What are Verkle Trees & KZG Commitments, and could they be applied on Bitcoin?

Shymaa M. Arafat
shar.academic@gmail.com, shymaa.arafat@gmail.com

An article submitted in partial fulfilment of the Trailerblazer Tier of the ZK MOOC course 2023

In mid 2021, I was doing some research about stateless nodes in Blockchains; the status quo was mainly around the idea of Merkle Trees. Until I read Vitalik Buterin blog about Verkle Trees [1], I was mainly concerned at the time about whether could the same be applied to Bitcoin; I was trying to enhanced the best known approach there which is Utreexo [2,3] and this idea like came of thin air and I wondered if I could try it on Bitcoin. However, although I usually find Vitalik writings smooth and informative, I honestly did not understand this one enough to try to apply a similar approach to Bitcoin. Maybe, a background from a Zero Knowledge perspective was really necessary. Now, with the help of the ZK MOOC [4] and specifically the Plonk lecture [5], I understand. So, let me share with you all in the next few lines what I have comprehended ....

First I will start with some briefing about the concept of stateless nodes in Blockchains, the common use of Merkle Trees within it, and about the Bitcoin Utreexo which is considered the most efficient til now. Then, I will explain how the idea of Merkle proofs are viewed from ZK perspective as a vector commitment, how vectors can be transformed into polynomials using Lagrange interpolation. After that I will take you into a deep dive into the KZG commitment scheme and Ethereum Verkle Trees. Hence, I can propose my idea of applying the same methodology on Bitcoin, and compare my idea to Utreexo alone and the FRI project on Utreexo; the latter of which I will have to explain it briefly first.

Stateless nodes

Stateless nodes emerged as a concept in Blockchains starting from 2012s as a halfway compromise between full nodes that contribute their computational resources to verify every transaction and every block, and light nodes that verify only their own transactions using SPV in Bitcoin for example; however, removing the verify burden naturally put such nodes at more security risks. Stateless nodes provide a reasonable compromise by having a trusted server keeping track of all needed information of the full system state (whether it's the UTXO set in Bitcoin or the accounts on Ethereum). You can find my attempt at a comprehensive literature survey in [6]; in a nutshell the main approach was for the stateless server to store a Merkle structure of the system states while stateless clients(nodes) store only the Merkle root and ask for proofs as needed in each transaction, see Fig.1.

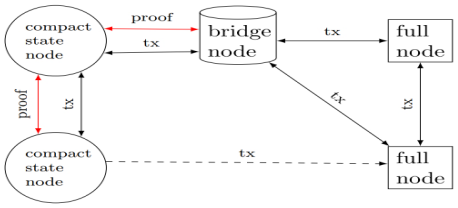


Fig.1 stateless nodes design

Merkle Trees

Almost everyone worked or studied Blockchains knows about Merkle Trees as structure that verify every leaf node in it (the system state) using a logarithmic proof size about the path from this leaf node to the Merkle root[1] (stored in the verifying node). However, let us explain them from a Zero Knowledge perspective as explained in [7]; the server here is considered as a prover that is challenged by a witness (the leaf node item the client is trying to verify) and is required to submit a proof that we wish it is as short (succinct) as possible, Fig.2. In addition to the fact that Merkle Trees give us logarithmic proof size, and we knew throughout the course that we can do better than that, the lecture points out to something I didn't notice before; Merkle proofs do not provide a proof of the vector length[2] (the number of UTXOs in Bitcoin, or accounts in Ethereum). Although never read a complain about that at least in the Bitcoin community (not sure whether it was one of the reasons Ethereum moved to Verkle Trees or not), it is something to be added to a 3rd fact that Utreexo does not provide fraud proofs. If you want fraud proofs you will have to choose a longer proof solution, you may read my attempt here [8] and also consider what is written here that Al-Salvador government seeked shorter proofs (a light nodes modified version of Utreexo)[9]. I find all of the above enough incentive to explore the Verkle Tree solution, but we'll have to understand it thoroughly first.

