# Balance: Dynamic Adjustment of Cryptocurrency Deposits

Dominik Harz
d.harz@imperial.ac.uk
Department of Computing
Imperial College London

Lewis Gudgeon
l.gudgeon18@imperial.ac.uk
Department of Computing
Imperial College London

Arthur Gervais
a.gervais@imperial.ac.uk
Department of Computing
Imperial College London

William J. Knottenbelt
wjk@imperial.ac.uk
Department of Computing
Imperial College London

## ABSTRACT

Financial deposits are fundamental to the security of cryptoeconomic protocols as they serve as insurance against potential misbehaviour of agents. However, protocol designers and their agents face a trade-off when choosing the deposit size. While substantial deposits might increase the protocol security, for example by minimising the impact of adversarial behaviour or risks of currency fluctuations, locked-up capital incurs opportunity costs. Moreover, some protocols require over-collateralization in anticipation of future events and malicious intentions of agents. We present Balance, an application-agnostic system that reduces over-collateralization without compromising protocol security. In Balance, malicious agents receive no additional utility for cheating once their deposits are reduced. At the same time, honest and rational agents increase their utilities for behaving honestly as their opportunity costs for the locked-up deposits are reduced. Balance is a round-based mechanism in which agents need to *continuously* perform desired actions. Rather than treating agents' incentives and behaviour as ancillary, we explicitly model agents' utility, proving the conditions for incentive compatibility. Balance improves social welfare given a distribution of honest, rational, and malicious agents. Further, we integrate Balance with a cross-chain interoperability protocol, XCLAIM, reducing deposits by 10% while maintaining the same utility for behaving honestly. Our implementation allows any number of agents to be maintained for at most 55,287 gas (≈ USD 0.07) to update all agents' scores, and at a cost of 54,948 gas (≈ USD 0.07) to update the assignment of all agents to layers.

## CCS CONCEPTS

• **Information systems** → *Reputation systems*; • **Security and privacy** → *Trust frameworks*; *Distributed systems security*; • **Theory of computation** → *Algorithmic game theory*.

## KEYWORDS

deposits; cryptocurrency; reputation; mechanism design; cryptoeconomic protocols


*CCS '19, November 11–15, 2019, London, United Kingdom*

ACM ISBN 978-1-4503-6747-9/19/11.
https://doi.org/10.1145/3319535.3354221

**ACM Reference Format:**
Dominik Harz, Lewis Gudgeon, Arthur Gervais, and William J. Knottenbelt. 2019. Balance: Dynamic Adjustment of Cryptocurrency Deposits. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3319535.3354221

## 1 INTRODUCTION

Capital deposits are a security measure to motivate economically rational agents to behave honestly in cryptoeconomic protocols. Yet, sources of uncertainty, such as exchange rate volatility, can lead to *over-collateralization*, where more capital than necessary is locked-up as a buffer. Moreover, locked capital is expensive: being unable to access funds, agents face an opportunity cost in the form of forgone returns that they could have accrued in alternative investments. Hence, *cryptoeconomic* protocol design plays an important role in balancing security with locked capital costs.

Cryptoeconomic protocols ($\pi$) combine cryptographic primitives and economic theory. In such protocols, a utility maximising agent chooses which action to perform based on self-interest, ignoring the impact of their action on other agents. However, cheating or other malicious behaviour by self-interested agents may harm the protocol as a whole. Therefore, protocols typically involve two types of incentives, e.g. [5, 12, 24, 26–30, 33, 44, 48, 53–55]: *(i)* payouts motivate agents to act in the best interest of the protocol, and *(ii)* deposits prevent malicious actions by punishing misbehaving agents. Yet, choosing the appropriate deposit is challenging due to private information and event-dependency:

**Definition 1** (Private information)**.** A valuation $v_A(\sigma)$ encodes the preference of an agent $A$ for a specific outcome in a protocol depending on an action $\sigma$[1]. We define this information as private, i.e. only known to the agent[2].

**Definition 2** (Event-dependency)**.** Deposits paid in cryptocurrencies can be subject to external events. Event-dependency is caused by events that occur outside of the underlying blockchain and affects the level of required deposits.

An agent $A$ performing an action in a protocol can *(i)* be intrinsically motivated to harm the protocol or *(ii)* receive an external payment to act maliciously due to e.g. bribing [36]. This preference is encoded in $v_A$ and is unknown to the designer of $\pi$. Since such a

[1]For brevity, we write $v_A(\sigma)$ as $v_A$ in the rest of the paper.
[2]See also Chapter 9.4.1 on games with incomplete information in [41].

valuation is *private information*, the required deposit for secure operation is ambiguous. Further, a deposit can be used as insurance for tasks performed outside the underlying ledger, e.g. performing verifiable computations [20, 30, 48], cross-chain assets exchanges [53], or exchanging digital goods [12]. For such usage, the protocol is *event-dependent*, e.g. affected by exchange rate fluctuations. Private information and event-dependency require protocol designers to dynamically calculate the required deposit levels. However, this leads to a dilemma: if the required deposit is too high, the agent is likely to refuse to participate in the protocol. A requesting agent fears that the providing agent has an incentive to cheat if the deposit is too low. In both cases, whether the deposit is set "too low" and "too high" is subjective. Thus, our central research question arises: *how can deposits in over-collateralised protocols be dynamically adjusted when assuming that agents have private information and there exists a dependency on external events?*

We present Balance which provides dynamic adjustment of deposits in over-collateralized protocols, improving agents' financial welfare, without compromising security. Balance is applicable to protocols that *(i)* are implemented on a decentralised ledger, *(ii)* use a deposit to prevent economically rational agents from performing undesired actions, *(iii)* require agents to over-collateralize due to at least one of the two identified sources of uncertainty, and *(iv)* verify interactions with agents through objective specifications $\phi$. Suitable examples for integrating Balance are FairSwap [12], TrueBit [48], NOCUST [27], and XCLAIM [53].

