The Principal–Agent Problem in Liquid Staking

Apostolos Tzinas*[1,3] and Dionysis Zindros[2,3]

[1] National Technical University of Athens
[2] Stanford University
[3] Common Prefix

Jan 27, 2023
Last update: May 18, 2023

Abstract. Proof-of-stake systems require stakers to lock up their funds in order to participate in consensus validation. This leads to capital inefficiency, as locked capital cannot be invested in Decentralized Finance (DeFi). *Liquid staking* rewards stakers with fungible tokens in return for staking their assets. These fungible tokens can in turn be reused in the DeFi economy. However, liquid staking introduces unexpected risks, as all delegated stake is now fungible. This exacerbates the already existing Principal–Agent problem faced during any delegation, in which the interests of the delegator (the Principal) are not aligned with the interests of the validator (the Agent). In this paper, we study the Principal–Agent problem in the context of liquid staking. We highlight the dilemma between the choice of *proportional representation* (having one's stake delegated to one's validator of choice) and *fair punishment* (being economically affected only when one's choice is misinformed). We put forth an attack illustrating that these two notions are fundamentally incompatible in an adversarial setting. We then describe the mechanism of *exempt delegations*, used by some staking systems today, and devise a precise formula for quantifying the correct choice of exempt delegation which allows balancing the two conflicting virtues in the rational model.

1 Introduction

When a validator participates in a proof-of-stake protocol, they *bond* their stake, locking it up for a period of time in exchange for rewards. This locked-up stake is *slashed* in case of validator misbehavior such as equivocation. Stakeholders *delegate* their stake to *validators* to also earn rewards. If the validator misbehaves, the funds of the delegator are also slashed. This introduces a *Principal–Agent* problem [21, 33] in which the

*Apostolos Tzinas was supported in part by the Algorand Centres of Excellence MEGA-ACE programme.

actions of the validator (the *Agent*) affect the capital of the delegator (the *Principal*).

However, staked assets are *illiquid* and cannot be used for other purposes such as in DeFi applications [35] because they are locked up. *Liquid staking* [25] is an attempt to solve this problem by issuing *token representations* of the staked assets that can be freely traded and utilized by stakeholders elsewhere in the blockchain ecosystem.

Liquid staking token representations are most valuable when they are *fungible*. However, this fungibility exacerbates the Principal–Agent problem of delegated stake. A liquid staking system *pools together* stakes from different participants, prompting the question of whom to delegate these pooled funds to. If the system has *proportional representation*, every pool participant decides whom to delegate to *in proportion* to their contributed shares. The crux of the issue arises from the fact that a malicious pool participant can choose to delegate to a colluding validator who then equivocates. This causes a portion of the pooled money to be slashed, affecting every pool participant in proportion to their contributed stake, even if they made no *unwise* delegation decisions, in a situation of *unfair punishment*. This causes a drop in the price of the liquid staking tokens. A rational attacker profits from this price drop by shorting the token.

Our contributions. Our contributions in this paper are as follows:

1. We introduce two desirable properties in the context of Liquid Staking: *Proportional Representation* and *Fair Punishment*.
2. We showcase the Principal–Agent problem in the Liquid Staking setting by describing a concrete attack leveraging it.
3. We give a precise description of the market conditions that enable this attack, and a formula for a liquid staking system configuration which can avoid it.

Related work. Proof-of-stake (PoS), introduced by PeerCoin [23], was formalized in later works [5, 6, 16, 19, 22]. Slashing [11] is a technique used to achieve economic safety [12] in many PoS systems, among others PoS Ethereum [13, 14] and Cosmos [8–10]. Outside of centralized exchanges, liquid staking was introduced independently by the team of Joe Bowman, Brian Crain, Felix Lutsch, Dev Ojha, Meher Roy (from Chorus One and Sikka), and Hyung Yeon Lee (from B-Harvest) in June 2019. First reported as *Delegation Vouchers* [24] they were later analyzed in a comprehensive report [25] with the help of the Liquid Staking Working Group. Lido [1] and Rocket Pool [31] popularized liquid staking in Ethereum, and Quicksilver [7], Stride [34] and pStake [30] in Cosmos. Quicksilver is the first protocol to propose proportional representation, but this is not yet implemented. Besides liquid staking, stake rehypothecation takes the form of restaking (EigenLayer [17]) and cross staking [3]. Exempt delegations, with one name or another, are used in Rocket Pool, and have been proposed for Cosmos [26].

2 Preliminaries

Loans. For the attack we will describe in this paper, some upfront capital is required. Sometimes a portion of this capital is needed only throughout *one* transaction. Towards this purpose, a *flash loan* [20] is obtained, which has zero duration and no funds at risk. We assume that a loan of duration Δt for capital u has an *upfront* cost factor β, an *interest rate* r and the amount to be repaid, including both the principal and interest, is $((1+r)^{\Delta t}+\beta)u$. The term β models the cost factor of a flash loan, which has duration 0. In order to take a non-flash loan, collateral in a different currency must be deposited for the duration of the loan. DeFi protocols typically require overcollateralization. We denote by $\gamma \geq 1$ the collateral ratio required by the loan provider. To take a loan of z in one currency, one must deposit the equivalent value of γz in another currency.
Proof-of-Stake. Proof-of-stake systems are secured by *validators* who propose and vote on blocks. To become a validator in a slashably safe [12] system, a stakeholder must *bond* their stake which locks it up for a particular period of time in return for rewards. Validators promise they will not *equivocate* by signing conflicting blocks. In case of equivocation, a percentage $0 < p \leq 1$ of the locked stake is *slashed* and the validator is permanently deactivated. In the cryptographic model, validators can be *honest* or *adversarial*. The honest validators run the prescribed protocol, and hence never equivocate, whereas the adversarial validators can deviate from the protocol arbitrarily.
Delegation. Since not everyone has the capacity to become a validator, a stakeholder can *delegate* their stake to a validator to participate in the validation process in their stead. The voting power of the validator accounts for the delegated stake, and delegated stake is also slashed in case of validator misbehavior. The stake bonded by a validator themselves and not delegated from others is known as *self-delegation*. Self-delegations as well as stake delegated from others is known as *delegated stake*, and the capital holder of delegated stake is known as the *delegator*, but will also

be referred to as the *principal.* A principal can *undelegate* or *redelegate* at any time, but must wait[4] for an *unbonding period* δ.