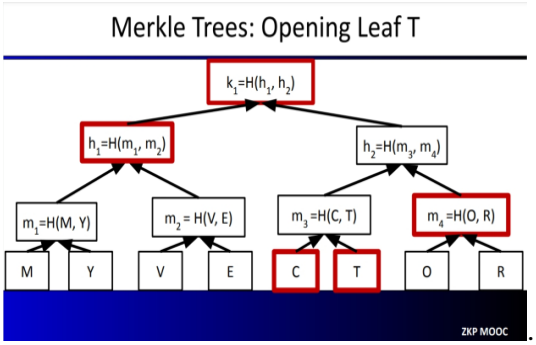
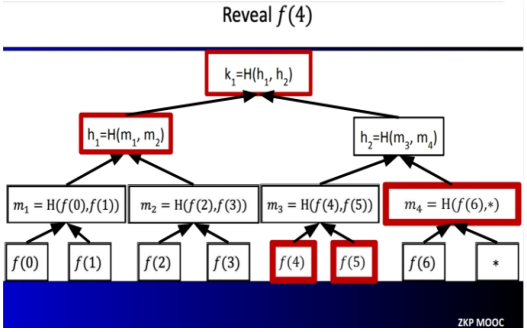


Fig.2 Merkle Trees a)leaves as a vector.



b) leaves as polynomial fn points

[1] Utreexo forest [3] was a breakthrough in Bitcoin for it prevents a worst case degenerate tree that results in linear time proofs by storing log n roots of complete powers of 2 subtrees.

[2] The large prover time was not considered a problem in stateless servers since they all build the tree only once then insert or delete in the same order complexity of finding a proof; that what resulted in the degenerate tree possibility which was solved by Utreexo, where new insertion take place at the end.

From Vector to polynomial coefficients representation

If you have heard the part of the lecture in [7], you probably know that it's not advisable for someone to prove a polynomial using a Merkle Tree; however it is really useful to try the other way around. What researchers realized they can benefit from is to treat any vector of *n* values as a set of points on a polynomial function of degree (*n-1*), and then apply one of the known polynomial commitment schemes on them to get shorter proofs [5]. For those of which who need the polynomial to be in its general form, we use ***Lagrange interpolation*** to get the coefficients [9]; the main idea is as follows:

- if the (a1,f(a1)) satisfy your function F(X), and let us assume for now f(a1)=b to ease understanding since it's a constant anyways; this means the modified function F(X)-b=0 at point a, consequently this means (X-b) divides the function F(X)-b;

F(X)-b=(X-b)* some other function of degree n-2. (let's call it q(X) for now)

-Getting back to our terminology:

F(X)-f(a1) = (X-f(a1))* q(X). => F(X)= (X-f(a1))*q(X) + f(a1)

-If we repeat this interpolation for all the values we have, and keeping in mind that all f(a) values are given as input parameter for our function, then for F(X) to represent the set of values {f(a_1), f(a_2), ..., f(a_i), ..., f(a_n)} the following formula is already derived for the coefficients[3], and in fact is present in Vitalik blog too [1], where "d" here is in place of n in what I wrote above:

$$\lambda_i(\tau) = \frac{\prod_{j=0, j\neq i}^{d}(\tau - a_j)}{\prod_{j=0, j\neq i}^{d}(a_i - a_j)} \in \mathbb{F}_p$$

let's consider Toi to be in place of X temporarily, thus:

$$f(\tau) = \sum_{i=0}^{d} \lambda_i(\tau) \cdot f(a_i)$$

If it helps to view it in Vitalik terminology, here is it:

$$g(X) = r^0 \frac{A_0(X) - y_0}{X - x_0} + r^1 \frac{A_1(X) - y_1}{X - x_1} + \ldots + r^n \frac{A_n(X) - y_n}{X - x_n}$$

