You Couldve Invented EigenLayer

(0:12 - 0:22)

Hi everyone, welcome to another episode of the Yboarding session. Today we're going to talk about you could have invented Agilear. I'm Kaido, I'm a protocol researcher here at Agilear.

(0:22 - 0:27)

Joining me today is Jeff, our smart contract architect. Great to have you here Jeff. Yeah, pleased to be here.

(0:27 - 0:35)

Awesome. Today we're going to talk about three, some... Kaido, I'm a protocol researcher here at Agilear. Joining me today is Jeff, our smart contract architect.

(0:35 - 0:38)

Great to have you here Jeff. Yeah, pleased to be here. Awesome.

(0:38 - 0:52)

Today we're going to talk about three subjects. First, we're going to describe why Eigenlayer, like why we think it should exist. Then from that why, think about what are some requirements we need to hit to build it out.

(0:53 - 1:03)

Lastly, once we know what our requirements are, let's build it out. Without further ado, let's jump into it. Let's start with the why.

(1:04 - 1:26)

As we go through the why, we're slowly going to fill in the requirements, what we need to build out Eigenlayer, what to solve this problem. Let's start with the thing that we all understand very well, which is the current model, how people build decentralized applications. We currently build decentralized applications this way, and we're going to have a very ETH focused idea here.

(1:27 - 1:42)

We build out decentralized applications on top of Ethereum. Ethereum provides the underlying security for all these applications. These dApps, I think, are very well-known.

(1:42 - 2:11)

Just to give out some examples, it could be a DEX, it could be a money market, it could be a yield farm, it could be a roll-up, etc. However, there are some limitations on what kind of dApps you can build, and those are generally infrastructure-related projects. When an infrastructure project, for example, a bridge wants to build, it cannot tap into Ethereum security.

(2:11 - 2:35)

It has to basically bootstrap its own security set to secure its decentralized service. Normally, this kind of security is based on a POS system, or proof-of-stake system. And these proof-of-stake systems, people use this because the other alternatives are proof-of-work and proof-of-authority, and proof-of-work is too energy intensive, proof-of-authority is overly centralized.

(2:36 - 3:04)

And when people decide to go on proof-of-stake, they need to think about, okay, what is the stake, right? We get the word stake in it, and people generally, once again, do the logical next step, which is they launch a native token. And we see this all the time with all layer ones, bridges, they're trying to secure different things with each other. However, we realize that actually the stake part is really, really difficult, for two parts.

(3:04 - 3:21)

Stake is a capital problem, meaning you want people to contribute capital. So you want a market structure, right? And the infras are basically people who are building the infrastructures. So there's a lack of platform for the stakers, or the people who can provide capital to find the people who are building the infrastructure.

(3:22 - 4:04)

That's the first thing we found, which is we want a platform to connect stakers and infra builders. That's the first thing we identified, right? They're builders, they're capital providers, stakers, how do we connect the two? The other problem we realized, you know, why is this really difficult to build is because native tokens in general are very volatile. Crypto is volatile enough, these new tokens are even more volatile.

(4:05 - 4:54)

The other thing is actually very hard to get these tokens, right? They're not listed on a lot of DEXs, they're not listed on a lot of centralized exchanges, so they have an access problem. So the second problem is, do you have to, right? This stake, does it have to be from this native token? Can it be from other tokens? That could be another thing we're trying to solve here, right? So I would just put a stake with other tokens. Yeah, and so to point out both of these address kind of aspects of the same issue, which is if you're an infrastructure developer, you don't necessarily know or even know how to or want to bootstrap your own kind of network by going out and finding all of these stakers yourself and convincing them to use your own tokens.

(4:54 - 5:18)

Yes, and you don't even know where the stakers sort of are, right? You have to sort of go to the blind world and maybe you only can find a subset of them. If there's a place where you can just tell everyone they're in the same spot, that would be the most advantageous. The other thing is, you also have to convince these stakers to buy your volatile token, and also it's hard to buy and sell, hard to acquire, to buy them to be part of the security infrastructure.

