(this is from WIP joint work with Chirag Kaudan and Leo Wong)

Given a “restaking state” of the world (a collection of restakers and tasks and an assignment of restakers to subsets of tasks), Eigenlayer performs an analysis of when the state is “cryptoeconomically secure” in form of conditions that when held, no profit can be made by any colluding group from corrupting tasks. We explore this direction by:

1. introducing dynamic

attacks: more sophisticated (but also realistic) attacks that can attack the model even if the inequalities hold.

1. giving a “profit-detecting algorithm” for the bad guys (the good guys can use it as well, or at least be aware that this algorithm is easy for the bad guys)

We think this leads to many interesting questions about how to approach security of restaking going forward.

Preliminaries

- stakers

form a set S

and tasks

(i.e. the middleware that are looking for re-staking from Eigenlayer) form a set T

.

- staker i

has stake

s_i

. Task j

has profit

p_j

and cost of corruption

(as a fraction of total stake) \alpha_j

. * the idea is if a subset of stakers of task j

have at least \alpha_j

of the total stake in j

, then they can gain p_j

of value, but they’ll then have their entire stake slashed.

- the idea is if a subset of stakers of task j

have at least \alpha_j

of the total stake in j

, then they can gain p_j

of value, but they’ll then have their entire stake slashed.

- we define a (restaking) state

to be a set of stakers S

, a set of tasks T

, and a record of which stakers are currently staking which tasks T

, formalized as: * for each j \in T

, S_j

be the set of stakers staking task j

- for each i \in S

, T_i

be the set of tasks restaked by staker i

.

- for each j \in T

, S_j

be the set of stakers staking task j

- for each i \in S

, T_i

be the set of tasks restaked by staker i

.

To visualize S, T, {S_j}, {T_i}

, we keep track of the information in a state with a state matrix

.

As a first pass for the state matrix, it is sufficient to represent the information with a 0

-1

matrix M

with rows indexed by S

and columns indexed by T

:

1

2

3

4

1

1

1

1

1

2

1

1

1

1

3

1

0

1

1

4

1

1

0

0

Here, M_{i,j} = 1

means the staker i

restakes task j

and 0

means she does not do so. It may be helpful for the reader to keep track of how much stake each staker has. Thus, we will work with an “explicit form” of the state matrix, where we let M_{i,j}

be equal to s_i

if staker i

is restaked in task j

. In this case our above matrix (with a choice of the s_i

) would instead be:

1

2

3

4

1

10

10

10

10

2

20

20

20

20

3

100

0

100

100

4

50

50

0

0

Eigenlayer’s main result is that a state is cryptoeconomically secure

if for all i, s_i \geq \sum_{T_i} \gamma_{ij} \frac{p_j}{\alpha_j}

, where:

- we sum over all the tasks T_i

that are restaked by staker i

,

- where \gamma_{ij} = \frac{s_i} {\sum {k \in S_j} s_k}

is the fraction of task j’s securing stake which is held by staker i.

The Attacks

To start, we allow the adversary to make sophisticated timing attacks. This means we assume they can attack multiple tasks at once and get all the profits of each before they are slashed. This is implicit in Eigenlayer, since they already look at the plausibility of attacking subsets of tasks rather than single tasks. There’s some inherent invincibility to protecting against the adversary having good timing, so we can’t really escape this.

We strengthen the adversary in the following ways:

1. We allow the adversary to adjust the amount of restake before/during their attack.

For example, we can assume they can start without restaking in certain tasks and suddenly restake them in a coordinated way.

1. We allow the adversary to split their identities arbitrarily, and/or change their (layer-1) stake.

For example, if the adversary only needs to collectively spend 50

million to attack a task A

, but has 100

million staked (and so are at the risk of having 100

million slashed), they can transfer half of their money to a new account and perform the attack with just the first account so that only 50

million will be slashed.

Both of these seem very realistic, especially in the light of our statement about timing.

Example 1 (shifting attacks)

To show that (1) is a viable attack vector, consider the following example:

1

2

3

1

70

70

0

2

70

0

70

3

0

70

70

where $p_1 = p_2 = p_3 = 50$

and $\alpha_1 = \alpha_2 = \alpha_3 = 3/4$

. We can test that for all s_i

, we have $s_i = 70 \geq 2 \frac{(1/2) 50}{(3/4)} = \sum_j \gamma_{ij} \frac{p_j}{\alpha_j}$

, as desired, so it is cryptoeconomically secure according to Eigenlayer. However, it is vulnerable against the following attack: any 2

players can collude (for example, the first 2

) to restake fully at the same time, which gives them the ability to corrupt all 3

tasks. Then they can get a reward of $150 = \sum_i p_i$

at the risk of losing only $70+70 = 140$

stake, for a 10

stake profit.