**Security intuition.** Balance allows agents to reduce their capital deposits over a sequence of periodic rounds, while preventing the addition of incentives to act maliciously. In fact, malicious agents that aim to misbehave in a protocol obtain more utility if they do early on, before their deposits are reduced. Conversely, honest and rational agents receive a higher utility by *consistently* acting in the interest of the protocol. Balance achieves this property for protocols which feature over-collateralization by reducing deposits to a lower bound. Above the lower bound, the *additional* utility gained from reducing the deposit is *less* than the opportunity cost for locking-up the deposit. Hence, a malicious agent gains no additional utility from cheating in a later round with a reduced deposit. However, honest and rational agents gain additional utility by reducing the opportunity cost of the locked deposit.

## Contributions

- **Introduction of Balance, a mechanism for deposit reduction:** the mechanism maps agents to layers according to their behaviour, where each layer is associated with a different level of deposit. The higher the layer, the lower the required deposit, mitigating over-collateralization. To the best of our knowledge, this is the first mechanism allowing for the dynamic adjustment of cryptocurrency deposits while maintaining the same level of security.
- **A formal analysis of agents' incentives:** we explicitly model agents' utilities with and without Balance. For agents committed to a protocol, we characterise the conditions on incentive compatibility for Balance. Further, we show that given a distribution of agents over honest, rational, and malicious agent types in a protocol, introducing Balance leads to an increase in social welfare. We show that with a *decrease* of the provided deposit, we can achieve a *greater* incentive for economically rational agents to behave honestly.
- **An integration with XCLAIM**: we integrate Balance with XCLAIM [53], a protocol that allows two cryptocurrencies to be exchanged without trusted intermediaries, using Solidity[3]. The clear exhibition of event dependency and private information in XCLAIM makes it a highly suitable Balance use case. We show that Balance reduces deposits by 10% while maintaining the same level of payoff for complying with the specification of the protocol. The implementation has linear complexity for storing the score of agents depending on the number of layers with a maximum cost of updating a score of a single agent of 55,287 gas (≈ USD 0.07). Further, curating the agents to layers during transition to the next round has constant complexity with 54,948 gas (≈ USD 0.07).

## Structure

In Section 2 we provide a definition for cryptoeconomic protocols including contracts and utilities. We provide an overview of Balance and its security assumptions in Section 3. Next, we introduce the design and functionality of Balance in Section 4. We define a model to prove incentive compatibility and social welfare increase in Section 5. Balance's security is evaluated in Section 6. We apply Balance to XCLAIM in Section 7. Finally, we present related work in Section 8 and conclude in Section 9.

# 2 CRYPTOECONOMIC PROTOCOLS

Cryptoeconomic protocols combine cryptographic primitives and economic theory to create applications on a decentralised ledger. We define a cryptoeconomic protocol as follows.

**Definition 3** (Cryptoeconomic protocol $\pi$)**.** A cryptoeconomic protocol implements agreements—containing publicly known specifications (i.e. the terms of the agreement), verified through cryptographic primitives in a decentralised ledger—between multiple agents. An agreement encodes an incentive mechanism through payments and deposits, seeking to promote honest behaviour.

The states of a protocol are as follows (cf. Appendix A).

(1) *Committed*: an agent $A$ commits to perform an action $\sigma$ that fulfils a specification $\phi$ defined by an agreement. The commitment requires payment of a deposit $D$.
(2) *Executed*: an agent $A$ performs an action $\sigma$. This action either complies with the specification or violates it[4].
(3) *Concluded*: if the action $\sigma$ complies with the specification, the deposit plus any payments is transferred to the agent. Otherwise, the agent's deposit is destroyed or transferred to another agent that suffered from $\sigma$[5].

In this work, we use notation as summarised in Appendix D.

[3]The implementation is available at https://github.com/nud3l/balance.
[4]There are protocols that implement more complex verification steps, e.g. over multiple rounds with accusations. However, at a high-level, the principle remains the same: the protocol implements a method to determine whether or not a specification $\phi$ holds.
[5]Once the protocol is concluded, agents are free to redeem their deposit (if it has not been destroyed or sent to another agent).

## 2.1 Contracts and agreements

A cryptoeconomic protocol $\pi$ is implemented through smart contracts. A contract includes agreements $\mathcal{A}$ in the form of (similar to [46]):

$$\mathcal{A} = \langle \phi, p, \mathcal{D} \rangle \tag{1}$$

$\phi$ represents a condition that needs to be fulfilled by an action and can evaluate to either true or false, i.e. $\phi : \sigma \rightarrow \{0, 1\}$. For example, the FairSwap protocol implements such a mechanism by arithmetic circuits [12]. $p$ are the payments made from a receiving agent $B$ to a performing agent $A$ as stated in the agreement. Further, agents need to provide deposits to *commit* to a protocol. $\mathcal{D}$ is a set of deposits $\{D_1, .., D_n\}$ paid by agents to participate in an agreement implemented in a smart contract. The smart contract acts as an escrow of the deposit. If an agent complies with the specification in an agreement, i.e. $\phi(\sigma) = 1$, the deposit is refunded. If an agent performs an action $\phi(\sigma) = 0$ (or no action), the smart contract will not refund the deposit to the agent.

## 2.2 Action choices

Following Definition 3, an agent commits to a protocol and can only then perform an action. The agent has two action choices.

(1) An agent $A \in P$ performs a *desired action* $\sigma_A \in \Sigma_d$, where $\Sigma_d$ is defined as the set of all actions that evaluate to true with respect to the specification of an agreement, i.e. $\phi(\Sigma_d) = 1$.
(2) An agent $A \in P$ performs an *undesired action* $\sigma_A \in \Sigma_u$, where $\Sigma_u$ is the set of all actions that evaluate to false for the specification of an agreement, i.e. $\phi(\Sigma_u) = 0$. This includes also no action $\sigma_A = \emptyset$ as this results in $\phi(\emptyset) = 0$.

## 2.3 Utility parameters

Each of the two action choices results in a utility for the agent expressed by $u(\sigma)$. The utility of an action depends on five parameters.

