doing so he can easily achieve the required statistical indistinguishability between his (R, e, s) values and those of the prover.

3 Preparation for Bulletproofs

3.1 Generators

Question: Why is a Pedersen commitment like $C = aG + bH$ binding? Why is it hiding?

Question: Why is it *not* binding, if we know x s.t. $H = xG$?

Question: Continuing as previous (that we know such x), is it still hiding? Justify both your answers.

Food for thought: There is some controversy over whether secp256k1's standard generator G was chosen "honestly". In part of our online prep course we discussed a property that can ameliorate this concern; what is it called?

3.2 NUMS

NUMS - nothing up my sleeve. Proving that although x, the preimage of H with respect to G (the standardized curve generator), exists, by definition, you do not know it, and anyone can verify this proof. This is a slippery area; it depends on some credibility of irreversibility.

If I choose a point H which is the output of a function $f : a \in \mathbb{Z}_q \rightarrow \mathbb{G}$ then we have two problems to address: the function f must map in some sense randomly into the elliptic curve group $\mathbb{G}$; the function f must be computationally irreversible in a strong sense.

Question: If f were not irreversible, what attack would that allow on Pedersen commitments using generators created from it?

Question: is it important that the input a is not a purely random string? Can it be anything?

Tougher question, optional: what if we created 256 generators $G_1, \ldots G_{256}$ by using a hash-to-curve function f which was well behaved (preimage resistant). Couldn't we use a variant of Wagner's attack to find at least one relative discrete log? For example $f(s_1) + \ldots f(s_{256}) = 0$ where s_n are the seed values put into the hashes for the generators; doesn't this fit exactly the structure of Wagner's k-sum attack, and thus you at least have a discrete log relation between say 255 of the generators and the last one, breaking the requirement for these commitments, that no relative discrete logs are known?

3.3 Vector Pedersen Commitments

Consider that you know a vector of values in $\mathbb{Z}_p$ of length n, call it $\underline{\mathbf{v}}$.

Question: How can you make a Pedersen-style commitment to $\underline{\mathbf{v}}$ with only *one* group element C, that is both perfectly hiding and computationally binding? Justify your answer.

Question: Justify with mathematical notation the statement: "Vector Pedersen commitments are additively homomorphic" (where "addition" is of course point addition).

3.3.1 Multi-vector commitments

Question: Does anything change if we want to commit to 2 vectors at once, i.e. can we commit also to $\underline{\mathbf{v}}, \underline{\mathbf{w}}$ in one group element C? How, if at all, must the construction change?

3.4 Proving knowledge of vector Pedersen commitments

In the online section of the course, we analyzed a special case of proving knowledge of the opening of a Pedersen commitment. Now let's do it in generality. Remember that the basic paradigm for these Σ protocols is:

- Create a one-time ephemeral instance of the "language" and send it to the verifier
- Receive a challenge from the verifier (or Fiat-Shamir for the non-interactive case)
- output the linear combination of your secret witness with the secret witness for the one-time instance from step 1 and send that as the response
- ... and note how verification is possible without revealing the secret witness

> Task: Implement this paradigm (at least, on paper) for a (non-vector) Pedersen commitment to a single secret value x, with randomisation r.

Question: Write the alteration to your solution above, for a *vector* Pedersen commitment. Do you see a problem, specifically, with doing this protocol for a vector of values in $\mathbb{Z}_p$, of very large dimension, $n > 100$ e.g.?

3.5 Making the proof compact - cut and fold

Reflecting on what we've just done, let's ask, can we make the proof compact, at least, sublinear in the vector dimension? Let's forget the idea of *blinding* or zero knowledge, for a moment, and focus only on how much data we use to transmit the witness for the claim that $C = \underline{\mathbf{a}} \cdot \underline{\mathbf{G}}$. We could simply directly send across $\underline{\mathbf{a}}$, but obviously that did not fix it - it is still linear size in n.