Anyways, the point is we can now get the polynomial coefficients in linear time, O(n), instead of

---

[3] If you want more details follow the polynomial example in [1], or try applying to the next a value, say a2, that must divide F(X)-f(a2)
F(X)-f(a2)= (X-f(a1))*q(X) + f(a1) - f(a2) = 0 at X=a2
(a2-f(a1))*q(a2) + f(a1)-f(a2)=0
q(a2)=[f(a2)-f(a1)]/[a2-f(a1)]
we can repeat again considering (X -a2) to divide [q(X) - this rational term] and so on

O(n log n) if we used Number Theory Transform, NTT. This Lagrange step was like a mandatory basic tool that we used in many commitment schemes through the course; we will use it here in the steps of the KZG Commitment scheme itself, and also in the way we use it in place of Merkle Trees. Also keep this linearity in mind as point of comparison with building a Merkle Tree, or forest, for a set of values that constitutes the system state at the starting point of time (say Block "B").

The KZG Polynomial Commitment Scheme

First, KZG is not an abbreviation like SNARKs or something; it's simply the initials of its three authers Kate, Zaverucha, and Goldberg. Any commitment scheme in general takes some input data of considerably large size, say d, and as a prover tries to convince the verifier of possessing the whole correct data by sending short proofs; think of proofs as expert witnesses that testify of correctness without you having to go through million or billion sized data, or as exam questions that tests students understanding without going through every piece of information addressed through the course. The Merkle Tree was a commitment that provides logarithmic length proofs for a vector of data, while the KZG takes a ***polynomial function of degree d*** and work on it in a finite field of degree ≥ d (that's why it proves the degree bound too compared to Merkle Trees) to magically produce ***constant size proofs.*** The magic comes from ***pairing*** [9] and some other finite group fields properties on elliptic curves, Fig3[4], which unfortunately makes it ***vulnerable to quantum computing*** like anything derived from elliptic curves.

Let us first explain how the KZG scheme works and get back later to Fig.3 when we understand why do we need pairing. The KZG is a trusted set-up commitment scheme so a startup phase must take place at the beginning:

-We pick a finite field F_p where p is a very large prime (such that all operations/calculations we mention along the way is going to happen "mod p"; ie, groups are closed under F_p).

-We define a ***finite cyclic group*** that is closed under Fp; ie, all elements {I, G, 2•G, 3•G,....(p-1)•G} ≤p-1 because add/mult is done mod p.

-We have I•G=G; "•" is multiplication and I=0 if it's an additive group, while "•" is exponentiation and I=1 if it's a multiplicative group. In practice elliptic curves use additive groups, you may explore [10] and the links added in it for why the original paper and many lectures use multiplicative group notation when they go deep in the details, while in reality elliptic curves use additive groups.

-Good for us that those groups are ***homomorphic***, something we are going to need in what

[4] Pairing and other Group and finite field properties was explained in lecture 6 of the course, but if you want a really deep dive in details I recommend [9] from which Fig.3 are annotated screenshots that speak for themselves better than my words.

follows to support editing, which means that associativity as we used to applies here; ie,
6G = 2G + 4G, 7G=3G+4G=5G+2G,.... for additive groups

-The ***trusted setup*** procedure starts by randomly picking a random secret[5] Toi (τ) and calculate some values from it which we are going to use through out the protocol. Then the secret τ is like destroyed in a mysterious way (think of throwing it out of a flying plane or killing everyone who knew it!!!), while the generated values are made public to the prover & verifier with no leakage fear since the Finite field math makes it computationally infeasible to retrieve the secret τ back from them (just not post quantum). Here, in KZG, we use τ to generate the values:

G, $\tau \cdot G$, $\tau^2 \cdot G$,..., $\tau^{(p-1)} \cdot G$, let's call them
$H_0, H_1, H_2, ..., H_{p-1}$ respectively, we will call this p-tuple **gp**