(1) Payment $p$: determines how much an agent $B$ has to pay for an action in an agreement $\mathcal{A}$ and how much the performing agent $A$ receives by fulfilling the specification in $\mathcal{A}$.
(2) Cost $c$: captures all costs associated with performing an action. This includes, for example, transaction costs and costs for executing an action $\sigma$ in $\mathcal{A}$.
(3) Deposit $D$: summarises the deposits by agents in $\mathcal{A}$.
(4) Expected future return $\mathrm{E}[rD]$: describes the opportunity cost for locking deposit $D$ within an agreement that could be used in another protocol to earn an interest[6].
(5) Valuation $v$: encodes the *private* preference of an agent for an outcome depending on an action $\sigma$.

The private valuation expresses that an agent $A$ prefers performing an action $\sigma \in \Sigma_u$. Reflecting our security focus, we focus on the worst case, where agents only derive a valuation from performing an undesired action. Assume $A$ is given the task to perform a computation in a protocol like TrueBit [48] and provided a deposit $D$ in the agreement $\mathcal{A}$ to perform a computation that fulfils the specification $\phi$. If $A$ has a positive valuation for $v_A$ that does *not* fulfil the specification and the provided deposit $D$ is smaller than $v_A$, the agent prefers the undesired action. Note that this private valuation is not limited to pecuniary value: it can include the non-pecuniary value that an agent may derive from performing an undesired action. In practice, $v$ might be hard to approximate. We reflect this by introducing agent types depending on the relation between $v$ and the protocol incentives $D$ and $p$ (cf. Section 2.6).

[6] We assume the future return rate is stochastic such that the opportunity cost is not known with certainty. We therefore denote the return rate as the expected return rate, $\mathrm{E}[r]$.

## 2.4 Performing agent utilities

The following are the utilities an agent $A \in P$ *performing* an action[7].

$$u_A(\sigma_A) = \begin{cases} p - c_A - \mathrm{E}[rD_A], & \text{if } \sigma_A \in \Sigma_d \\ v_A - D_A - c_A - \mathrm{E}[rD_A], & \text{if } \sigma_A \in \Sigma_u \end{cases} \tag{2}$$

The agent tries to maximise its utility by choosing an appropriate action. We note two observations from equation (2). First, performing an undesired action becomes a rational choice if $\sigma_{A,i} \in \Sigma_d, \sigma_{A,j} \in \Sigma_u : u(\sigma_{A,i}) < u(\sigma_{A,j})$. In a protocol $\pi$ without Balance agents can enter or leave the game at any time and their past actions are not considered. Hence, the game consists of single, disconnected rounds, i.e. it is a *single-shot game*. Second, executing a desired action depends on the publicly known payout $p$ as well as the agent specific costs $c_A$ caused by performing the action and the expected loss of interest $\mathrm{E}[rD_A]$. An agent will choose this action if it yields higher utility than the other actions[8].

## 2.5 Receiving agent utilities

An agent $B \in P$ *receiving* a certain action has a different utility from the performing agent $A$. This agent has no influence on the performance of the action of agent $A$ if we assume that the payout cannot be changed after commitments are made. We assume that the agent that receives an action is reimbursed with the deposit that is provided by the performing agent. The utilities depending on $A$'s actions are calculated as in (3).

$$u_B(\sigma_A) = \begin{cases} v_B - p - c_B, & \text{if } \sigma_A \in \Sigma_d \\ D_A - v_B - c_B, & \text{if } \sigma_A \in \Sigma_u \end{cases} \tag{3}$$

## 2.6 Agent types and adversaries

The performing agent $A$ influences the utility for both performing and receiving agents. Most importantly, the security of a cryptoeconomic protocol depends on a core assumption: adversaries are economically rational and computationally bounded. Considering Eq. (2), an agent $A$ chooses to perform a desired action if the utility of the desired action is higher than the utility of the undesired action, i.e. $p - c_A - E[rD_A] > v_A - D_A - c_A - E[rD_A]$. Inspired by the BAR model [1], this notation allows us to express three types of agents, from which we can deduct the adversaries.

**Definition 4** (Performing agent types)**.** We categorise performing agents into the following three types.

[7] We omit the valuation $v_A$ for the desired action $\sigma \in \Sigma_d$ as we assume agents do not have an intrinsic or external motivation to comply with the specifications of agreements in $\pi$.
[8] This is a deterministic model where we assume that if an agents "cheats", it loses its deposit. However, one could extend the model to include a probability of the agent being caught if the specification function $\phi$ includes non-determinism.

(1) Type $\mathcal{T}_d$: this type will always perform a desired action as its utility resulting from a desired action is larger than the valuation of an undesired action, i.e. $v_A \ll D_A + p$.
(2) Type $\mathcal{T}_u$: this type will always perform an undesired action as $v_A \gg D_A + p$. By definition, the deposit is never large enough to prevent an undesired action by $A$.
(3) Type $\mathcal{T}_r$: this type is undecided in the single-shot game setting which decision to take as $v \approx D_A + p$.

By this definition, a cryptoeconomic protocol is only able to ensure that economically rational adversaries, i.e. type $\mathcal{T}_r$, and honest participants perform desired actions. Malicious adversaries as expressed by the type $\mathcal{T}_u$ cannot be economically motivated to perform a desired action. These agent types will perform the undesired action as their valuation for the undesired outcome (from the perspective of the protocol designer) is much greater than the economic damage due to the loss of their deposit.

Following Definition 4 the notion of security that we employ in our construction.

**Definition 5** (Security under rational agents)**.** A cryptoeconomic protocol is secure if a economically rational performing agent $A$ would not choose an undesired action because the utility of a desired action is greater, i.e. $u(\Sigma_d) > u(\Sigma_u)$.