(5:19 - 5:48)

There's a lot of bottlenecks here. And the last thing is, for these stakers, when they buy your token, they're incurring an opportunity cost of, when they stake into your system, they're incurring an opportunity cost if they just bought ETH and staked ETH. So this is basically, how do we compete with the native yield ETH gives you? Yeah, so if they were staking with ETH instead of your token, they'd be getting the ETH staking rewards.

(5:48 - 6:02)

Which is like 4%. Yes, and in all likelihood, your native token might be more volatile. So you're taking risk on both the price and the yield.

(6:02 - 6:19)

Yes. And if you want to compete, you need to convince them that your yield is good and your token is stable or looking good. Exactly, which is the yield is good part, which makes a lot of the Cosmos and that's the most prominent POS chains.

(6:19 - 6:53)

A lot of them are around 10, 20, even 30% emission rewards because they want to compete with that native yield ETH is giving. So I'm going to just rephrase it into lower opportunity cost. And the last thing is many times as we become more modular and become more complex in our designs of this blockchain system, a lot of dApps are going to rely on different infrastructures.

(6:53 - 7:18)

And the easiest example, we can think of a dApp that's a DEX that can do cross-chain swaps, right? The infrastructure here would be a bridge provider. And because your dApp's now being secured by two things, to attack your system, I do not need to attack Ethereum, I just need to attack the part that has the lowest security, which is infrastructure. And you have a fragmentation security problem.

(7:18 - 7:37)

To attack your system, I just need to find the weakest link and that becomes really, really problematic. So it would be better if we can pool security or in some way to bootstrap the security of the infrastructure from this huge ETH set so the lower bound of attack is going to be a lot higher. Strengthen your weakest link.

(7:37 - 7:54)

Yes. So let's say, how do we do it? We're going to introduce the concept of restaking here. Basically, you're securing both at the same time with the same stake to pool security instead of fragmenting them.

(7:57 - 8:13)

All right. And now we cover, we basically speed run through the intro section of the article. In the actual article, we go much more in depth about why these are the actual problems and why these requirements are set this way.

(8:13 - 8:32)

But now we have all these requirements here, we're going to go jump into the building section, right? That's, I think, the meat of this whiteboarding session. We're going to think about, okay, on a smart contract level, how do we actually achieve each of these goals we set? Okay. Conceptually, we realize what this platform is.

(8:32 - 8:52)

How do we implement it on Ethereum? Okay. So let's do a really simple sketch to start out with. So we've got our staker and we're going to have some kind of a smart contract pool where stakers can put their tokens in.

(8:53 - 9:17)

So stakers need to be able to deposit. They also need to be able to get their tokens back so they have to be able to withdraw. And then the last piece of this conceptually is that there needs to be some way to slash these, the staker kind of misbehaves in some way that the developer has grabbed.

(9:17 - 9:32)

I see. So in, just going back here, that means each individual infrastructure developer would just write a contract and whatever they want the stakers to commit to, they would just encode that into the slashing contract. Yes.

(9:32 - 9:36)

The EFX, the UR slash. Exactly. Okay, perfect.

(9:36 - 9:49)

And so now we actually implemented the first one. What about, how do we do the second one? Staking with other tokens? Yeah, staking with other tokens. Well, this token pool could be any token.

(9:49 - 10:02)

Yes. Because we're on Ethereum, there's like a million different tokens. Yeah, so actually we, because on Ethereum, we can actually solve this problem, right? If you're the infrastructure developer, you can just say, hey, we're going to take USDC.

(10:02 - 10:10)

Hey, we're going to take X token. We're going to take Y token. It's entirely permissionless for people to use any kind of tokens because you're living on Ethereum.

(10:11 - 10:31)

Okay, next one. How do we lower the opportunity cost for stakers, especially the company with the native yield? So luckily, there's already a bunch of existing tokens that essentially encapsulate Ethereum's native yield. So these are liquid staking tokens, or LSTs.

(10:32 - 10:44)

So we could just pick an LST as the token that goes into the pool. And then these stakers can continue to get Ethereum's kind of native yield while being in it. Wow.