Liquid Staking. Delegated stake earns rewards, but remains locked and is illiquid. Principals often wish to rehypothecate their delegated stake, for example to take loans [20] or to, more broadly, participate in the DeFi [35] economy. Protocols that enable this ability are known as *liquid staking protocols* [25]. Some such protocols [1, 34] operate in the form of smart contracts (*e.g.,* Lido and Rocket Pool in Ethereum) or separate appchains[5] (*e.g.,* Stride, pStake, and Quicksilver in Cosmos). Stakeholders *deposit* their funds into the liquid staking protocol. Upon deposit, these contracts act as delegators and delegate the incoming funds to their choice of validators. They collect this delegated stake into a *pool* and receive staking rewards from these holdings. During the deposit, a new derivative asset is minted, which is given to the depositor as a claim to the delegated stake held by the liquid staking contract. Such derivative tokens, when issued from the same liquid staking contract, are fungible with one another[6]. We are only concerned with liquid staking protocols that are fully fungible. At any time, the derivative asset holder can *redeem* their derivative asset. During redemption, the contract burns the holder's derivative assets and returns the respective assets to the holder. For completeness, we illustrate the basic deposit and withdrawal functionalities of any liquid staking protocol in Appendix B. In our treatment, we consider a generic asset that we denominate in ASSET and the respective derivative token, that we denominate in $\mathsf{stASSET}$, issued by an arbitrary liquid staking protocol. We assume a perfectly efficient market for ASSET and $\mathsf{stASSET}$, as well as a sufficiently deep loan market with rates r_{A}, β_{A}, and r_{st}, β_{st} respectively.

Exchange rates. Initially, ASSET and $\mathsf{stASSET}$ are priced at a 1:1 exchange rate, as one can be exchanged for the other by redeeming or withdrawing. However, the balance of the liquid staking protocol in ASSET holdings can change with time due to two reasons. Firstly, it continu-

[4]The waiting period may sometimes be *waived* and redelegations allowed instantly if no other redelegations have happened within δ, such as in Cosmos [15]. The important point for us is that, after redelegation has commenced, the redelegated stake is still prone to slashing due to the *old* validator's misbehavior for a period of δ.

[5]An appchain is a separate Cosmos zone, connected with other Cosmos zones using IBC/ICA [36] and functions similarly to a smart contract.

[6]The exact fungibility constraints depend on the protocol. For example, stATOM in Stride [34] and stETH in Lido [1] are fully fungible. However, in the proposed Liquidity Staking Module [2] of Cosmos, derivative tokens are only fungible when they have been delegated to the same validator in the same batch.

ously receives rewards for staking the ASSET (these rewards are auto-compounded). Secondly, if a validator it delegates to misbehaves, a portion of its ASSET can get slashed. These events do not change the supply of stASSET in the market. The deposit and redemption operations must adjust their price. Let b_0 ASSET denote the amount of ASSET holdings of the liquid staking protocol, and s_0 stASSET denote the total market supply of stASSET that the protocol has issued. When the user deposits b ASSET, the protocol mints $s = b\frac{s_0}{b_0}$ stASSET. On the other hand, when the user burns s ASSET, the protocol returns $b = s\frac{b_0}{s_0}$ delegated ASSET to the user. These delegated ASSET can be unbonded to convert them to ASSET. Because the user can always go back to the protocol and exchange b for s or vice versa, we assume that the price of stASSET denominated in ASSET in the market is the same as the quoted protocol price. We refine this assumption in Section 7. Users can buy and sell stASSETs by *swapping* them on-chain [4].
Governance. The choice of which validator the liquid staking protocol's assets are delegated to depends on the particularities of the protocol. In centralized protocols, the decision is taken by a central party or central committee of parties, who may not be the liquid stake holders themselves. Some protocols allow the principals to vote. During the voting process, anyone can propose for a proportion of the protocols's assets to be delegated to a validator of their choosing. Each principal can then vote *yes* or *no* to the proposal. Decisions are often taken by weighted majority.

3 Representation

In a staking protocol, whether liquid or not, each principal has an opinion about which validator they wish to delegate to.

Definition 1 (Delegation Wish). *For each principal, we define their* delegation wish *to be a particular validator of their choice.*

It will generally be desired that these delegation wishes point to honest validators. We formalize this in the notion of *wisdom.*

Definition 2 (Wisdom). *A principal is* wise *if their delegation wish points to an honest validator. Otherwise, the principal is* unwise.

In the liquid staking protocols described in Section 2, the decision of which validators to delegate to is up to the majority of the stakeholders. This creates a problem. A stakeholder holding a minority of the stake may wish to delegate this stake to a particular validator, but the rest of the stakeholders can overturn him by a majority vote. Hence, in these protocols we have a situation of *only the majority* being represented, instead of *everyone being equally represented* [28].

On the contrary, in a *proportional representation* system, the majority of the stakeholders decide where to delegate the majority of the stake, but the minority of the stakeholders also decide where to delegate the minority of the stake.

Definition 3 (Proportional Representation). *In a* proportionally represented *liquid staking protocol, each validator is delegated a proportion of the liquid staking pool's ASSET equal to the sum of the proportions of stASSET held by the principals who wish to delegate to that validator.*

To achieve this, the process of liquid staking becomes different: Each principal must *signal* their *intent* indicating which validator they wish the pool to delegate to. This election mechanism was introduced by Quicksilver [7] even though it has not yet been implemented, at the time of writing.

First, the principal deposits ASSET into the protocol and signals delegation intent to the validator of their choice. Then, the protocol delegates the deposited ASSET to that validator. The principal is now a holder of tradable stASSET representations of the delegated ASSET. This stASSET can now be transferred to a different owner.

At some later time, the stASSET holder may wish to redelegate their underlying ASSET to a new validator. The stASSET holder can *resignal* their delegation intent and the protocol will redelegate the underlying ASSET to the new validator. The goal is for all stASSET holders to be represented proportionally to their stake.

Note that proportional representation may not be instant. Redelegation speeds are limited by the underlying blockchain's unbonding period. Hence, a stASSET holder may have to wait δ before their corresponding ASSETs are redelegated to the validator of their choice.

4 The Principal–Agent Problem

In the proof-of-stake systems described in the previous sections, principals grant permission of their funds to validators so they can participate in consensus on their behalf. Principals have ownership of the stake, but validators have control over it. The stakeholders rely on validators to act according to their best interest: Stay online and follow the protocol.

However, validators (the agents) may have extrinsic motivation to misbehave and plot against principals. This creates a conflict of interest that is known as the Principal–Agent problem.

For example, a malicious validator can equivocate, which causes the principal's funds to be slashed. If the malicious validator has limited self-delegation and no reputation to lose, the validator may be able to profit from the principal's loss. However, a validator with a larger self-delegation will themselves be affected by the slashing. This is why self-delegation offers a layer of protection against the Principal–Agent problem.