It's quite a deep puzzle to ask, how could we send less? The key insight is, as far as I know, originally from a paper of Bootle in 2016 (see paper), although I wouldn't be amazed to find something earlier. The first version of the Bulletproofs paper had a nice way of looking at it which I paraphrase here, in my old analysis:

> ... start by considering a single vector ... but wait, in (the Bulletproofs paper), Bünz goes back (helpfully!) even a step further and considers just committing to 2 scalars a, b. Let's say we commit to them with a commitment $C = aG_1 + bG_2$ (as in the previous section we are omitting blinding). If you wanted to fold this commitment together so as to reveal only *one* scalar in the commitment opening, you'd need to somehow combine a and b together. As we've already observed at least once, "combining" values under commitment in a naive way loses the binding property – a commitment to $a + b$ is useless as it might just as easily be a commitment to $(a + \alpha), (b - \alpha)$ as to (a, b). A commitment to something like $(ax + b)$, with x being the challenge as per usual, seems like a step up – but how is the

verifier going to verify the commitment? $C(ax+b) = xC(a) + C(b)$ by linearity, but that is not a function of the original commitment C. We need a function $f(a, b, x)$ from these three values to a single scalar a', which, when combined with a function $g(G_1, G_2, x)$ from the basepoints and x to a new basepoint G', such that we can construct a commitment verifier-side. This construction is:

$$
\begin{aligned}
&a' = ax + bx^{-1}, \quad G' = x^{-1}G_1 + xG_2 \\
&\therefore C' = a'G' = \left(ax + bx^{-1}\right)\left(x^{-1}G_1 + xG_2\right) \\
&= aG_1 + bG_2 + x^2 aG_2 + x^{-2}bG_1 = C + x^2 L + x^{-2} R
\end{aligned}
$$

where C was the original commitment.

Note, in that, that the points L and R are not calculable by the verifier - they have to be provided by the prover, which is a big loss for this protocol for proving 2 variables. But notice, we did win *something* - we only transferred one scalar a', not two (a, b). Our trick is that we can **recurse** this protocol, as per the following:

Now we have consider again $C = \underline{\mathbf{a}} \cdot \underline{\mathbf{G}}$. For simplicity assume its length is 16. We simply cut the vector $\underline{\mathbf{a}}$ in half, two vectors $\underline{\mathbf{a}}_L$ and $\underline{\mathbf{a}}_R$ each of length 8. We do the same with our vector of generators: $\underline{\mathbf{G}}_L, \underline{\mathbf{a}}_R$. After we receive the challenge value x, we construct as per above $\underline{\mathbf{a}}' = (x\underline{\mathbf{a}}_L + x^{-1}\underline{\mathbf{a}}_L)$ and: $\underline{\mathbf{G}}' = (x^{-1}\underline{\mathbf{G}}_L + x\underline{\mathbf{G}}_R)$.

This is algebraically no different at all to the (a, b) case above. As then, we have halved the length of the vectors; $\underline{\mathbf{a}}'$ has length 8, and the same for the generators. And $\underline{\mathbf{a}}' \cdot \underline{\mathbf{G}}' = C + x^2 L + x^{-2} R$, if we define $L = x^2(\underline{\mathbf{a}}_L \cdot \underline{\mathbf{G}}_R)$ and $R = x^{-2}(\underline{\mathbf{a}}_R \cdot \underline{\mathbf{G}}_L)$. As before, we do have to send L, R to the verifier - but they are *always* one point each, no matter how big the vector!

3.5.1 Recursion

Notice how at the end of that process you have a new point $C' = \underline{\mathbf{a}}' \cdot \underline{\mathbf{G}}' = C + x^2 L + x^{-2} R$, so $(C', \underline{\mathbf{a}}')$ represent a new instance of the same problem we started with, of half the dimension. So we can recurse, at each step of the recursion throwing off a pair of points L_i, R_i. At the end of your recursion you have a vector of length 1, so you just send the individual scalar.

> Task: Justify to yourself that you understand this protocol by writing it either on paper or in Python or Rust or something, and by doing so, calculate the number of bytes that the Prover needs to transfer to the Verifier (in this interactive protocol), for our example in which $\underline{\mathbf{a}}$ has 16 elements each of which are, say, 32 bytes and the group elements are also 32 bytes.