(10:44 - 11:18)

What if I don't hold a liquid staking token? I'm actually an ETH staker myself. How would I, can I still be part of this? Yeah, so in that case, we could build a system that allows people to use their kind of natively staked ETH. And at Eigenlayer, we have built a system called Eigenpods that enables something like this, but it's fairly complex, so I think we should skip over it.

(11:19 - 11:31)

Okay, but it's possible. So it's possible to have native ETH sort of deposit here, or we can use LSTs. Either way works, but just Eigenpods is just a lot more complicated.

(11:31 - 11:44)

Yeah, it's another intermediary that makes it possible to accomplish basically the same thing, but with natively staking. I see, I see. Okay, so it seems like we got another problem solved with this really, really simple design so far.

(11:45 - 11:59)

Easy peasy. Easy peasy. All right, last part is, we see this problem again, right? Each infrastructure provider needs to build its own token pool, and there's going to be a million token pools and deposit only into one.

(11:59 - 12:33)

How can we share or do pool security instead of fragmenting them once again? Yeah, so if we notice, the depositing and withdrawing is very simple. It should be pretty straightforward how this is going to work with any token pool. The kind of unique part here is the slashing, right? So we can split that up into its own kind of abstraction that is, we could call it perhaps a slasher.

(12:33 - 12:47)

It defines what it is that gets you slashed. Yeah, it does. And then if we wanted to, we could add kind of more slashers that all share the same token pool.

(12:49 - 13:08)

So in this way, we'd be basically combining several kind of pieces of infrastructure that are all supported by the same token pool. I see, but that seems a little bit dangerous if any slasher can slash a token pool, right? So how can the staker choose, I don't want the malicious one to slash me. Yeah, good point, good point.

(13:09 - 13:31)

Yeah, so the staker, we could add to the token pool, staker defines kind of what slashing, what slashers are allowed to slash. It's sort of like enrolling, right? Whenever you enroll into a slasher, you're enrolling to that commitment per se. Yeah, I agree.

(13:31 - 13:39)

That's a good, perfect. So now let's just erase this part since we're no longer having the slash. Yes, we've moved it over.

(13:40 - 13:58)

Perfect. And then we're gonna have the slash function under a bunch of different slashers. So just to recap, now the infrastructure developers actually do not need to build out individual token pools anymore because they just need to build out their slashing provision.

(13:59 - 14:25)

And the staker will stake into one token pool and enroll into different slashing conditions. Yeah, and with that you can basically, through restaking, you can achieve pool security now. And this simple design, even though it seems very crazy, it's actually the minimum viable design for Agiler, right? This achieves all the goals we set out to do.

(14:25 - 14:33)

This is not a product yet because we still have a lot of work to do. A lot of problems to solve. But actually, we are done with the MVP version of Agiler.

(14:33 - 14:51)

Yeah, basic requirement. Awesome. In the remaining of the article or the video, what we're gonna do is we're gonna slowly improve this design to make it into an actual product that people can use as stakers and infrastructure developers can come and join this platform.

(14:54 - 15:02)

Okay, okay, okay. Now let's try to make Agiler more of a product. And there's gonna be product questions we need to solve.

(15:02 - 15:40)

The very first one is, this is similar that all proof-of-stake protocols run into, which is the people who provide capital or the stakers are different from people who are actually running the softwares, right? The stakers doesn't operate. And to solve that, we need a separate entity, but we need to think about how do we actually work those separate entities together? So yeah, there's plenty of professionals or semi-professionals who, so run software, we term them operators. Okay.

(15:41 - 16:20)

But in trying to have, so ideally, we'd have stakers delegate to operators who do the actual operation. But again, we have this problem where really we need a platform that help these two sides meet each other. So we'll introduce another smart contract, again on Ethereum, that we term the delegation manager because it handles the delegation between staker, delegating the staker's stake to the operators.

(16:21 - 16:30)

Precisely. Okay. And the first thing that this is going to need is the ability for a staker to actually delegate to an operator.

(16:31 - 16:35)

Yep. With the veto, and perhaps we can draw an arrow. Uh-huh.