-Now, for any function f of degree n≤p, if we wanted to calculate $f(\tau)$ it is
$f(\tau) = f_0 + f_1 \cdot \tau + f_2 \cdot \tau^2 + ... + f_n \cdot \tau^n$

-If we defined the commitment to be $com_f = G \cdot f(\tau)$ then
$com_f = f_0 \cdot G + f_1 \cdot G \cdot \tau + f_2 \cdot G \cdot \tau^2 + ... + f_n \cdot G \cdot \tau^n$

which we can substitute with the H values (the **gp** tuple) as follows:
$com_f = f_0H_0 + f_1H_1 + f_2H_2 + ... + f_nH_n$
Then we can calculate the commitment without needing to know τ), also recalling that all arithmetic is done in "mod p" the result is just one field element from our original finite cyclic group. Note that this trusted setup phase (the H values go tuple) does not depend on the polynomial function we are trying to prove, that's why it's called ***Universal Trusted Setup*** and considered an advantage when compared to circuit specific setup protocols like Groth16.

-Now, we proceed to the 3 regular steps of any interactive proof, where the verifier challenges the prover to evaluate a randomly generated point, say "u", on the function; ie, after receiving "u", the prover should send back "v" along with enough proof that v=f(u).

-The way to prove the correct evaluation here depends on Lagrange interpolation that I have just told you about in the previous section; if v=f(u) then (X-u) must divide f(X)-v, so the prover must prove the existence of a quotient polynomial q(X) in the equation:
f(X)- v = (X-u) * q(X)

-The prover proves q(X) by sending a commitment of it, $com_q = G \cdot q(\tau)$
-The verifier then uses the pairing magic to check the equation is satisfied at X=T; ie,
$f(\tau) - v = (\tau - u) * q(\tau)$.
mult by G:
$com_f - v \cdot G = (\tau - u) * com_q$

√. √. ? √

[5] Sometimes referred to as **CRS** for Common Reference String if you happen to see the abbreviated term or SRS/URS; see https://crypto.stackexchange.com/questions/88565/crs-vs-srs-in-zk-snark, or page 82 in the original reference that standardized it https://t.co/1HMN81Kj1u (this is a pdf downloading link)

By multiplying both sides by G the verifier can substitute with the commit values without needing *T*, then remains the check for (*T*-u) and that's where pairing is used. If you can trust me for that (I mean the scientific material) and prefer not to bother your head with the details jump over Fig.3 and go to Fig.4 directly for a summary of the KZG steps, if you care to understand how pairing works and where it comes from try to follow the details in Fig.3 or watch [9] for further deeper understanding.

-Finally, remember that KZG supports batch proving with only one commitment needed, what is called **π** in Fig.4, and that as we will use it next. Remember also, that we already learned how to transform in linear time a point-value representation into a polynomial coefficients representation using Lagrange interpolation; ie, having a vector of values instead of a set of coefficients is not a problem for KZG.