## 2.7 Mechanism utilities

Social welfare describes the sum of all agent utilities in a mechanism [51]. If we combine equations (2) and (3), we can calculate the sum of utilities for performing and receiving agents $\{A, B\} \in P$ by considering the actions of *performing* agents $A$. Assuming that every $A$ is matched with at least one $B$, we split $P$ into two groups corresponding to whether the performing agents perform desirable or undesirable actions. $P$ is therefore comprised of a group $\Lambda_D$, for the pairs of agents containing performing agents who perform desirable actions and $\Lambda_U$ for undesirable actions. Thus summing for pair of agents $x \in P$, where there are $X$ pairs, such that $2X = |P|$:

$$u(\sigma_A) = \begin{cases} \sum_{x=1}^{X}(v_{x,B} - c_{x,B} - c_{x,A} - \mathrm{E}[rD_{x,A}]), & \text{if } A \in \Lambda_D \\ \sum_{x=1}^{X}(v_{x,A} - c_{x,A} - v_{x,B} - c_{x,B}), & \text{if } A \in \Lambda_U \end{cases} \quad (4)$$

Hence, if we take all actions within a certain amount of time, we can specify the utility for a protocol $\pi$. Notably, agents acting individually rationally and truthfully after commitment does not imply that the mechanism as a whole maximises social welfare.

# 3 SYSTEM OVERVIEW

We introduce a mechanism called Balance, a verifiable layered curated registry. The idea is to use Balance as an extension to an existing cryptoeconomic protocol, i.e $\pi_{\text{Balance}}$, to control the amount of deposit an agent has to provide to resolve the dilemma of having "too high" or "too low" deposits. The general intuition of the mechanism is displayed in Figure 1 and is as follows:

- *Commitment*: an agent $A$ has to provide a deposit $D_1$ to commit to a protocol encoded by a smart contract. For protocols with private information and event-dependency, the deposit is typically set relatively high[9]. An agent is not trusted at any point in time. However, an agent can reduce its deposit by executing actions that are desired by the protocol.
- *Execution*: agents are tracked by a decentralised registry that stores a ranking of their contributions towards the protocol. Agents are initially assigned to the lowest layer which results in the highest required deposit. When performing desired actions, the agent collects a score within a round.
- *Curation:* if the agent's score is high enough by the end of a round, it moves to the next layer (provided there is a higher layer). The mechanism defines a factor for each layer that is multiplied with the base deposit. This will affect the deposit an agent has to provide. Further, after each round, the score of an agent is reset to retain an incentive to act in the best interest of the protocol. Thereby, this mechanism introduces a *sequential game*.

When an agent is assigned to a lower layer, i.e. having to provide a higher deposit, the agent must ensure that the deposit requirement is met within the current round. Otherwise, agents will move down a layer or be removed from the registry.

## 3.1 Actors

Balance includes three actors with the following roles.

(1) **Performing agent** $A$: an agent $A$ is committed to $\pi_{\text{Balance}}$ and chooses action $\sigma \in \{\Sigma_d, \Sigma_u\}$ evaluating to true or false with respect to the specification $\phi$ of an agreement $\mathcal{A}$.
(2) **Receiving agent** $B$: a receiving agent $B$ requested the performance of an action $\sigma \in \Sigma_d$ as expressed by the specification $\phi$ in the agreement $\mathcal{A}$. The execution of the action is left with an agent $A$.
(3) **Registry $\mathcal{R}$:** the registry is a smart contract implemented on a ledger. The registry keeps a mapping of scores of performing agents in a round $t$, a mapping of agents to layers in a round $t$, and updates the mapping of agents to scores and layers during transition to the next round $t + 1$. The registry has direct access to the result of a performing agent's action $\phi(\sigma) = \{0, 1\}$ to update the agent's score.

Each actor is identified on the ledger using a private/public key pair. We note that agents can take several identities on the blockchain, i.e. Sybil identities are possible. The adjustment of deposit and consequences of performing actions is bound to a public key. An agent $A$ that performs several desired actions to reduce its deposit can have a Sybil identity $A'$. However, the identity $A'$ is in the view of the registry $\mathcal{R}$ a separate agent and hence will not benefit from the reduced deposit of the desired action sequence by identity $A$. Further, we show in Section 6.2, that agents in Balance cannot profit from Sybil attacks.

## 3.2 System properties

Balance achieves the following properties.

(1) **Auditability**: performing and receiving agents $A$ and $B$ have public insight into the amount of deposit $\mathcal{D}$ and layer assignment of performing agents $A \rightarrow \mathcal{L}$ (Section 3.3).
(2) **Reduction of opportunity costs**: Balance reduces the opportunity costs of locked-up deposits $\mathrm{E}[rD_A]$ for performing

[9]For example, in the Dai stable coin, users have to provide at least a 150% deposit to issue Collateralized Debt Positions (CDPs).