(16:35 - 17:03)

Staker there. So what happens when the staker delegates to a specific operator? Does it, you know, basically, yeah, how does that, what sort of things you're tracking here? So here, basically we're tracking, we're taking the stake that we have tracked in the token pool and communicating to the delegation manager how much is staking. So the delegation manager understands how much is delegated to each of these operators.

(17:03 - 17:15)

I see, yeah, that makes a lot of sense. But also because, you know, the operator are running the services, does it still make sense for a staker to enroll into different slashers? Not really. Yeah.

(17:15 - 17:42)

Yeah, so we can erase enroll here, and instead think about enrollment happening at the delegation manager level. And while we're at it, you know, people might not want to serve these applications forever, so we can add an exit function that allows operators to disenroll from any slasher. I see.

(17:42 - 17:54)

So basically we need to remove this arrow because we're no longer tracking it there. Good point, and move it down to here. Yep, 100%.

(17:54 - 18:15)

Perfect, and now I think with this design, we sort of separated the role of stakers and operators by introducing the role of the operators and also the delegation manager to facilitate the delegation between the two. Yeah, we split up these user groups and provided a way for them to interact with each other. And I said, yep, perfect.

(18:16 - 18:28)

And now we have, we achieved, you know, the stakers doesn't need to run a bunch of auction softwares. Okay, now we figured out, you know, the separation. There's another problem.

(18:29 - 18:42)

We are only, for the staker, right, we're only allowed right now to stake one token. That means if we want to stake another token, we need to launch this entire thing once again. How do we solve that? Well, that seems easy enough.

(18:45 - 19:08)

No, so we have this kind of single token pool. It would be nice if we had a bunch of token pools that each kind of split out and, you know, each managed their own token. And we could combine these with kind of one, you can think of it almost as like a puppeteer.

(19:08 - 19:15)

At each one of these token pools, it's just like a little puppet that gets dragged up. I see. So we could call these the token pools.

(19:16 - 19:40)

This could be the token manager, the token pool manager that just kind of takes care of coordinating all of these token pools together inside it. I see. So the other way to sort of understand it, correct me if I'm wrong here, is you can also understand the token pools as LP manager, where they instantly deposit the tokens into a token pool.

(19:41 - 20:01)

For example, I deposit LST, goes into LST token pool, and then the token manager tracks how many shares I have over the total shares of, you know, the old LST I've been depositing. Yeah, like a liquidity provider LP accounting model we're using here for the smart contract side. Yeah, that's a good description.

(20:02 - 20:26)

And using that kind of share-based accounting also means that we can handle things like rebasing tokens or tokens where kind of the yield from each staking perhaps is captured in a variety of ways and unify them. I see. That also means we need to modify a little bit on the stake and withdrawal function.

(20:26 - 20:35)

We basically have to say, I mean, it's still a stake, but instead of just calling it, we have to pass it in which token we're staking. Yeah, you want to pass in a little more. I see.

(20:35 - 20:42)

That's the same for the withdrawal part? Yeah, of course. Perfect. And now we can support more tokens.

(20:42 - 20:53)

Any dump. Oh yeah, actually, yeah, that's absolutely correct. Okay, next up we're going to talk about expanding the infrastructure design you can have under this infrastructure.

(20:53 - 21:28)

One thing we didn't really talk about is how does this withdrawal actually work, right? In this entire new design, after we added delegation manager and the slashers, the withdrawal is going to first check, okay, which is the staker delegated to, and it's going to check, is this operator, what is this operator is currently enrolled in, and it's the enrolled services saying he is, right, there's multiple steps here. So this is a potential problem that could happen. For example, if I'm an operator who's also a staker at the same time, this is an attack I can carry out.

(21:30 - 21:54)

This is a time axis, and this is first where I commit something malicious, meaning someone can submit evidence to the slashing contract and slash my funds. Right at the same block, or right after it, what I do is I call the withdrawal function. Because I submitted so close before anyone can submit the evidence to slash me, I can actually get my funds out.

(21:54 - 22:01)

So at this point, I have zero funds. At this point, I have 100%. At this point, I have 0%.