In traditional staking protocols, principals have the responsibility to delegate their funds *wisely.* When a malicious validator misbehaves, only the stake of unwise principals is slashed. No wise principal gets *unfairly punished.*

Definition 4 (Fair Punishment). *A staking protocol has Fair Punishment if no wise principal's stake gets slashed as a result of a malicious validator's actions.*

With the introduction of liquid staking protocols, the principal is no longer directly delegating to the validator of their choice. Instead, the protocol is now responsible for the delegation process and stake allocation to validators. Although the principal can express their delegation wish, ultimately it is the protocol that decides where funds are delegated based on its *delegation strategy.* All liquid staking tokens are fungible, hence all validators delegated to by the protocol become agents for all principals. The Principal–Agent problem is exacerbated. The principal's funds are now effectively delegated to validators he has not necessarily chosen, some with mischievous intentions.

5 Attack

We now describe an attack an adversary can conduct which leverages the Principal–Agent problem of liquid staking. First, we observe that fair punishment in the class of protocols we are concerned about is impossible.

Claim. Any *fungible* liquid staking protocol with *Proportional Representation* deployed over any proof-of-stake consensus protocol which slashes equivocating validators by a rate of $p > 0$ cannot have *fair punishment.*

Demonstration. To see why the above claim holds, consider the following simplistic attack illustrated in Figure 1. Let b_0 be the amount of delegated ASSET in the protocol's delegation pool, and s_0 be the total amount of stASSET tokens outstanding before the attack commences. The initial quoted price of stASSET is $\frac{b_0}{s_0}$.

Initially, the adversary $\mathcal{A}$ creates a new validator $\mathcal{V}$ under her control[7]. We do not require any of the existing protocol participants to delegate to this validator for the attack to work, i.e., we assume, without loss of generality, that all participants are wise and all other validators are honest. At time t_2, the adversary deposits b ASSET to the protocol, signalling delegation intent to $\mathcal{V}$. Due to proportional representation, the protocol respects this intent and delegates b ASSET to $\mathcal{V}$. The protocol now holds b delegated ASSET to $\mathcal{V}$. Through this deposit, the adversary obtains $s = \frac{s_0}{b_0}b$ stASSET, and the quoted price remains $\frac{b_0+b}{s_0+s} = \frac{b_0}{s_0}$. Lastly, at time $t_4 > t_2$, validator $\mathcal{V}$ equivocates. This causes a proportion p of the capital b to be slashed. The rest $(1-p)b$ ASSET is returned back to the protocol. However the amount of stASSET circulating in the market remains $s_0+s = s_0+b\frac{s_0}{b_0}$. The new quoted price is now $\frac{b_0+(1-p)b}{s_0+s} = \frac{b_0}{s_0}(1 - p\frac{b}{b_0+b}) < \frac{b_0}{s_0}$.

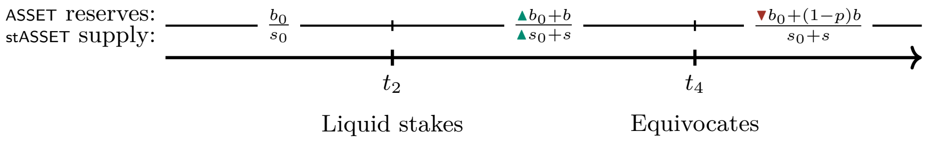


Fig. 1: Timeline of the simplistic attack.

Because stASSET is fungible, *every* stakeholder is negatively affected, proportionally to their holdings. The losses are socialized. As everyone else was wise, this constitutes unfair punishment. ◇

The above attack requires the adversary to expend capital b to cause harm to others, and is irrational. In the remainder of this section, we explore how to make this attack profitable. The profitable attack works for protocols with an *unbonding period* $\delta > 0$. First, we show how to attack without adversarial losses; later, how to profit.

Attack with no initial capital. With a subtle change to the above construction, the attack can be performed without the adversary expending any capital (Figure 2). Before depositing, the adversary acquires a

[7]To do so, she uses a fresh identity to suppress potential suspicions. Most validators have a real-world presence and can be held legally accountable [25, p. 29], but this validator is pseudonymous.

flash loan of b ASSET. At time t_2 she deposits *those* borrowed funds instead of her own. For now, we assume that the borrowing of money is free and there is no cost for the flash loan (we revisit this assumption in Section 7). During equivocation, the adversary does not want to be holding any stASSET of her own, as the price of stASSET is about to drop. She also needs to repay the acquired flash loan. Therefore, after the adversary obtains $s = \frac{s_0}{b_0} b$ stASSET from the deposit, she immediately sells[8] them for $b = \frac{b_0}{s_0} s$ ASSET in the open market, at time t_3. She uses the obtained b ASSET to repay the flash loan. The acts of taking the flash loan, depositing, swapping, and returning the flash loan, can all be performed in a single transaction.

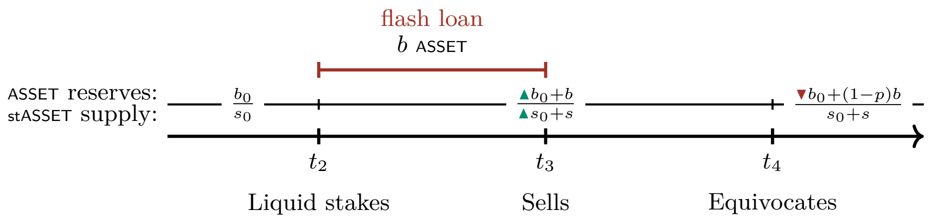


Fig. 2: Timeline of the attack with no initial capital.

The adversary has now managed to add b ASSET delegated to validator $\mathcal{V}$ in the protocol's delegation pool while not currently holding any stASSET. The loss has been averted. At this time, even though the stASSETs have changed hands, the liquid staking protocol cannot redelegate its ASSETs instantly due to $\delta > 0$. The adversary can now equivocate at time t_4 and, as before, cause the price of stASSET to drop.

Making the attack profitable. The profitable version of the attack (Figure 3) works similarly to the above attack, but with some extra steps. As before, the adversary begins by spawning the colluding validator $\mathcal{V}$, deposits b ASSET, obtained by a flash loan, at time t_2, sells the acquired s stASSET to repay the flash loan at time t_3, and equivocates at time t_4.