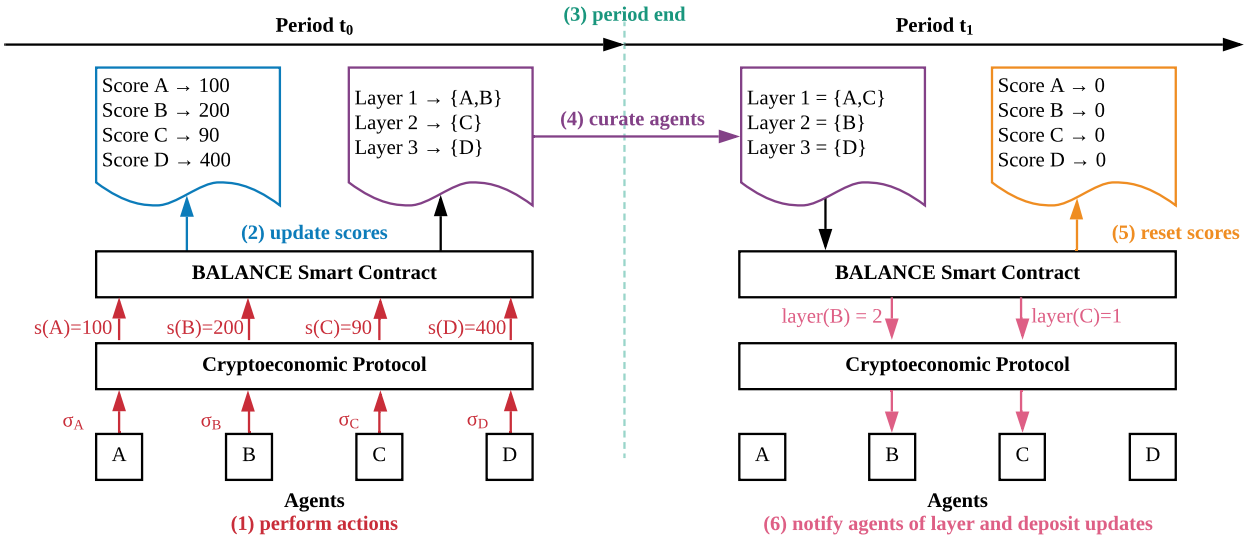



**Figure 1: Balance is implemented as a smart contract and integrated into an existing cryptoeconomic protocol. Agents $A$, $B$, $C$, and $D$ are assigned to layers in the registry representing their deposit level. In step (1) they are performing actions as part of a cryptoeconomic protocol. By performing these actions, they obtain a score that is forwarded to the Balance smart contract. Step (2), is triggered by an agent's actions and updates its scores. Scores reflect a reputation within a round, where the higher the score the better. In step (3), the current round ends. The next interaction of an agent with the cryptoeconomic protocol triggers step (4), the curation of agents to layers. The score of each agent is used to update the mapping of agents to layers, whereby agents can either stay at the same layer, get promoted or demoted by one, or get removed from the registry. Updating the assignment of an agent to a layer is determined by the upper and lower bounds of the layer an agent is currently assigned to. Within that update, step (5) resets the scores of the agent for the next round. Last, in step (6), agents that changed their layer are notified (e.g. via events emitted from a smart contract).**

agent $A$ by reducing deposit depending on the layer assignment $A \rightarrow \mathcal{L}$ (Section 4.1).

(3) **Strategy-proofness**: a performing agent $A$ behaves truthfully with respect to its valuation of an outcome $v_A$ independent of its type $\mathcal{T}$ (Section 5.2, 5.4).

(4) **Transparency**: assuming performing agents $A_1$ and $A_2$ provided the same initial deposit $D$, but are now assigned to different layers such that $D_{A_1} \neq D_{A_2}$, $B$ does not need to choose between $A_1$ and $A_2$ as Balance ensures that each deposit provides the same utility for $A_1$ and $A_2$ to act in the interest of $B$ (Section 5.5).

(5) **Social welfare increasing**: the joint welfare of performing agent $A$ and receiving agent $B$ is increased with Balance (Section 5.6).

(6) **Sybil resistance**: a performing agent $A$ cannot gain additional utility from Balance by creating Sybil identities $A'$ (Section 6.2).

## 3.3 Blockchain model

Balance operates as part of $\pi$ implemented on a blockchain. This blockchain provides a ledger functionality as for example defined in [3, 4, 9, 42]. We assume that the ledger has *finality* as in consensus protocols like [6, 38, 47]. Additionally, we assume given the number of participants $n$ in a consensus protocol, the number of Byzantine faults is $f < n/3$. In cases where a consensus protocol does not provide finality, like Nakamoto consensus [40], we assume that a security parameter $k$ is set such that with high probability the transaction is securely included [18]. Apart from that, we assume that less than 1/3 of miners are malicious [13]. Setting an appropriate parameter for $k$ and having less than 1/3 miners malicious, prevents the impact from selfish-mining attacks.

Aside from the ledger, we assume there is an execution environment and a scripting language available that supports the creation of secure smart contracts to implement agreements $\mathcal{A}$ and the registry $\mathcal{R}$, for example the Ethereum Virtual Machine [49, 50]. We assume that the agreement $\mathcal{A}$ including its specification $\phi$, payments $p$, and deposits $\mathcal{D}$ is implemented by a smart contract SC such that SC $\Leftrightarrow$ $\mathcal{A}$. Further, we require the possibility of creating generic data structures to store the information about agents' scores and layers as part the registry $\mathcal{R}$ and that such information is readable by performing and receiving agents $A$ and $B$. Moreover, we assume that the cryptographic primitives in the ledger are securely implemented.

We make no further assumptions regarding the blockchain. Miners and adversaries are able to re-order transactions, censor transactions, and read the information from transactions before and after they are included in the blockchain.

**Property 1 Auditability**

*The deposit $\mathcal{D}$ and mapping of performing agents to layers $A \to \mathcal{L}$ is publicly available within $\mathcal{R}$.*

### 3.4 Agent assumptions

We make the following assumptions. First, for clarity of exposition, we make the simplification that if an agent performs a single desired action at a certain time, they transition to the higher layer. This is a simplification because in the full specification, agents would need to perform a number of actions in order to change layer. This assumptions strengthens the adversary, as the reduction of deposit requires only a single action. Second, we assume that $\pi_{\text{Balance}}$ unambiguously detects an undesired action, i.e. $\phi(\sigma) = 0$. In this case, the agent takes on a new identity in the protocol and is able to continue to interact with the protocol starting again from the lowest layer. Third, we assume that in the long run, the market for a protocol $\pi$ is perfectly competitive, in the sense that the price (or payments) made in the system will be equal to the marginal cost of each transaction. This assumption provides a higher incentive to perform undesired actions as $p = 0$ *reduces* the utility of performing a desired action as described in Eq. (2). Fourth, we assume that performing an undesired action is free of direct costs[10]. This assumption increases the utility for performing an undesired action as we set $c_A = 0$ on for the undesired action utility.

Overall, our assumptions are pessimistic: we assume agents gain no payments from performing desired actions and can perform undesired actions free of direct cost. Hence, we ensure that incentive compatibility holds even under these pessimistic assumptions (cf. Section 5). In practice, we expect that these assumptions can be modified to include positive payment vectors and costs for executing undesired actions. This would allow more aggressive deposit reductions, however, calculating these are left as future work.

## 4 BALANCE

Extending $\pi$ with Balance adds two properties. First, we introduce the *registry* $\mathcal{R}$ consisting of layers as well as functions to update the score of agents and to curate agents into layers.

Second, we include a *score* $s$ into $\mathcal{A}$ such that:

$$\mathcal{A} = \langle \phi, p, s, \mathcal{D} \rangle \tag{5}$$

$s$ is added to the current agent's score if the specification $\phi$ of the agreement evaluates to true. All details of the agreement are public knowledge.