(22:01 - 22:18)

And somewhere down in the future, someone's going to submit a slashing function, call the slashing, and maybe this is only 500 milliseconds. Actually, that's not possible because we have the block time limit. Go with the idea.

(22:19 - 22:52)

Even though it's very, very small, you cannot let this happen. So to prevent this from happening, we have to introduce a concept called unbonding. Jeff, you want to explain further? Yeah, so rather than allowing people to instantaneously withdraw, as in that timeline that you just drew, and erased, and erased, we will have people queue a withdrawal, and later, they will complete the withdrawal.

(22:53 - 23:29)

We could talk about queuing and completing a withdrawal. And so an unbonding period really just forces a delay between starting the withdrawal, initiating it, or queuing it, and completing it. And so what the process will look like is the staker will start the withdrawal process, queue a withdrawal, will decrease the amount that is delegated to the operator who the staker is delegated to, and later, after the unbonding period, the staker will complete the withdrawal.

(23:29 - 23:42)

And at the time when they're completing the withdrawal, we'll check to make sure that enough time has elapsed. And again, that the operator has not been slashed by any of these slashing contracts. I see.

(23:42 - 24:12)

My question here is, when you say the unbonding period, I'm assuming each infoculture developer would have their own unbonding period. Where is that unbonding period track in this entire thing? And what if I'm an operator, I'm enrolled in different infrastructure with different unbonding periods? Yeah, so good question. Okay, so we could say, let's say one of these has an unbonding period of five days, another has an unbonding period of six days, and another has an unbonding period of seven days.

(24:13 - 24:40)

So when you as an operator enroll in each of these, it will look to see if the new slashing contract that you're enrolling in has a longer unbonding period. Because ultimately what we care about is the longest unbonding period of all of the slashing contracts that you've enrolled in. So basically you would track a variable here, sort of like a mapping between each operator and how long the unbonding period is.

(24:40 - 24:54)

Yes, and the length of it will be the longest of all of the slashing contracts that they've enrolled. I see, and that will be updated every time you enroll and every time you exit. If, you know, you trigger the changing condition.

(24:54 - 25:02)

Absolutely, yeah. So if you're enrolled in all three of these, give an example, seven day unbonding period. I see, okay.

(25:02 - 25:39)

So one thing you mentioned at the end is when you complete a withdrawal, you need to check with each slashing contract. Is that correct? Yeah, so because it's this kind of delegated model, when you're withdrawing, you need to check that kind of none of these contracts has slashed the operator that you delegated. So you could imagine looking up kind of every slashing contract that the operator's enrolled in and calling each one of them and saying, oh hey, is the operator slashed? Yes, no.

(25:39 - 25:51)

If they are slashed, then you can't complete the withdrawal, but if they are, or if they aren't slashed, then you can complete the withdrawal. But if the operator is slashed, then you're not allowed. I see.

(25:52 - 26:13)

I know you're a pretty big fan of gas optimization. Does that, you know, this design make you like squeeze? Yeah, so anytime that you're talking about kind of iterating over an array or doing the same thing a bunch of times, it's not necessarily ideal. Also, as a developer, maybe you don't want to handle all of these interactions.

(26:13 - 26:32)

You want to just abstract away just the smallest amount of kind of slashing functionality. Oh, that's interesting. So you're saying the staker, like the token manager is made for a staker, the delegation manager is made for the operator, and you, like right now, the slasher is developed by the infrastructure developer, but it has to interact with all these different things.

(26:33 - 26:45)

Instead, if there's like another entity that they just interact with, that would be a lot easier for them to develop. Yeah, so we could imagine doing something kind of similar to what we did with all these token pools. Wait, let's actually erase this.

(26:46 - 27:08)

Sure. Yeah, move it out over here. So we could imagine we move kind of these slashers over here, and we introduce a new kind of coordinating contract that is the slasher manager.

(27:08 - 27:28)

And it really serves to just kind of coordinate all of these slashers together and do these system interactions itself. I see. So basically the slasher manager will basically, again, acts like a mapping.

(27:28 - 27:38)

Okay, for this operator, it's a slash, yes or no. Yeah, it basically is just slash. I see.