A small extra trick will allow her to profit. Before forcing the price of stASSET to drop, at time $t_0 < t_4$ the adversary *shorts* stASSET: She takes a loan of z stASSETs and sells them for $b^* = z\frac{b_0}{s_0}$ ASSET in the market. Lastly, at time $t_5 > t_4$ after the price drop, the adversary closes her short position by repaying z stASSET. To recover this amount of

[8]Instead of *selling*, the adversary can *redeem*, but this may incur an unbonding delay, which can be rectified by taking a loan. See Section 7.

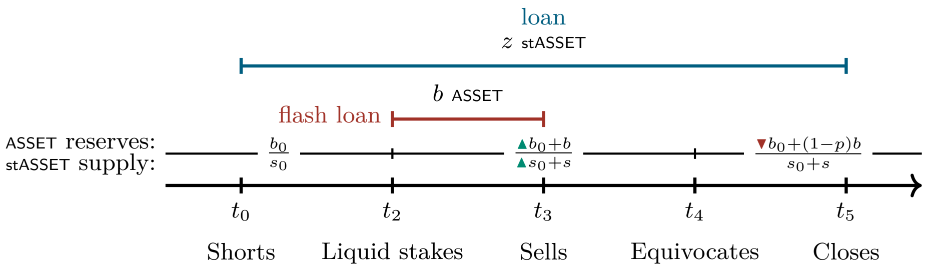


Fig. 3: Timeline of the profitable attack.

stASSET, at time t_5, the adversary *deposits* $b' = \frac{b_0}{s_0}(1 - p\frac{b}{b_0+b})z$ ASSET into the protocol, which allows her to issue the exact required stASSET to be paid back. This concludes the attack.

Her total profits from the attack are $\alpha = b^* - b' = z\frac{b_0}{s_0}p\frac{b}{b_0+b}$ ASSET. A larger short position $z\frac{b_0}{s_0}$ and a larger stASSET price drop percentage $p\frac{b}{b_0+b}$, yields higher profits for the adversary.

So far, we have allowed the adversary to take a loan indiscriminately without any concern for collateral. In practice, loan platforms require a collateral, so the attack impact will be limited by the adversary's initial capital available for collateralization. Let u ASSET be the initial capital of the adversary. If no overcollateralization is required she can obtain a loan of up to $z = u\frac{s_0}{b_0}$ stASSET and then sell them back for $b^* = u$ ASSET. Her profit relative to her initial capital is then $\alpha = up\frac{b}{b_0+b}$ ASSET.

6 Exempt Delegations

Exempt delegations (proposed in LSM [2]) are a mechanism to alleviate the Principal–Agent problem in liquid staking. In this mechanism, an exempt delegation amount c ASSET is associated with each validator. It is a measure of the validator's trustworthiness. The liquid staking protocol is now redesigned to impose restrictions on how much of the protocol's pooled moneys can be delegated to a particular validator based on the validator's exempt delegation. The restriction is parameterized by a factor ϕ (in practice, $\phi > 1$) and is given by the inequality $b \leq \phi c$: Only up to ϕc ASSETs are allowed to be delegated in aggregate by the liquid staking protocol to a validator with a reserve of c ASSET in exempt delegations.

A new validator begins its lifecycle with $c = 0$. They can then raise their own exempt delegation amount by locking aside a chosen amount of ASSET, and marking it as *exempt*. Those assets are delegated to the

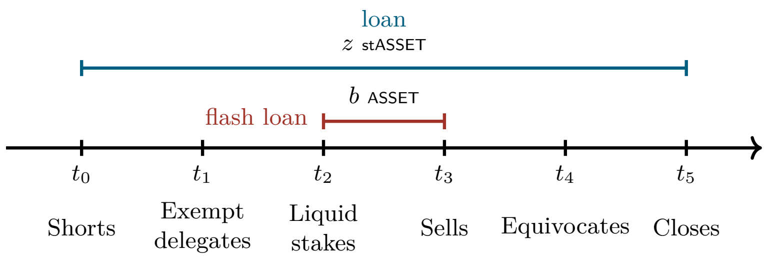


Fig. 4: Timeline of the exempt delegation attack.

validator as usual. However, the *exempt* marking means that those delegated assets cannot be part of the liquid staking protocol pool, but must remain locked aside. Additionally, these specially marked delegations are slashed[9] at a potentially higher rate $q \geq p$. Exempt delegated assets cannot be undelegated in a way that causes a violation of $b \leq \phi c$.

Principals, whether wise or unwise, are not expected to participate in exempt delegations; instead, it is the validator who exempt delegates to themselves (or someone who trusts the validator for extrinsic reasons). This means that, in case of validator misbehavior, the exempt delegation slashing qc is a penalty that only affects the validator.

This raises the cost of the attack described in the previous section. The adversary must first, at time t_1 (where $t_0 < t_1 < t_2$), exempt delegate a sufficient amount $c \geq \frac{b}{\phi}$ ASSET to $\mathcal{V}$ before she can liquid stake b ASSET. Whereas the stASSETs corresponding to those b ASSETs can be, as before, sold at t_3 to separate the agent from the principal, the c amount remains with the agent, holding her financially liable to misbehavior. After equivocation at t_4, in addition to any other costs, the adversary loses qc ASSET. At the conclusion of the attack, the adversary undelegates the unslashed $(1-q)c$ ASSET exempt delegation. The timeline of the refined attack is illustrated in Figure 4 and the respective sequence diagram in Figure 5.

The attack may remain profitable despite exempt delegations. To maximize her shorting leverage, the adversary wants to use all of her initial capital u ASSET to obtain the z stASSET loan. Upon swapping z for b^* ASSET, the adversary can then use part of it as c. The rational adversary should not waste any unnecessary resources on c; therefore, after choosing b she can set $c = \frac{b}{\phi}$. Since the maximum amount the adversary

[9]We abstract some of the irrelevant implementation details here. See Appendix A for how the real protocol works in the context of Cosmos.

exempt delegates cannot be more than her initial capital u ASSET, it must always hold that $b \leq u\phi$.

The profit of the attack now becomes $\alpha = b^* - b' - qc$. Solving for $\frac{d\alpha}{db} = 0$ yields the optimal $b = \sqrt{upb_0\frac{\phi}{q}} - b_0$, which maximizes the adversary's profit, subject to the constraint $0 \leq b \leq u\phi$.

The intuition for why exempt delegations protect the system is that, for the adversary to profit from the short, she must cause a significant shift in the price. The shift in the price is determined by the factor $p\frac{b}{b_0+b}$, so the adversary aims for a large b. But because $b \leq \phi c$ must be respected, this incurs a large penalty $qc = \frac{q}{\phi}b$.