Help with the task: it's important to understand this aspect of the interaction: the Prover, at each step of recursion, will be sending values L_i and R_i, but not (wastefully) also sending C_i, i.e. the commitment at each step, because

the Verifier, given the corresponding challenge value x_i, can calculate $C_i = C_{i-1} + x_{i-1}^2 L_i + x_{i-1}^{-2} R_i$ (don't sweat the exact notation of indices i here; I suggest writing out the protocol step by step on paper with arrows/rounds). At each round there is an x from the Verifier and then a L, R pair from the Prover, until the end when the Prover just transmits a scalar. Also all these xs can just be random challenge values, or, if you want to construct the non-interactive version (a bit fiddly to do correctly) you should make each x a function of the whole of the conversation transcript up to that point.

3.5.2 Soundness

As we've mentioned, there is no pretense to zero-knowledgeness for these algorithms (we will touch on that, soon). But indeed there is no point having a super-compact proof of knowledge of the opening of C to a vector a, which does not have soundness!

As per earlier discussions, this *is* intended to be a POK, so we want to build an **extractor**, that successfully extracts the vector $\underline{\mathbf{a}}$. This is quite a different scenario to what you've seen before, although you still want to use something like "forking". Hence the "Help" section below.

Task: Attempt to build such an extractor that demonstrates that any validating proof can only be produced by a knower of the vector $\underline{\mathbf{a}}$.

HELP: You can exploit the recursion to your advantage. If you can prove that at any step of the recursion, you can extract the vector $\underline{\mathbf{a}}_i$ from already knowing the vector $\underline{\mathbf{a}}_{i+1}$, then the job is complete. Notice that the algorithm actually requires the prover to reveal the final vector of dimension 1, so this actually works.

Follow up question For a vector of length n, how many forks of execution are needed, roughly (just O(n) scaling is fine)?

Next thing is to consider that in the core equation that links the steps, $C' = C + x^2 L + x^{-2} R$, we are going to take C' as a commitment whose opening we know (see previous paragraph), and we have three unknowns C, L, R; we only actually need the opening of C, but since there are three unknowns you will need 3 transcripts, so a 3-way fork (or "rewind" twice, perhaps). Basically your goal is to express C in terms of three different C' values. I hope that's enough!

4 Bulletproofs

4.1 The Inner Product Argument

We have already seen the heart of this algorithm, so it will hopefully not be very difficult to understand. What changes here is that we commit to two vectors at once $\underline{\mathbf{a}}, \underline{\mathbf{b}}$ (but we already know how to Pedersen commit that), and (a slight

efficiency gain, we "fold in" the commitment to the inner product of those two vectors, i.e. our overall commitment (now P, was C) commits to $c = \underline{\mathbf{a}} \cdot \underline{\mathbf{b}}$, also.

At this point you want to read Bulletproofs 2017, **Section 3**. Before reading it, read Section 2.4 "Notation". A couple of notes:

- *statistical witness-extended emulation* should be read as a slight extension of "soundness".
- ...as additional support, you can *optionally* read some of my old commentary on this algorithm in Section 6.1 of my old analysis; you can stop when you get to 6.1.3 "Knowledge Soundness".

4.1.1 Protocol 1 and Protocol 2

Another way to organize this in your mind: consider that we are examining a relation $\mathcal{R} : \{(\underline{\mathbf{G}}, \underline{\mathbf{H}}, C, c\ ;\ \underline{\mathbf{a}}, \underline{\mathbf{b}}) : C = \underline{\mathbf{a}} \cdot \underline{\mathbf{G}} + \underline{\mathbf{b}} \cdot \underline{\mathbf{H}} \wedge c = \underline{\mathbf{a}} \cdot \underline{\mathbf{b}}\}$. (This is a common notation in academic papers; write the relation as (comma-separated-statement (semi-colon) comma-separated-secret-witness (colon) proved property).