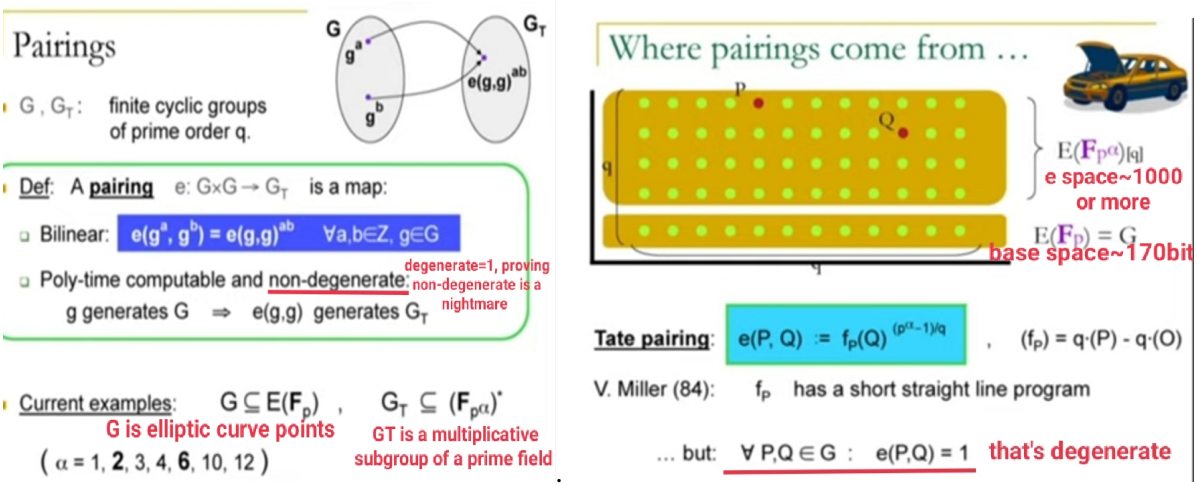


Fig.3 visualizing pairing, an annotated screenshots from [9]

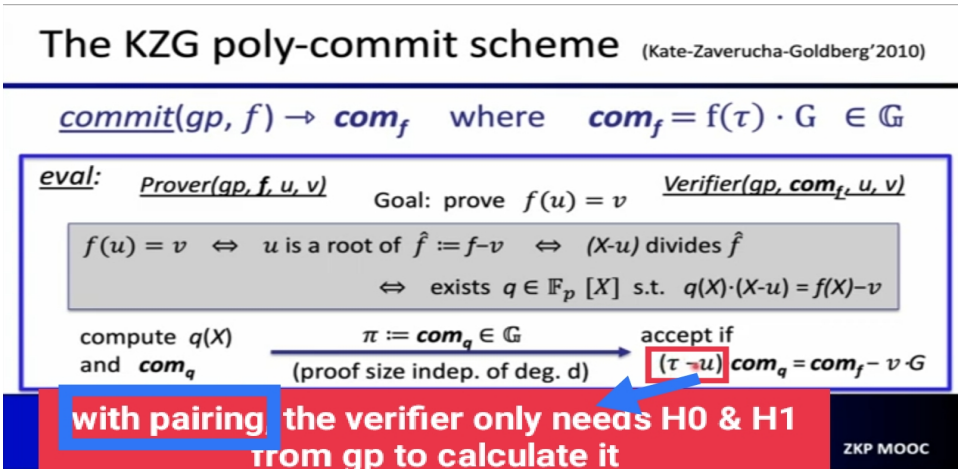


Fig.4: the KZG steps, an annotated slide from the ZK MOOC, [5]

Verkle Trees

After understanding the KZG polynomial commitment scheme, one can easily follow Fig.5 to see how it's applicable to vector commitments and can replace Merkle Trees with much shorter proof size; only 2 field elements. The stateless server (the prover) simply starts by transforming the set of values representing the system into the corresponding polynomial coefficients using Lagrange interpolation, and commit to the polynomial, then the verifiers (stateless clients) asks the server to send proofs for their challenges (verify Bitcoin UTXOs in the transaction in hand, or whatever elements of the system state in the corresponding Blockchain); naturally multiple proofs can be batched when possible (within the same system state, one transaction or could be one block).