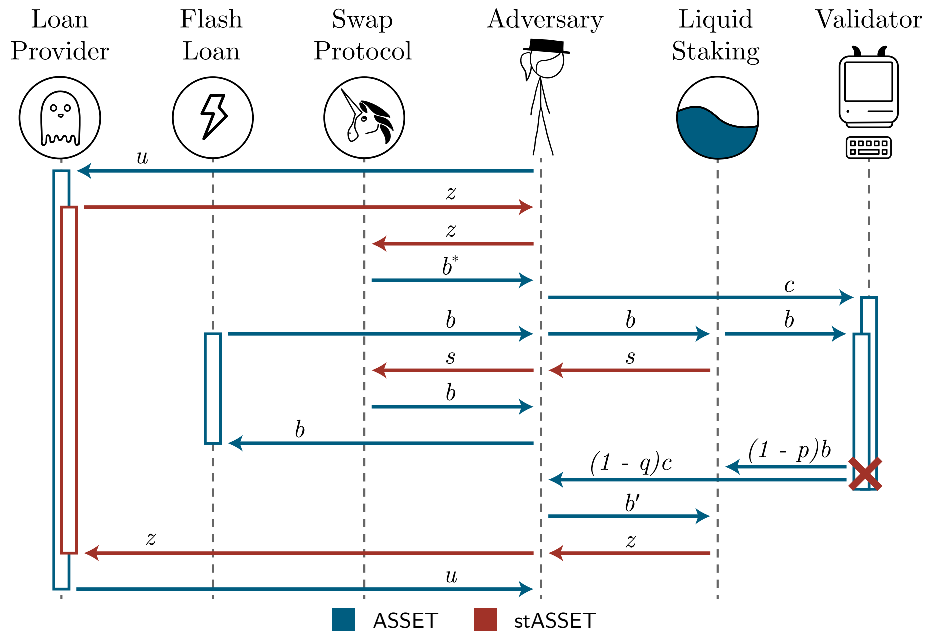


Fig. 5: Sequence diagram of the attack with exempt delegations.

To make the attack irrational, we select the parameter $\frac{\phi}{q}$ such that $\alpha \leq 0$. Plugging in the adversarially optimal value for b in the inequality $\alpha \leq 0$ and solving for $\frac{\phi}{q}$ yields

$$\frac{\phi}{q} \leq \frac{b_0}{pu}\,. \qquad (*)$$

Plugging the values we believe the protocol to operate under into p, b_0 and u, we calculate a secure $\frac{\phi}{q}$ for an assumed maximum adversarial market domination $\frac{u}{b_0}$. In Figure 6, we show the adversarial profit for different values of $\frac{\phi}{q}$ and $\frac{u}{b_0}$. For the figure, we used a slashing rate $p = 0.5$.

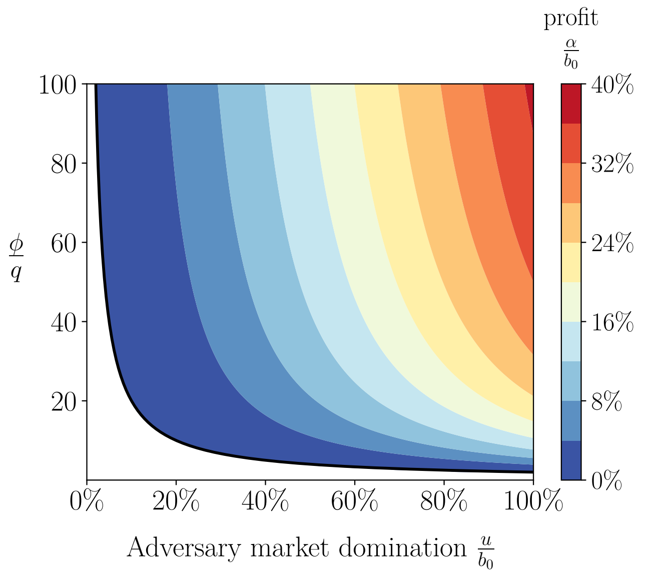


Fig. 6: Adversarial profitability based on market domination $\frac{u}{b_0}$ and parameter $\frac{\phi}{q}$. The white area indicates secure parametrizations.

A higher exempt delegation slashing rate q makes the attack more expensive. This is because a larger portion of the exempt delegation c, holding the adversary accountable, is slashed. A lower exempt delegation factor ϕ also makes the attack more expensive since a larger exempt delegation is required to liquid stake the same b ASSET. Hence, a lower $\frac{\phi}{q}$ makes the protocol less vulnerable to the attack. We recommend always setting $q = 1$ if possible, as this allows for larger values of ϕ, increasing liquidity, without any harm to anyone besides the adversary.

Repeating the attack. If the adversary finds herself in a situation where the attack is profitable, the attack can be repeated in quick succession to siphon off almost all of the money in the liquid staking protocol. This corresponds to moving across the x axis in Figure 6. As the attack repeats, b_0 decreases and u increases as money moves from the reserves of the staking protocol to the hands of the adversary. We conclude that the protocol must be configured with enough margin such that the conditions for the attack never emerge.

Proportional representation VS fair punishment. Proportional representation and fair punishment, as indicated in Section 5, are conflicting properties in liquid staking. Without exempt delegations ($\phi = \infty$), the protocol has full proportional representation, as the principal can signal delegation to any agent of their choice without restriction. There, an adversary can always cause unfair punishment of principals. However, with the introduction of exempt delegations, if we make $\frac{\phi}{q}$ smaller, the pool of available agents to choose from is reduced to only the wealthy amongst them, so proportional representation becomes limited. For a sufficiently small $\frac{\phi}{q}$, the protocol has fair punishment in the rational model.

This concludes the main contribution of our paper. The next sections make some of the financial arguments slightly more precise.