(27:39 - 27:50)

And that would also mean that when you're doing the withdrawal, you don't need to iterate through all the slasher contracts. You just need to check once with the slasher manager and you'll be good to go. Precisely.

(27:50 - 28:04)

Awesome. I think other benefits I see in this is because a lot of stakers are really small players. So if we can cut down gas costs for them as much as we can, we can encourage more people to participate in the system as well.

(28:04 - 28:09)

Absolutely. Ideally, we'd like EigerLayer to be usable by everyone. Yeah, awesome.

(28:09 - 28:29)

I think that basically concludes our entire product pipeline. How can we make EigerLayer a product that's currently used today? And the diagram you see today is basically just Baby. And this design fits very closely with our current architecture.

(28:30 - 28:37)

Some variables' names are different. A lot of it is using as illustrative purposes. The entire idea is close to home.

(28:37 - 28:45)

Would you say that's the case? Yeah, very much. Awesome. Thank you so much, guys, for tuning in for this episode.

(28:45 - 29:02)

We're gonna spend a little time after this just to walk through some other bonus parts. But now we've actually covered the entire thing, how you could have invented EigerLayer. If you wanna learn more, feel free to click on the link below in the description to go on our research forum for more discussion.

(29:03 - 29:06)

Thank you very much. All right. Welcome to the bonus section.

(29:07 - 29:44)

The bonus section is titled, Who's Trusting Who? The reason why we made this section is because, as you can see, the diagram before was really complicated, a lot of different moving parts, and we talked about it relatively quickly. We wanna expand the dedicated section just to explain the cost of something baked in to the entire system. And to get started, just to recap, we have the stakers who are staking tokens, delegating the operators to run these services, and then combine and provide a service that the infrastructure developers are building.

(29:45 - 30:01)

Cool? So, who's trusting the infrastructure developers? Okay, let's go that direction. Who's trusting the infrastructure developer? Let's think about it. Okay, what could an infrastructure developer screw up? The simple example is just code was buggy.

(30:02 - 30:18)

The worst case is your code is buggy, and it caused a mass slashing event. Yeah, very bad. Who will be slashed? So, the operator would commit the bug or do something bad, and the operator will be slashed.

(30:18 - 30:31)

But because the stakers are delegating their stake to the operators, the staker will be slashed as well. There's a very heavy trust assumption on the infrastructure developers to write good code. But that is always very, very difficult.

(30:31 - 30:46)

So, in the meantime, we introduce a mutually trusted community with very limited power. We call it the veto committee. What the veto committee can do is to reverse this accidental or buggy slashing events.

(30:46 - 31:08)

So, instead of trusting the infrastructure developers for writing good code, all we're trusting is the veto committee. If the bug was messy, veto committee can reverse a slashing event. Right now, veto committee, you can think of it as a mutually trusted party, the old three sides of the table.

(31:09 - 31:20)

Okay, and who's trusting the operators in this? Great question. I would say the operator is the one with the most amount of trust placed in the system. This is not just with the eigenlayer.

(31:21 - 31:42)

This is with any proof-of-stake system that people do delegation. So, if the operators is malicious or try to screw other people up, it can cause the stake pair to lose all the stake. At the same time, it can screw the infrastructure developers by providing a really bad service for the infrastructure developer customers.

(31:43 - 32:03)

For example, if the infrastructure developer is an Oracle, the operator can keep giving out really, really bad prices, and the Oracle itself will be buggy and not useful at all. So, there's a very heavy trust assumption placed on the operators. And once again, this is not just for eigenlayer.

(32:03 - 32:18)

This is a general proof-of-stake weakness we see. Okay, what about stakers? Who's trusting them? Oh, this is a great question. If you think about this entire thing, in my view, I don't think anyone is trusting the stakers.

(32:18 - 32:44)

And the reason why no one is trusting the stakers because their trust are being replaced by what I call E plus E. Basically, eigenlayer smart contract on top of Ethereum. Because of this program, basically they're making these programmable commitments, the operators, infrastructure developer do not need to trust on the stakers. If they commit to what they're doing, then why no one needs to trust the staker because of eigenlayer's infrastructure?