Notice that this is slightly different than writing $C = \underline{\mathbf{a}} \cdot \underline{\mathbf{G}} + \underline{\mathbf{b}} \cdot \underline{\mathbf{H}} + \underline{\mathbf{a}} \cdot \underline{\mathbf{b}} U$ for some single generator U. That is what the paper calls "Protocol 2", and the reason it uses it is because it is more efficient to "wrap up" all three elements into a single commitment (group element).

Question: If the Verifier has c and C given to him by the prover, can he then construct $C^* = C + cU$, send it to the prover, and ask him to run Protocol 2? In fact it would not be safe; explain why not.

This is why the Bulletproofs paper overlays Protocol 1 on top of Protocol 2; the generator U is modified with an unpredictable challenge *after* the Prover chooses (what I called here) C.

4.1.2 Scaling

Question: Give a formula for the size of this inner product proof, expressed in terms of group elements, and scalars.

Question, discussion: Try to get an approximate sense of how the verification time, or amount of computation, varies with the size of the problem statement (i.e. the size of the vectors).

The tradeoff you see here is common to all such "folding" schemes.

4.2 Using an IPA to ZKP that a value is in range

Before reading on: Question: Can you see how an inner product could be used to help prove that a value is within a specific range? Hint: It's easiest to use a range of the form $0 \ldots a^k - 1$. Hint: think about how values are represented as a list of digits (or bits, perhaps).

Read through Section 4.1 of the Bulletproofs paper. It does a pretty good job of explaining the logic of each step. To summarize, we are creating a proof that a Pedersen commitment $V = vG + \gamma H$ commits to a value v in range $0 \ldots 2^n - 1$,

by decomposing v into its bit representation and then enforcing that the vector of those bits, are each values $\in 0, 1$.

4.2.1 Steps leading to (39)

Here I will just fill in some steps of logic, as reaching this equation in particular might be quite difficult to "get", otherwise. As a bird's eye view, we are trying to convert the logic of "prove that the Pedersen-committed value v is in the range $0 \dots 2^n - 1$ with a succinct argument, by converting the statement into an claim that an inner product of the bits of the binary representation of v equates to a certain value related to v, and thus be able to use the log-scaled IPA that we developed in the previous sections."

First check you understand the 3 main constraints we are trying to prove: $< \underline{\mathbf{a}}_L, \underline{\mathbf{2}}^n >= v$, $\underline{\mathbf{a}}_L \circ \underline{\mathbf{a}}_R = \underline{\mathbf{0}}^n$ and $\underline{\mathbf{a}}_L - \underline{\mathbf{a}}_R - \underline{\mathbf{1}}^n = \underline{\mathbf{0}}^n$.

Next, understand why this equation makes sense, for a verifier-given challenge value y: $< \underline{\mathbf{a}}_L, \underline{\mathbf{a}}_R \circ \underline{\mathbf{y}}^n >= 0$. Question: Explain why this is needed instead of just requiring $< \underline{\mathbf{a}}_L, \underline{\mathbf{a}}_R >= 0$.

Formally justifying the use of the vector $1, y, y^2, \dots y^{n-1}$ is usually based the so-called "Schwartz-Zippel lemma". Look it up; note in particular the relevance of the size of the challenge space, and the degree of the polynomial. This is a core feature of many ZKP systems that you may study in future.

The next step in the argument is to repeat exactly the trick of "check that multiple things are true at once by embedding them into the coefficients of a polynomial (which we just did, with a polynomial of degree $n - 1$ in y), for a polynomial of degree 2, in z, to capture all of the 3 conditions in one equation:

$$z^2 < \underline{\mathbf{a}}_L, \underline{\mathbf{2}}^n > + z < \underline{\mathbf{a}}_L - \underline{\mathbf{a}}_R - \underline{\mathbf{1}}^n, \underline{\mathbf{y}}^n > + < \underline{\mathbf{a}}_L, \underline{\mathbf{a}}_R \circ \underline{\mathbf{y}}^n >= z^2 v$$

The third term on the LHS may confuse you, but satisfy yourself that $< \underline{\mathbf{a}}_L \circ \underline{\mathbf{a}}_R, \underline{\mathbf{y}}^n >$ is the same as that term, and it should then be clearer.