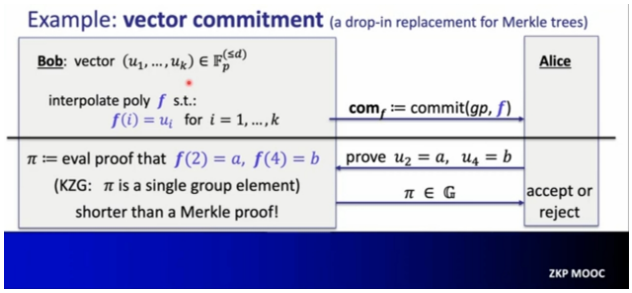


Fig.5: KZG to replace Merkle Trees

Comparison with Utreexo

Note also that in KZG we can batch multiple proofs at once which is pretty useful in verifying for example all UTXOs in a block or at least a transaction in Bitcoin; this is comparable to when Merkle solutions try to arrange the leaves such that most requested proofs are adjacent and thus have many siblings in common. In other words, we moved the saving from probable due to the Pareto distribution nature of UTXOs (Tadje Dyraj claims in his paper [2] that he gets near linear performance in actual runs and that the 2logn worst case is very rare), to 100% guaranteed linearity no matter how the UTXO spending distribution changes. Finally, a fraud proof is always conceivable in the KZG case (the numbers will not match), while Utreexo was once criticized for lacking that (although still favored for the performance).

Edits & the Homophobic property

Recall when I said we are lucky to have the Homophobic and the batching properties, that's because they are pretty useful in the specific problem of handling (highly dynamic) system states in blockchains. Simple and short as in[1], actually derived from the details in [9] and lecture 6 that $e(g^a,g^b)= e(g,g)^{ab}$, Fig 3a, we can calculate new commitments from older ones since

Com(F+G)= Com(F)+Com(G),. Com(F) we already know

$G=Li * (v_{new} - v_{old})$

Where "v" here is the same as our terminology, while "L" here comes from Lagrange interpolation formula; ie, **Li** is a pre-computed commitment for the polynomial that equals 1 at

the position we're trying to change and 0 everywhere else[6]. Maybe the exact code used in Ethereum to calculate Li will differ in the case of Bitcoin for the difference between the underlying structure we are trying to replace is it a Merkle Patricia Trie or a complete Merkle Tree or forest; however the fact that it costs only a constant number of elliptic curve multiplications. Maybe I will dig more into this if I turned this idea to a real project; although not exactly strict scientific reference writing style I added an extra Vitalik note with portions of the code in it combined with [1] since this is an article not yet a research paper.

Bitcoin and ZKP - Closure & Final thoughts

Although many of the famous zero knowledge research ideas like Pederson commitments, Mimble Wimble, and Bullet proofs started as suggestions into Bitcoin they were almost nearly never applied on Bitcoin; there is no confidential transactions in Bitcoin except for some rare uncelebrated layer-2 applications[7]. That's for the "hiding", as for the "binding" property of commitments that made blockchainers call almost all their scaling solutions "ZK", the only scaling Bitcoin solid solutions I know about are variations of lightning nodes and Utreexo stateless nodes; infact one may also consider attaching a small side chain to the main Bitcoin blockchain through a small valued dust UTXO a scaling solution (there are more than 10 million UTXOs in average with value less than 1$[8]).

One could say this is probably because scaling is a much more persistent problem in Ethereum to incentivise the research development cycle. However, I found a new starting proposal [11] that aims to add a ZKP opcode to Bitcoin that enables sending shorter proofs, and provide fraud proves too; the proposal is submitted in April 28th so it's pretty early to be fully analyzed and investigated by the top Bitcoin developers. The reply that appeared till the minute of this writing links it to an older thread from Nov 2022 "***Merklize all the things***" [12]; I don't claim reading the whole old thread with more than 10 msgs, but the original email seems suggesting to explore all the Ethereum scaling solutions with brief introduction on all needed terminology from various roll-ups to what is smart contracts; one of the comments [13] excludes ***Verkle Trees*** suggestions as they are ***not post-quantum*** which was not very convincing for me since it's the same for considered competitive ideas as he says in his own words...

"*> Re Verkle trees, that's a very interesting construction that would be super useful as a tool for something like Utreexo. A potentially substantial downside is that it seems the cryptography used to get those nice properties of Verkle trees isn't quantum safe. While a lot of things in Bitcoin seems to be going down the path of quantum-unsafe (I'm looking at you, taproot), there are still a lot of people who think*

---

[6] A pretty usual procedure in ZK polynomial commitment schemes when we define a function that has different values is to use Lagrange interpolation and a Selector function to make each term "i" exists only at value "i" by multiplying by a term derived from S that is 1 in this case and zero otherwise.