7 Cost of Money

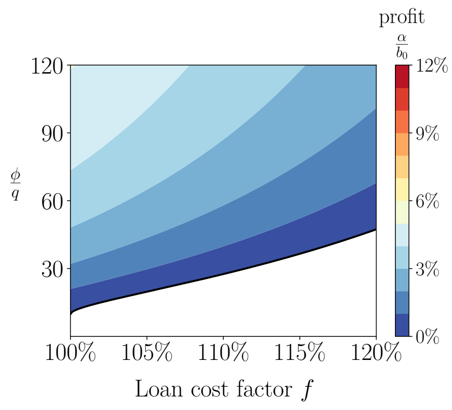
In the previous sections, we assumed that the cost of borrowing money is free. In this section we incorporate the cost (interest and collateral) of borrowing money into our model. For the flash loan, the adversary must pay an extra $\beta_{\mathsf{A}} b$ ASSET in fees upon returning the borrowed b ASSET. The adversary must also pay interest on the z $\mathsf{stASSET}$ she borrowed. The duration of the loan was $\Delta_z = t_5 - t_0$, so the total loan amount to be repaid, including the principal and interest, is $((1 + r_{\mathsf{st}})^{\Delta_z} + \beta_{\mathsf{st}})z$ $\mathsf{stASSET}$. Letting $f = ((1 + r_{\mathsf{st}})^{\Delta_z} + \beta_{\mathsf{st}})$ be the cost factor of the loan, to recover this amount of $\mathsf{stASSET}$ and pay back the loan, the adversary must now deposit $b' = \frac{b_0}{s_0}(1 - p\frac{b}{b_0+b})fz$ ASSET into the protocol instead of $\frac{b_0}{s_0}(1 - p\frac{b}{b_0+b})z$ ASSET like before.

Additionally, loans must be overcollateralized. Let γ_{st} be the collateral ratio of a standard $\mathsf{stASSET}$ loan. The adversary, using her initial capital u as collateral can get a loan of up to $z = \frac{u}{\gamma_{\mathsf{st}}}\frac{s_0}{b_0}$ $\mathsf{stASSET}$ instead of the previous $u\frac{s_0}{b_0}$ $\mathsf{stASSET}$. The loaned z $\mathsf{stASSET}$ are then converted to $b^* = \frac{u}{\gamma_{\mathsf{st}}}$ ASSET. To perform the attack, part of the b^* ASSET is used to exempt delegate c like before, but another part is now used to pay $\beta_{\mathsf{A}} b$ for the flash loan cost. Hence $c + \beta_{\mathsf{A}} b \leq b^*$ and the adversary may only use up to $b \leq \frac{u}{(\frac{1}{\phi}+\beta_{\mathsf{A}})\gamma_{\mathsf{st}}}$ to move the price of $\mathsf{stASSET}$.

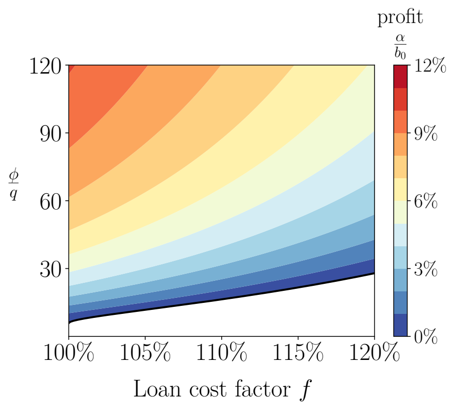
Taking into consideration the cost of borrowing money, the final profit of the attack is now $\alpha = b^* - b' - qc - \beta_{\mathsf{A}} b$ for the new values of b^* and b'. Solving for $\frac{d\alpha}{db} = 0$ gives the optimal $b = \sqrt{\frac{ufpb_0}{(\beta_{\mathsf{A}}+\frac{q}{\phi})\gamma_{\mathsf{st}}}} - b_0$, which maximizes the adversary's profit, subject to the new constraints $0 \leq b \leq \frac{u}{(\frac{1}{\phi}+\beta_{\mathsf{A}})\gamma_{\mathsf{st}}}$. Solving the inequality $\alpha \leq 0$ for $\frac{\phi}{q}$ and plugging in the adversarially optimal value for b yields the new parametrization that renders the protocol secure

$$\frac{\phi}{q} \leq \frac{b_0\gamma_{\mathsf{st}}}{fpu + fu - b_0\beta_{\mathsf{A}}\gamma_{\mathsf{st}} - 2u\sqrt{fp(f-1)} - u}\,.$$

The cost of borrowing money increases when the collateral ratio γ_{st}, the flash loan cost factor β_{A} or the loan cost factor f increase. While the cost of borrowing money goes up, the attack becomes less profitable for the adversary. Thus, the protocol can afford to increase $\frac{\phi}{q}$ and still remain secure. The effect of varying f is illustrated in Figure 7 for an adversary with 30% and 50% market domination $\frac{u}{b_0}$. While f increases, the safe bound of parameter $\frac{\phi}{q}$ can increase with it. The black line indicates where the attack becomes unprofitable for the adversary ($\alpha = 0$). The white area under the black line represents the configuration in which the protocol is secure in the rational model.



(a) Initial capital u is 30% of b_0.



(b) Initial capital u is 50% of b_0.

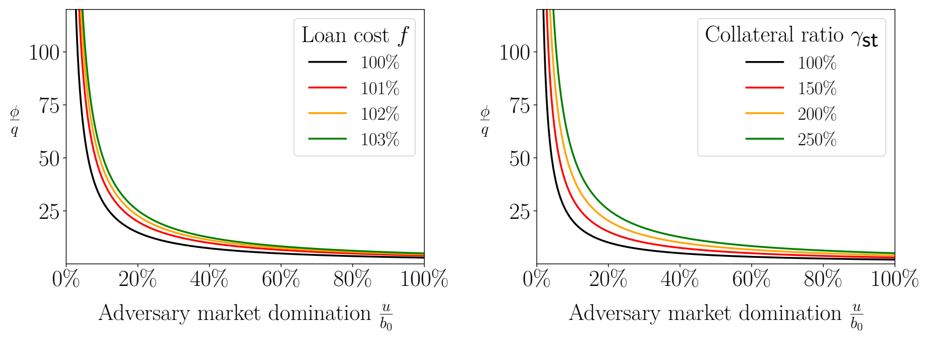
Fig. 7: Cost of borrowing and attack profitability.

In this illustration, we consider a blockchain with slashing percentage $p = 0.5$ and market conditions[10] with ASSET flash loan cost factor $\beta_{\mathsf{A}} =$

[10]ETH and Lido stETH borrowing rates on Aave [18] as of 27 Jan, 2023.

0.09% and collateral ratio $\gamma_{\mathsf{st}} = 146\%$. At the time of writing[10], the annual cost of borrowing stASSET is $r_{\mathsf{st}} = 1.59\%$ (and $\beta_{\mathsf{st}} = 0$ for non-flash loans). If the attack duration is $\Delta_z = 20$ sec, we have $f = 1 + 10^{-8}$. Hence, in practice, $f \simeq 1$, and money borrowing is almost free for short durations.