### 4.1 Integrating layers

A layer is a set of agents that have to provide the same relative deposit. We use Balance to assign agents to layers, making this a verifiable layered curated registry as illustrated in Figure 2. Intuitively, the higher the score of an agent, the higher the layer an agent is assigned to.

We formally define a finite order of layers $\{L_1 \prec \ldots \prec L_\omega\}$, where each layer $L_m \in \mathcal{L}$ has a lower bound $l$, an upper bound $u$ and a deposit factor $f \in \mathcal{R}^+$. We explain the use of the bounds in Section 4.3. The factors determine the ordering of layers. Each layer maps to a deposit level in order, $\mathcal{L} \to \mathcal{D}$. For instance, $L_2$ requires deposit $D_2$.

$$L_i = \langle l, u, f \rangle \tag{6}$$

The set of agents are mapped to the layers, i.e. $P \to \mathcal{L}$. Each agent is only mapped to a single layer at any point in time. The *layer* function returns the layer an agent is currently assigned to. We define the layer($A$) as the current layer of an agent $A$, layer($A$)+1 as the higher layer with a lower deposit factor, and layer($A$) − 1 as the lower layer with a higher deposit factor.

The layers are used to calculate the deposit an agent needs to provide. We define a base deposit $D_{\text{base}}$ (for the lowest layer) and a factor $f$ that are used to calculate $D_m$, where $m \in \mathcal{L}$. Further, the deposit $D_A$ of an agent $A$ is determined by the base deposit and the factor of the layer the agent is currently assigned to.

$$D_m = D_{\text{base}} f_m \tag{7}$$

$$D_A = D_{\text{base}} f_{\text{layer}(A)} \tag{8}$$

Factors are ordered similarly to the layers. Formally, factors are a finite order $\{f_\omega \prec \ldots \prec f_1\}$ where the factor corresponding to the highest layer $L_\omega$ is the *smallest*.

**Property 2 Reduction of opportunity costs**

Balance reduces deposit lock-up.

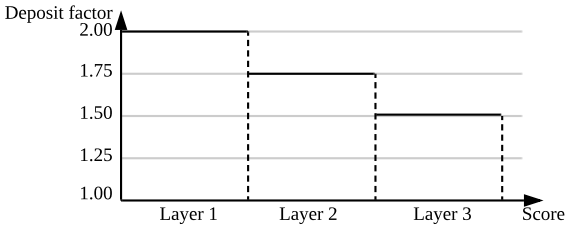



**Figure 2: An intuition of Balance with three layers. The more an agent contributes to the integrated protocol (measured by a score), the further it moves up the layers (from 1 to 3). The higher the layer, the lower the relative deposit needed.**

### 4.2 Updating scores

An agent that just committed to an agreement starts at the lowest layer with the highest deposit factor. An agent can move up or down the layers in the registry depending on its score $s_A$.

Agents can increase their score in any round by performing actions. The score for an action is determined from the agreement, as defined in (5). The *update* function takes an action by an agent and updates the agent's score.

$$\text{update} : \sigma_A \to s_A \tag{9}$$

### 4.3 Curating agents

We define a function *curate*. This function takes as input all agents in an agreement $\mathcal{A}$ and results in a new assignment of agents to layers depending on their scores. If an agent's score is higher than

[10] For example, an agent might go offline and not perform any action in $\mathcal{A}$.

the upper bound of its currently assigned layer, the agent progresses to the higher layer. If an agent's score is lower than the lower bound of its currently assigned layer, the agent falls back to the lower layer.

$$\text{curate}(A) = \begin{cases} \text{layer}(A), & \text{if } l_{\text{layer}(A)} \leq s_A \leq u_{\text{layer}(A)} \\ \text{layer}(A) + 1, & \text{if } u_{\text{layer}(A)} < s_A \\ \text{layer}(A) - 1, & \text{if } s_A < l_{\text{layer}(A)} \end{cases} \tag{10}$$

As layers are a finite order, an agent cannot progress above the highest layer or below the lowest layer. An agent can leave the registry by de-registering. This detail is left open to the actual underlying cryptoeconomic protocol. We note that most protocols require the agent to perform the action they initially committed to. If an agent leaves prematurely, i.e. before the concluded phase, the agent's deposit is usually destroyed. However, an agent can certainly participate in multiple agreements in the same contract. The scores of these multiple agreements are added to the overall score of the agent. An agent already in the highest layer can in the best case remain there. An agent that is currently in the lowest layer and misses its lower bound is excluded from the registry. Further, any agent that performs an undesired action in any agreement part of a smart contract is immediately excluded from the registry and the deposit is destroyed or refunded to another agent.

### 4.4 Time period

According to our blockchain assumptions, Balance requires a minimum number of blocks to consider transactions confirmed depending on the security parameter $k$ [17]. Further, Balance requires a common period $t$ over all its agreements $\mathcal{A}$. We can determine a *minimum* time period by considering the agreement that requires the longest time even in the case of multiple chains (e.g. XCLAIM as in Section 7). However, the time between blocks $\Delta$ is not constant on all chains (e.g. [11]). In the case of a single chain it is sufficient to express the time as a number of blocks, i.e. $t$ depends solely on the security parameter $k$ and the number of transaction $n_{\text{tx}}$ to execute an agreement such that $t = kn_{\text{tx}}$.

If Balance is used in the multi-chain case, we need to include a buffer $b_\Delta$ to account for deviations in $\Delta$. The time period in seconds for *one* chain is expressed by $t_{\text{chain}} = b_\Delta(n_{\text{tx}}k\Delta)$. We can then calculate the total time by summing the time periods for all involved chains within an agreement. Last, the time period can be expressed as a number of blocks by dividing the total time by the average $\Delta$ of the chain on which Balance is implemented.

However, we note that the time period also determines how often agents need to perform desired actions to remain or progress in the layers. Hence, protocol designers need to consider how often agents execute actions within the cryptoeconomic protocol.