To get from here to (39) is a little tricky, but only technically. Bear in mind the following: your goal is to get **one** inner product, not three, but your advantage is: you only need to include $\underline{\mathbf{a}}_L$ (one one side of the IP) and $\underline{\mathbf{a}}_R$ on the other side, and **you don't care at all about dangling terms involving only y, z and constant vectors, since the verifier can calculate them separately**. Hence, just start by putting $\underline{\mathbf{a}}_L$ on the LHS of your IP. Hence you could start something like this:

$$< \underline{\mathbf{a}}_L, \underline{\mathbf{a}}_R \circ \underline{\mathbf{y}}^n + z\underline{\mathbf{y}}^n + z^2\underline{\mathbf{2}}^n > + z < -\underline{\mathbf{a}}_R, \underline{\mathbf{y}}^n > + \dots$$

where $\dots$ specifically means "and other terms which don't depend on $\underline{\mathbf{a}}_L, \underline{\mathbf{a}}_R$. As a reminder, this is not yet a zero knowledge argument; we're not trying to make sure these vectors are hidden, we're making sure we don't have to publish their full length (but, that is about to change).
Also notice the obvious fact that $z < \underline{\mathbf{a}}, \underline{\mathbf{b}} >=< \underline{\mathbf{a}}, z\underline{\mathbf{b}} >=< z\underline{\mathbf{a}}, \underline{\mathbf{b}} >$ which is needed for rearrangements.

Task: Hopefully with these hints you can successfully reconstruct (39).

4.2.2 Making the proof ZK

The short version is, we add vectors $\underline{\mathbf{s}}_L, \underline{\mathbf{s}}_R$ to $\underline{\mathbf{a}}_L, \underline{\mathbf{a}}_R$ to blind them. The Bulletproofs paper breaks this into two parts; in Section 4.1 it shows a step by step proving protocol for the relation (36), with these blinding factors included, but ignoring the IPA that we've just developed. Then in Section 4.2 it shows how to "plug in" the IPA.

This, the full range proof algorithm, has a lot of moving parts. I would recommend, after reading through Section 4.1, you divert your attention to this excellent commentary on the algorithm, stopping before the discussion of aggregation. The notation is a little different but it's worth the effort.

Question: The algorithm uses three challenge values x, y, z. Explain the precise purpose of each one of these challenges.

> Task: Taking concretely an amount in range $0 \ldots 2^{32}-1$, try to calculate the exact size of a range proof, using this construction over the field $\mathbb{F}_p$ where p is the order of the secp256k1 group and group elements are encoded as per BIP340. Then repeat the calculation for 64 bit integers instead.
>
> Hint: The transcript of the proof is something like $A, S, T_1, T_2, \tau_x, \mu, \hat{t}$, then L, R pairs (how many?), then two final scalars a, b to complete the IPA. Note that the proof is of course Fiat-Shamir-ized but the verifier obviously calculates the x, y, z themselves at the appropriate times.

> Task: Without "cheating" by reading Appendix 3, try to sketch out how you would prove that this protocol has SHVK, using the concept of *simulation*?

4.3 Long verification times and Halo

The fundamental problem with this clever "folding" trick to prove complex statements in ZK is, while it produces compact $\mathrm{O}(\log n)$ proofs, the verifier has to "unpack the loop"; specifically in the case of this Bulletproofs IPA, we have to construct the generators $\underline{\mathbf{G}}, \underline{\mathbf{H}}$, or more precisely, we have to construct the final *single* generators G^*, H^*, from the starting vectors, in linear combination. This multi-scalar multiplication is $\mathrm{O}(n)$ work.

Halo/Halo 2 has a really nice idea to avoid this. Read a bit about it here and see if you can understand the gist. Obviously you can also read the original Halo paper itself.

If we have time, we can discuss this general idea of recursive proof verification and 2-cycles of curves. It's really interesting!