Free money borrowing makes the adversary more powerful and her attack more profitable. Hence, a safe $\frac{\phi}{q}$ under free borrowing ($f = 100\%$, $\beta_{\mathsf{A}} = 0$, $\gamma_{\mathsf{st}} = 100\%$) is also safe when borrowing is not free. This is illustrated in Figure 8. Safe values of parameter $\frac{\phi}{q}$ for $f = 100\%$, indicated by the black line, are also safe for $f > 100\%$ under any adversarial market domination $\frac{u}{b_0}$. Similarly, safe parametrizations for $\gamma_{\mathsf{st}} = 100\%$ are also safe for $\gamma_{\mathsf{st}} > 100\%$. We deduce that formula $(*)$ suffices for calculating safe protocol parameters.



(a) For varying loan cost f. (b) For varying collateral ratio γ.

Fig. 8: Optimal $\frac{\phi}{q}$ for varying adversarial market domination $\frac{u}{b_0}$ and different market conditions.

Bookkeeping in USD. If the attacker does bookkeeping in a different currency, such as USD, the attack remains profitable. She begins by buying ASSET for USD. At the end of the attack, she sells ASSET for USD. Because the attack concerns a particular liquid staking protocol, and not the whole ASSET network, the price of ASSET will likely not be significantly affected. This attack decreases the market confidence in stASSET, but not in ASSET. In fact, because the attack causes slashing of ASSET, the supply of ASSET is decreased and the price of ASSET with respect to the reference currency may even increase. Lastly, any price fluctuations of ASSET with respect to USD will likely be minor, as the attack has a short duration of a couple of seconds.

The Market price of $\mathsf{stASSET}$. Let us consider the price v of $\mathsf{stASSET}$ denominated in ASSET in the market.

Upper bound. Because the option always exists to mint at a rate of $\frac{s_0}{b_0}$ by depositing, the price of $\mathsf{stASSET}$ denominated in ASSET in a perfectly efficient market is $\frac{b_0}{s_0}$ at maximum. Otherwise, no rational buyer would use the market. Hence, the market rate is $v \leq \frac{b_0}{s_0}$.

Lower bound. There are two options to convert s $\mathsf{stASSET}$ to ASSET: either sell at the market rate to obtain $b = vs$ ASSET, or use the redemption mechanism. Using the redemption mechanism, the ASSETs become available after time δ. Initially, using s $\mathsf{stASSET}$, a redemption is made of $b' = s\frac{b_0}{s_0}$ delegated assets. To get b ASSET immediately (and avoid having to wait for the unbonding period), a loan of b ASSET is taken [25, p. 13] and repaid after duration δ. The amount of ASSET that needs to be paid back, including principal and interest, is $((1 + r_{\mathsf{A}})^{\delta} + \beta_{\mathsf{A}})b$ ASSET. We set this amount to be equal to b', the amount of ASSETs that will be unbonded after δ time. Solving for b, we get $b = s\frac{b_0}{s_0((1+r_{\mathsf{A}})^{\delta}+\beta_{\mathsf{A}})}$. Therefore, in an efficient market $v \geq \frac{b_0}{s_0((1+r_{\mathsf{A}})^{\delta}+\beta_{\mathsf{A}})}$.

We deduce that the bounds for an efficient market of ASSET and $\mathsf{stASSET}$ are

$$\frac{b_0}{s_0((1 + r_{\mathsf{A}})^{\delta} + \beta_{\mathsf{A}})} \leq v \leq \frac{b_0}{s_0}\,.$$

The longer the duration δ, the larger the potential price deviation (c.f. the empirical analysis in *Liquid Staking: Basis Determinants and Price Discovery* [32]). Such loans are available in practice and some protocols [29, §5] [27] even automate this process.

It is also possible for the protocol to use the principle of *remittances* (matching) [29, §5] [27] to match depositing and redeeming parties, so that the redeeming party does not have to incur any unbonding delay. If the redeemed amounts exceed the deposited amounts, some amounts will necessarily incur a delay. The above bounds may be tighter due to a shorter effective unbonding delay.

References

1. Lido: Ethereum Liquid Staking. 2020.
2. Liquidity staking module. Available at: `https://github.com/iqlusioninc/liquidity-staking-module`, 2022.
3. S. Agrawal. Mesh Security. Presentation at Cosmoverse 2022.
4. G. Angeris, H.-T. Kao, R. Chiang, C. Noyes, and T. Chitra. An analysis of uniswap markets. 2021.
5. C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas. Ouroboros Genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 913–930, 2018.
6. I. Bentov, R. Pass, and E. Shi. Snow white: Provably secure proofs of stake. *IACR Cryptology ePrint Archive*, 2016:919, 2016.
7. J. Bowman, V. Modali, and R. Mortaki. Quicksilver Protocol, The Cosmos Liquid Staking Zone. 2022.
8. E. Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph, 2016.
9. E. Buchman and J. Kwon. Cosmos Whitepaper: A Network of Distributed Ledgers. 2016.
10. E. Buchman, J. Kwon, and Z. Milosevic. The latest gossip on BFT consensus. *arXiv preprint arXiv:1807.04938*, 2018.
11. V. Buterin. Slasher: A Punitive Proof-of-Stake Algorithm. Available at: `https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm`, Jan. 2014.
12. V. Buterin. Minimal Slashing Conditions. Available at: `https://medium.com/@VitalikButerin/minimal-slashing-conditions-20f0b500fc6c`, Mar 2017.
13. V. Buterin and V. Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.
14. V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang. Combining GHOST and Casper. *arXiv preprint arXiv:2003.03052*, 2020.
15. Cosmos SDK Team. Cosmos SDK Staking module. Available at: `https://docs.cosmos.network/v0.47/modules/staking`.
16. B. David, P. Gaži, A. Kiayias, and A. Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.
17. EigenLayer Team. EigenLayer: The Restaking Collective. Available at: `https://2039955362-files.gitbook.io/~/files/v0/b/gitbook-x-prod.appspot.com/o/spaces%2FPy2Kmkwju3mPSo9jrKKt%2Fuploads%2F2dCfPgItRfQbX25KriQv%2Fwhitepaper.pdf?alt=media&token=d4d94480-3f01-4e63-bc92-a0658ea37aab`, Jan 2023.
18. E. Frangella and L. Herskind. Aave V3 Technical Paper. Available at: `https://github.com/aave/aave-v3-core/blob/master/techpaper/Aave_V3_Technical_Paper.pdf`, Jan 2022.
19. Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.