## 5 INCENTIVE COMPATIBILITY

The performing agent commits to the protocol by providing a deposit $D_m$. The deposits required by each layer are such that $D_1 > D_2 > \ldots > D_\omega$: Balance rewards the performance of desired actions by letting agents move to the next layer, thereby decreasing the amount of deposit they need to lock up. The utility that the agent gets depends on their choice of action from $\sigma_A \in \{\Sigma_d, \Sigma_u, \Sigma_\emptyset\}$. Given that the performing agent is committed to the protocol, a utility maximising agent will never choose to do nothing ($\sigma_A \in \Sigma_\emptyset$) as they would receive a positive utility from committing either a desired or an undesired action (depending on their valuation, $v_A$).

Figure 3 presents the payoffs that the receiving agent would receive by deciding to perform either a desired or undesired action. The performing agent receives a payoff after each move. If we consider a single-shot game at each round, the agent has to decide between the resulting utility of two actions, i.e. the desired and undesired action.

### 5.1 Action choice

We can express the condition for choosing a desired action for an agent $A$ at layer $m \in [1, \omega]$ with the following Eq.s:

$$v - c_A - \text{E}[rD_m] - D_m < p - c_A - \text{E}[rD_m] \tag{11}$$

This amounts to requiring that

$$v < D_m + p \tag{12}$$

**Theorem 1.** Considering a single-shot game, if the decision to perform a desired action holds true for the highest layer $L_\omega$ then it will also hold true for all previous layers.

Proof. If the agent in the highest layer decides to perform a desired action, Eq. (12) holds true. As $D_\omega < \ldots < D_1$, the utility of the agent at layer $L_\omega$ is higher than at the previous layers. Hence, a rational agent deciding to perform a desired action at layer $L_\omega$ makes the same decision in the previous layers. This also allows it to reach the highest layer. □

### 5.2 Incentive compatibility for types $\mathcal{T}_d$ and $\mathcal{T}_u$

We now extend the analysis to consider the intertemporal payoffs resulting from the agents' decisions. If an agent always does a desired action, i.e. an agent of type $\mathcal{T}_d$, they get a payoff of $p - c_A - \text{E}[rD_1]$ in the first period. In the second period, invoking a discount factor $0 < \delta < 1$ and a real rate of interest $r$, agents receive a payoff of $(\frac{\delta}{1+r})(p - c_A - \text{E}[rD_2])$ in today's terms. The payoff is discounted reflecting the fact that future payoffs are less valuable than present ones, because a payoff today would receive a rate of return $r$. This continues until the agent is in the highest layer, $L_\omega$. In the highest layer, agents would receive a payoff of $(\frac{\delta}{1+r})^{\omega-1}(p - c_A - \text{E}[rD_\omega])$ in today's terms. Thus summing to infinity yields the following payoff $u(\sigma_d)$ for agents who always performs a desired action each round:

$$\begin{aligned} u(\sigma_d) = &\sum_{t=0}^{\omega-1} (\frac{\delta}{1+r})^t (p_A - c_A - \text{E}[rD_{t+1}]) \\ &+ \sum_{t=\omega}^{\infty} (\frac{\delta}{1+r})^t (p_A - c_A - \text{E}[rD_\omega]) \end{aligned} \tag{13}$$

Where convergence of the latter term requires that $|(\frac{\delta}{1+r})| < 1$.

If we assume that an agent is of type $\mathcal{T}_u$, the agent will indefinitely perform an undesired action: at the lowest layer $L_1$, the utility of the undesired action is higher than the utility of the desired action. Hence, this agent's utility is as follows.

$$u(\sigma_u) = \sum_{t=0}^{\infty} (\frac{\delta}{1+r})^t (v - c_A - \text{E}[rD_1] - D_1) \tag{14}$$

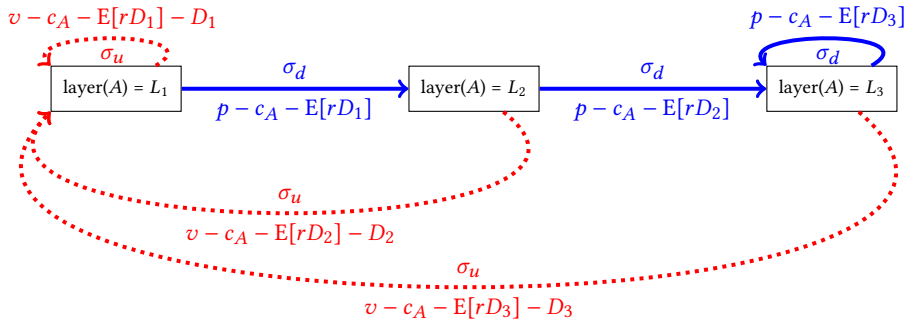



**Figure 3: Layer transitions for agent $A$ in a simplified 3-layer system. The red dashed lines correspond to an undesirable action($\sigma_u$); the black solid branches correspond to a desirable action ($\sigma_d$). The agent starts in $L_1$. If the agent chooses $\sigma_d$ they are moved into layer $L_2$, receiving payoff $p - c_A - \mathrm{E}[rD_1]$, thus reducing the agent's deposit to $D_2$. In contrast, if the agent chooses $\sigma_u$ in $L_1$ they receive payoff $v - c_A - \mathrm{E}[rD_1] - D_1$ and are removed from the registry. Assuming agents can rejoin the protocol, this is equivalent to being returned to $L_1$. This process is repeated analogously in $L_2$ and $L_3$.**

Assuming that $|(\frac{\delta}{1+r})| < 1$, such that the series of payoff converges, this can be expressed as:

$$u(\sigma_u) = \frac{(1+r)(v - c_A - \mathrm{E}[rD_1]) - D_1)}{1 + r - \delta} \tag{15}$$

**Theorem 2.** Agents of types $\mathcal{T}_d$ and $\mathcal{T}_u$ in Balance act individually rational and truthful to their valuation.

Proof. By Definition 4, agents that are committed to the protocol cannot increase their utility by hiding their true valuations: the protocol is strategy-proof. The utility of each agent in an infinite time horizon is given by Eq.s (13) and (14), respectively. Thus, Balance is incentive compatible for types $\mathcal{T}_d$ and $\mathcal{T}_u$. □

**Property 3 Strategy-proofness**

*Balance offers strategy-proofness for types $\mathcal{T}_d$ and $\mathcal{T}_u$.*

## 5.3 Decision boundary for type $\mathcal{T}_r$

The rational agent of type $\mathcal{T}_r$ is indifferent between performing an undesired or desired action in the highest layer $L$. Hence, consider the following strategy $S_{\text{layer-cycling}}$ for an agent:

**Definition 6** (Layer-Cycling)**.** The agent performs a desired action in the first $\omega - 1$ layers but commits an undesired action once in layer $\omega$.