4.4 Using an IPA as a Polynomial Commitment Scheme

Obvious first question ... what even is a "Polynomial Commitment Scheme". **Read** the introduction section of Chapter 14 of Thaler's book.

Second obvious question is, "why is that useful?". Remember earlier, how we embedded multiple conditions into a single equation like this:

$$z^2 < \underline{\mathbf{a}}_L, \underline{\mathbf{2}}^n > +z < \underline{\mathbf{a}}_L - \underline{\mathbf{a}}_R - \underline{\mathbf{1}}^n, \underline{\mathbf{y}}^n > + < \underline{\mathbf{a}}_L, \underline{\mathbf{a}}_R \circ \underline{\mathbf{y}}^n >= z^2 v$$

. . . ? Now imagine us doing this with a polynomial of *much* higher degree. Just for fun here is ChatGPT's answer (to a slightly extended version of that second question):

> Context: In SNARKs like PLONK, Halo, Marlin, and many others, the prover wants to convince the verifier that a large computation (encoded as an arithmetic circuit) was carried out correctly.
>
> Why PCSs help: The computation is encoded into polynomials. To check correctness, the verifier only needs to be convinced of evaluations of these polynomials at certain points. A PCS lets the prover commit once to the polynomial, then later prove evaluations succinctly.
>
> Advantage vs alternatives: Without PCSs, the verifier would have to recompute or check the full polynomial (huge cost). With PCSs, verification is reduced to a constant-size proof, independent of polynomial degree.

Question: What specific detail is wrong about this (otherwise, very good!) summary? Hint: Halo.

Imagine the evaluation of a polynomial $p(x)$ as a dot product between the coefficients of the polynomial, say the c_i in $p(X) = \sum_i c_i X^i$, and the powers of the sampled value X, i.e. $1, X, X^2, \ldots X^{n-1}$.

Then it's easy to see that we could have a commitment to the polynomial via its coefficients, and embed an evaluation of the polynomial at a random challenge in the structure $C = \underline{\mathbf{c}} \cdot \underline{\mathbf{G}} + \underline{\mathbf{u}} \cdot \underline{\mathbf{H}} + p(e)Q$ if e is the random challenge, and $\underline{\mathbf{c}}$ is the vector $1, X, X^2, \ldots X^{n-1}$, but evaluated at $X = e$.

We can directly apply the Bulletproofs IPA to this structure! If the polynomial were of degree 1 billion, what would be the approximate size of this proof that $p(e)$ is an honest evaluation of it, at e?

Question: There's a specific reason that the pure concept of "zero-knowledgeness" becomes meaningless for *enough* evaluations $p(e_i)$ for $i = 1 \ldots q$. Do you see why? Does this violate our definition of zero knowledgeness from the start of this document?

5 Further Study and Applications

Examine how the range proof was implemented in code in both Elements/Liquid and Monero. There is another long-standing library that has this code called Dalek; see e.g. this source code.

The main other usage (which is probably the biggest usage in practice) is the application to arithmetic circuits. Considering fan-in 2 gates, we can represent multiplications and additions in such a circuit, and prove in ZK that the circuit

is satisfied for some input-set that is treated as the secret witness. See Section 5 of the Bulletproofs paper, and also:

The process of converting a circuit, via R1CS, into an inner product is explained in detail here. It is a pretty complicated process! But, it follows the exact same paradigm as we saw for the range proof case; we start, now, with $\underline{\mathbf{a}}_L, \underline{\mathbf{a}}_R, \underline{\mathbf{a}}_O$ for the multiplication gates (linear constraints, i.e. adding constants or multiplying by a constant, are "free") and then use basically the same tricks to (a) achieve zero knowledge and (b) convert into a single inner product so we can apply the IPA folding scheme.

If there's time, examine the case of the "Frozen Heart vulnerability" as it applied to Bulletproofs:

5.1 Interesting docs

1. https://zcash.github.io/halo2/index.html
2. https://docs.zkproof.org/reference.pdf
3. The DALEK documentation is quite good!
4. Teaching video: how the Bulletproofs IPA is used in Halo2