Some readers may find this example unsatisfying because the attackers have more than 1/3

the total stake to attack consensus (in fact, they can perform a 51\%

attack since they actually have more than 1/2

the stake!). It is not hard to extend this example to a bigger case such as the following:

1

2

3

4

5

1

70

70

0

0

0

2

70

0

70

0

0

3

0

70

70

0

0

4

0

0

0

2000

2000

5

0

0

0

2000

2000

6

0

0

0

2000

2000

In this case, we can assume rows 4, 5, and 6 are altruistic and non-corruptible stakers representing the “rest of the world.” We can assume all the α_i

's and p_i

's are the same as before and have $p_4 = p_5$

arbitrarily big (say 2000 to be similar in scale to the big restakes). Here the colluding stakers are very small compared to the rest of the stakers, but are still able to attack the system. We can see that the same inequalities still hold; as long as

1. the attackers are restaking some tasks that are “unpopular” with the rest of Eigenlayer, and

2. the attackers locally have a lot of relative stake in those tasks,

such attacks are possible (albeit alleviated by e.g. the continuous relaxation introduced by Eigenlayer).

Example 2 (splitting attacks)

Consider the following initial matrix

1

2

3

1

70

70

0

2

70

0

70

3

0

70

70

4

700

700

700

With p_1 = p_2 = p_3 = 100

and \alpha_1 = \alpha_2 = \alpha_3 = 2/3

.

We can check that the s_1 = s_2 = s_3 =70 \ge2*frac{(1/10)*100}{2/3}=30

and s_4=700>3*frac{(9/10)*100}{2/3}=405

. Thus it is cryptoeconomically secure by Eigenlayer's definition.

However, for rich enough stakers, they could also split up their stakes into two (or more) nodes with one that has the exact sufficient amount to corrupt a set of task(s).

Consider the new matrix below

1

2

3

1

70

70

0

2

70

0

70

3

0

70

70

4

0

0

0

4*

280

280

280

In this scenario, staker number 4 split him/herself into two nodes (4 and 4). *We assume that 4* is the amount of stake that the staker wants to sacrifice.

Now, with a stake of 280

, staker 4 have just enough power to attack and perfectly time the attack to corrupt task 1, 2 and 3 to obtain a profit of $\sum_{j \in \{1,2,3\}}p_j = 100 + 100 + 100 = 300 > 280$

. The slashed stakes are only 280

. Thus in the end the overall profit from the attack is $300 - 280 =20$

.

Profit-detection for an Adversary

We just saw that each type of attack can make a profit on its own. Combining the attacks, therefore, would make even more profit. It is an interesting complexity question to see when profit opportunities would arise.

A priori, detecting if type (1) attacks (restaking-on-the-fly) are possible is an $O(2^{|T|+|S|})$

problem, since we need to search over all subsets of stakers and all subsets of tasks to see if the attack is viable. Reducing the problem from the point of view from a specific subset of attackers changes it to $O(2^{|T|})$

, and this does not even take into account the a priori infinite possibilities of the type (2) attacks (splitting funds, etc.). In this section, we show that this is actually a $O(T \log(T))$

problem, and present an algorithm that a group of attackers can use to search for maximal profit (or alternatively, an algorithm that defenders can use to calculate a suspected colluding group of attackers' maximal profit).

With these assumptions, we can explore their potential earnings in an attack with the following chain of logic:

1. Let's assume that the adversaries are stakers 1 through m

and the other users are stakers $m+1$

through $m+n$

, so our matrix has m+n

rows.

1. From the adversary’s point of view, we can collapse all the other stakers into a single row where their stakes are summed per column (i.e. attacking such a situation is equivalent to attacking the version with n

rows).

1. Similarly, we can just collapse all of the adversary into a single row as well, since by (1) above we can readjust their stake for an attack if necessary.
2. This allows us to just collapse our matrix into a 2-row matrix, with the first row being the adversary and the second being the rest of the world.