[7] There's however, a current evolving email thread "let's Merklize all things" , and a starting FRI project (link)

[8] See bitinfocharts for the most dustiest addresses, and addresses holdings

quantum safety is important in a lot of contexts."

In addition, if I've understood the topic correctly Verkle Trees are a drop-in replacement for Merkle solutions like Utreexo; ie, not a tool to be used by Utreexo which is the case for this newly starting FRI[9] project [14] that he could have aimed at (but STARKs are post quantum?) since I didn't find any appearance of the world "Verkle" in the original post or the whole thread. Maybe he meant that he favors [14] over Verkle Trees because they're post quantum and also is built upon Utreexo which is kind of solid existing in Bitcoin and they do not exactly prefer unnecessary changes to the status quo. Still I believe the idea of providing constant size proofs (instead of existing logarithmic proofs) for stateless clients is worth exploring and not that hard to investigate. I think people who are that concerned about post quantum solutions may prefer to imploy a full node anyway; while people who now use protocols like SPV (Simple Payment Verifier) inspite of Utreexo existence may consider upgrading their security strength if it have such shorter proofs, and those people are not probably worried about post quantum scenarios.I even talked to ChatGPT about it [15]; you may view this article as a WIP (Work In Progress) research idea that I may get into actually coding and comparing performance metrics (if I did not, the article is still here incase someone found more potential for it in the future), and indeed I will try to read thoroughly and follow up on the Bitcoin developers mailing list. So *please add to my knowledge, correct my wrongs, and express your options in the comments.*

References

[1] https://vitalik.ca/general/2021/06/18/verkle.html, https://notes.ethereum.org/@vbuterin/verkle_tree_eip

[2] Dryja T.,"Utreexo: A dynamic hash-based accumulator optimized for the bitcoinutxo set.", IACR Cryptol, ePrint https://eprint.iacr.org/2019/611.

[3] Utreexo site

[4] ZK-learning; https://zk-learning.org/.

[5] lecture 5, part 1, KGZ polynomial commitment scheme; https://youtu.be/tAdLHQVWlUY

[6] https://github.com/Shymaa-Arafat/lab2/blob/main/DeFi_2UTXOS2_NewOutline.pdf

[7] lecture 4 of the course, mins 26-42.

[8] https://github.com/Shymaa-Arafat/lab2/blob/main/DeFi_3fraudProofs.pdf

[9] Dan Boneh, "The Basics of Pairing", 3rd BIU 2013, https://youtu.be/F4x2kQTKYFY

[10] https://crypto.stackexchange.com/questions/105777/what-is-the-difference-between-those-two-kzg-polynomial-commitment-schemes/105778?noredirect=1#comment226483_105778

[9] FRI and STARKs were covered in lecture8 and need a separate article to summarize; let me just say here it doesn't require a trusted set-up, and it's post quantum secure because it depends only on cryptographic hash functions (no elliptic curves or closed group Finite fields), but it produces more than logarithmic order length proofs. Also, they does not have the homomorphic property that simplifies editing for such a highly dynamic environment as Blockchains.

[11] Weiji Guo, "proposal: new opcode OP_ZKP to enable ZKP-based spending authorization"; https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2023-April/021592.html

[12] Salvatore Ingala, "Merkleize All The Things"; https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2022-November/021182.html

[13] Billy Tetrud, "Merklize All The Things"; https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2022-December/021277.html

[14] "A STARK proof to sync a Bitcoin full node in an instant"; https://github.com/ZeroSync/ZeroSync

[15] my dialogue with ChatGPT; https://twitter.com/ChatGPTBot/status/1630816833635794944