20. L. Gudgeon, S. Werner, D. Perez, and W. J. Knottenbelt. DeFi Protocols for Loanable Funds: Interest Rates, Liquidity and Market Efficiency. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 92–112, 2020.
21. M. C. Jensen and W. H. Meckling. Theory of the firm: Managerial behavior, agency costs and ownership structure. *Journal of Financial Economics*, 3(4):305–360, 1976.
22. A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In J. Katz and H. Shacham, editors, *Annual International Cryptology Conference*, volume 10401 of *LNCS*, pages 357–388. Springer, Springer, Aug 2017.
23. S. King and S. Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19(1), 2012.
24. F. Lutsch. Delegation Vouchers: A Design Concept for Liquid Staking Positions. Available at: `https://medium.com/chorus-one/delegation-vouchers-cfb511a29aac`, Jun 2019.
25. F. Lutsch. Liquid Staking Research Report. Available at: `https://mirror.chorus.one/liquid-staking-report.pdf`, June 2020.
26. Z. Manian. ADR ADR-061: Liquid Staking. Available at: `https://docs.cosmos.network/v0.47/architecture/adr-061-liquid-staking`, Sep 2022.
27. Marinade team. Marinade: Unstake Liquidity Pool. Available at: `https://docs.marinade.finance/marinade-protocol/system-overview/unstake-liquidity-pool#order-matching`.
28. J. S. Mill. Of True and False Democracy; Representation of All, and Representation of the Majority Only. 1862.
29. Parallel Team. Parallel Finance White Paper. Available at: `https://docs.parallel.fi/parallel-finance`.
30. pStake Team. pStake: Unlocking Liquidity for Staked Assets. Available at: `https://pstake.finance`.
31. Rocket Pool Team. Rocket Pool. Available at: `https://rocketpool.net`.
32. S. Scharnowski and H. Jahanshahloo. Liquid Staking: Basis Determinants and Price Discovery. *Available at SSRN 4180341*, 2022.
33. A. Smith. *The Wealth of Nations*. 1776.
34. Stride. Stride: Multichain liquid staking. Available at: `https://www.stride.zone`.
35. S. M. Werner, D. Perez, L. Gudgeon, A. Klages-Mundt, D. Harz, and W. J. Knottenbelt. SoK: Decentralized Finance (DeFi). *arXiv preprint arXiv:2101.08778*, 2021.
36. T. Yun, J. Lee, and S. King. ICS-027: Interchain Accounts. Available at: `https://github.com/cosmos/ibc/blob/main/spec/app/ics-027-interchain-accounts/README.md`, Aug 2019.

Acknowledgements

The authors wish to thank Joachim Neu for the suggestion of the repeated attack; Zaki Manian, Aidan Salzmann and Vish Modali for the insights on the history and implementation details of liquid staking; Viraj Nadkarni for the fruitful discussions during the development of our model; Ertem Nusret Tas, Vasiliy Shapovalov, Nikolaos Kamarinakis, Dimitris Karakostas and Aleksis Brezas for reading early versions of the paper and providing valuable feedback; and Harry Karavassilis and Zeta Avarikioti for the creative contributions. We are grateful to the Algorand Centres of Excellence MEGA-ACE programme managed by Algorand Foundation for partially funding this paper. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Algorand Foundation.

Appendix

A Liquid Staking Module (LSM)

In the context of Cosmos, the exempt delegation mechanism is planned to be applied at the consensus layer by the Liquidity Staking Module (LSM) [2]. When this mechanism is used, *assets* are first delegated to a validator by a principal who obtains *delegated assets*, marking them as exempt or non-exempt. In the case of non-exempt delegated assets, these are then *tokenized* into *LSM shares*, representations of delegated assets that are minimally fungible (fungible among the other tokens that were delegated in the same batch to the same validator). These tokenized shares are subject to the exempt delegation constraint $b \leq \phi c$.

The tokenized shares can then be *deposited* into the liquid staking protocol, which issues liquid staking tokens (stASSETs), as usual, in a process termed *refungibilization*. The protocol does not need to delegate further, as it can readily start reaping the delegation rewards (as long as a relevant so-called *LSM record*, which enshrines its holder with the privilege of claiming the rewards associated with a particular delegation, is also transferred along with the tokenized shares). It also does not need to perform further exempt delegation constraint checks as these are enforced by the LSM. When the user *redeems* stASSETs, the protocol may elect to give back tokenized shares instead of ASSETs. Those can then be unwrapped into delegated assets, that can afterwards be undelegated into ASSETs after the relevant unbonding period expires.

Through this mechanism, the exempt delegation c of a validator is a *shared* amount across potentially multiple liquid staking protocols that opt to accept tokenized shares instead of ASSETs directly. The *intent* necessary for proportional representation can be read by the liquid staking protocol by simply looking at the LSM tokenized share records, and no separate voting is necessary when entering the protocol. The factor ϕ is decided not by the liquid staking protocols' governance, but by the governance of the underlying chain. The slashing factor q is applied directly by the chain and not by the liquid staking protocol. In the current[11] LSM design, $q = p$. If a liquid staking protocol participates in multiple chains, the ϕ factors can be different in each chain. In our exposition, we abstract out these implementation details to highlight the economic issues at hand.

[11]Zaki Manian, personal communication, Jan 1st, 2023

B Liquid Staking Protocol Tokenomics

Algorithm 1 The basic tokenomics of all liquid staking protocols.

```
contract liquid-stake extends ERC20
    $b_0 \leftarrow 0$
    $s_0 \leftarrow 0$
    payable function constructor
        require(msg.value $\geq 0$)
        $b_0 \leftarrow$ msg.value
        $s_0 \leftarrow b_0$
        balances[msg.sender] $\leftarrow$ msg.value
    end function
    payable function deposit
        $b \leftarrow$ msg.value
        $s \leftarrow b \cdot \frac{s_0}{b_0}$
        balances[msg.sender] $\leftarrow$ balances[msg.sender] + s
        ▷ Maintain the invariant $\frac{b_0}{s_0} = \frac{b_0+b}{s_0+s}$
        $b_0 \leftarrow b_0 + b$
        $s_0 \leftarrow s_0 + s$                                        ▷ Mint
    end function
    function withdraw($s$)
        require(balances[msg.sender] $\geq s$)
        require($s_0 > s$)
        $b \leftarrow s \cdot \frac{b_0}{s_0}$
        ▷ Maintain the invariant $\frac{b_0}{s_0} = \frac{b_0-b}{s_0-s}$
        $s_0 \leftarrow s_0 - s$                                        ▷ Burn
        $b_0 \leftarrow b_0 - b$
        balances[msg.sender] $\leftarrow$ balances[msg.sender] $- s$
        msg.sender.transfer($b$)
    end function
end contract
```