Starting with one of our earlier examples

1

2

3

4

1

10

10

10

10

2

20

20

20

20

3

100

0

100

100

4

50

50

0

0

our reductions would change the matrix to (assuming our adversaries are rows 1

and 2

):

1

2

3

4

1

30

30

30

30

2

150

50

100

100

Now, what is a perfect sophisticated timing attack? We claim the maximum damage the adversary can perform is the following:

1. pick some amount of restake C

to be slashed (effectively pulling some money out into other nodes, and make C

be the amount of slashing that an attack would cost, and have a subset of the nodes with total stake C

be the imminent “sacrifice” for the current attack; (formally, this changes the first row to be all C

)

1. simultaneously attack all

tasks where $\frac{C}{C + M_{2, j}} \geq \alpha_j$

, the fraction of stake needed to attack task j

, obtaining value p_j

for the attack.

1. after picking up all available p_j

, the nodes with total amount C

will be slashed.

From an adversary’s perspective, finding C

is a finite problem; it is clearly better for her to look at all the tasks and the minimal

restake needed to attack each task, and then only looking through those minimal valeus as potential values of C

. The minimal restake needed to attack task j

is $A_j = \alpha_j M_{2,j} / (1-\alpha_j);$

that is, a restake of C

is enough to attack task j

if and only if $C \geq A_j$

. Using this observation, the adversary can search through their attacks for maximal profit with the following $O(t \log(t))$

algorithm:

1. sort (in increasing order) all the columns by $A_j = \alpha_j M_{2,j} / (1-\alpha_j)$

.

1. for $j \in \{1, 2, \ldots, t\}$, test $C = A_j$. This would lose C but gain an amount $p_1 + \cdots + p_j$, so the total profit is $p_1 + \cdots + p_j - A_j$

.

1. Find $\operatorname{argmax}_j p_1 + \cdots + p_j - A_j$.

With our running example, assuming that the p_j's are $\begin{bmatrix}90 & 70 & 100 & 80 \end{bmatrix}$, and the α_j's are $\begin{bmatrix} 1/2 & 1/3 & 1/2 & 3/4 \end{bmatrix}$, Then, the list of A_j listed in increasing order (and associated profit $E(j) = p_1 + \cdots + p_j - A_j$) is

Task j	2	3	1	4
A_j	25	100	150	300
p_j	70	100	90	80
$E(j)$	45	70	110	40

Thus, the most profitable for any attackers is to corrupt task 1, 2, 3 with total stake of 150 to be slashed.

It would be interesting to see how far this could be taken – for example, it might serve as good basis for a “collusion detector” for Eigenlayer to run and monitor. (Eigenlayer mentioned this type of construct as a potential idea)

Ending Thoughts

First and foremost, this is a work in progress and we are going to work on it for the rest of the semester. We just thought this would be a nice midway point to share some interesting directions. Any criticism, suggestions, and questions are very welcome!

Second, we think this does not fundamentally make Eigenlayer implausible. We think it just means that the security of restaking is complicated but interesting. Eigenlayer, very reasonably, started with a simple risk model. The ideas presented here are useful and practical ones to keep in mind going forward as we construct more complex but more robust theories of “security.” (in fact, we think the “continuous relaxation” version of Eigenlayer might ameliorate some of our issues; we will look in this direction next) We think Eigenlayer’s foundational work here is very valuable.

Much of the attacks can be ameliorated by making it very hard for stakers to restake quickly (which protects against type 1 attacks) or to change their layer-1 stake quickly (which protects against type 2 attacks). Seems hard to make either go away fully, though.

Finally, when we started thinking about this problem, we thought of the following space of questions:

1. Given a (restaking) state, can a set of stakers S

restake and then corrupt enough tasks to make a profit?

1. Given a state, can a set of stakers S

with an additional amount of capital C

restake and then corrupt enough tasks to make a profit? (this would allow them to e.g. make some new nodes)

1. Given a state, can a set of stakers S

with an additional amount of capital C

restake and bribe

enough other stakers to corrupt enough tasks to gain money? (bribing can be really cheap; for example, you can bribe people to stop staking for a while en masse, so maybe it only costs 100K for 2 million dollars worth of stakers to promise to exit staking middleware A

in one month, and then in one month when they leave abruptly you do a big attack on middleware A

, and those stakers lose no money.)

The third question looks like it might be eventually interesting (but intractable for now?). The second question seems to reduce to the first by assuming S

contained an extra colluder who has staked some amount of ETH C

but has not restaked in anything. Thus, we believe that the first question (as we have studied here) is the best starting point.

(we thank Barnabe Monnot and Davide Crapis for useful conversation. We also thank Sreenam Kannan and Soubhik Deb of Eigenlayer for letting us view their whitepaper early. This work is a part of a collaboration between the EF and SJSU’s CAMCOS program.)

EDIT 1 (4/8/23): corrected a couple of errors with the examples. The numbers were wrong.