We index these cycles with $\gamma$, such $0 \leq \gamma \leq \infty$. For instance, the first iteration of this strategy is denoted by $\gamma = 0$; the next iteration by $\gamma = 1$. If we consider a one-shot game, whether the agent optimally commits a desired or undesired action is determined by (12). However, this does not consider the fact that once the agent performs an undesired action they need to work back up through the layers, with greater opportunity costs for their locked-up deposit, by performing $\omega - 1$ desired actions to reach the highest layer again. Thus, we provide a framework which captures the intertemporal trade-off that agents face in Balance in committing an undesired action once in the highest layer.

For each cycle complete $\gamma$ starting at $t = 0$, the payoff of such a strategy can be expressed as

$$u_A = v - c_A - D_\omega - \mathrm{E}[rD_\omega] + \sum_{t=1}^{\omega-1} (\frac{\delta}{1+r})^t \{p - c_A - \mathrm{E}[rD_t]\} \tag{16}$$

where $D_\omega$ denotes the deposit made in the highest layer $L_\omega$. The corresponding expression for infinite cycles is provided in Eq. (27) in Appendix B.1.

Now consider the stream of payoffs to an agent who always commits a desired action once in the highest layer. We can group the stream of payoffs into sections of size $\omega$, to correspond to the number of rounds required for a player playing $S_{\text{layer-cycling}}$ to complete one full cycle. Thus the equivalent utility for performing desired actions for a single cycle $\gamma$ is as follows:

$$u_A = \sum_{t=0}^{\omega-1} (\frac{\delta}{1+r})^t \{p - c_A - \mathrm{E}[rD_\omega]\} \tag{17}$$

Similarly, the corresponding expression for infinite cycles is provided in Eq. (25) in Appendix B.1. Lemma 2 in the same Appendix shows that comparing the payoffs of a Eq. (17) and (16) over infinite cycle repetitions $\gamma \to \infty$ is the same as comparing those equations for a single cycle. Thus, equating (27) and (25) provides us with an expression for the value of $v$ at the boundary.

$$v = p + D_\omega + \sum_{t=1}^{\omega-1} (\frac{\delta}{1+r})^t (\mathrm{E}[rD_t] - \mathrm{E}[rD_\omega]) \tag{18}$$

Eq. (18) captures several salient aspects of Balance. Firstly, the boundary valuation depends on the levels of $p$ and $D_\omega$. An increase in the payment that the agent receives or an increase in the deposit in the final layer serve to increase the value of $v$ required such that the agent is indifferent between committing a desired or undesired action. The minimum valuation $v$ also depends on the difference in opportunity costs (forgone returns) on deposits, with the opportunity cost of funds in lower layers contributing more: $(\frac{\delta}{1+r})^t$ declines as $t$ increases.

As detailed in Appendix B.2, explicitly invoking deposit ratios between the layers, such that $D_\omega = f_\omega D_{\text{base}}$ and $D_t = f_t D_{\text{base}}$, and setting $p = 0$, enables (18) to be rewritten as

$$v = f_\omega D_{\text{base}} + \sum_{t=1}^{\omega-1} \left(\frac{\delta}{1+r}\right)^t \left(\mathrm{E}[rD_{\text{base}}(f_t - f_\omega)]\right) \tag{19}$$

Note that on the assumption that an agent can advance a single layer $m$ per time period $t$, to make this relationship between layers and time explicit we swap $m$ for $t$. Rearranging this expression

for $f_\omega$, and assuming a linear relationship between the smallest factor $f_\omega$ and the other factors $f_m$ (see Appendix B.1) enables the indifference boundary to be plotted - see Figure 4.

## 5.4 Incentive compatibility region for type $\mathcal{T}_r$

We can characterise the incentive compatibility region for the rational agent as follows.

**Theorem 3.** Provided $f_\omega > f_{\bar{\omega}}$, performing $\sigma_d$ at the highest layer (and not $S_{\text{layer-cycling}}$) is incentive compatible for type $\mathcal{T}_r$.

Proof. We set $D_{\text{base}}$ as a relative value where $D_{\text{base}} = 1$. Assuming a linear relationship between the smallest factor $f_\omega$ and the other factors $f_t$, as detailed in Appendix B.2 (35), enables us to express the boundary condition for incentive compatibility for agents of type $\mathcal{T}_r$ $f_{\bar{\omega}}$ as follows:

$$f_{\bar{\omega}} = \frac{v\omega - v - \sum_{t=1}^{\omega-1} \left(\frac{\delta}{1+r}\right)^t \mathrm{E}[r f_1(\omega - t)]}{\omega - 1 - \sum_{t=1}^{\omega-1} \left(\frac{\delta}{1+r}\right)^t \mathrm{E}[r(\omega - t)]} \tag{20}$$

Thus, provided $f_\omega > f_{\bar{\omega}}$, incentive compatibility holds for $\mathcal{T}_r$. □

**Property 3 Strategy-proofness**

*Balance offers strategy-proofness for type $\mathcal{T}_r$.*

## 5.5 Transparency

Balance seeks to be transparent to receiving agents $B$. $B$ does not have to choose between performing agents $A$ based on their layer assignment, but selects $A$ as without Balance.

**Theorem 4.** Given an initial deposit $D_1$ for a performing agent $A$ of type $\mathcal{T}_r$ and Balance with $\omega$ layers, $A$ cannot decrease its utility for performing a desired action and does not have a higher utility for an undesired action assuming $f_\omega > f_{\bar{\omega}}$.

Proof. If $A \to L_m$ at a point $t = 0$, we consider two cases: First, $A$ can consider an *infinite* horizon. By Theorem 3, we know that if $f_\omega > f_{\bar{\omega}}$, the utility for performing a desired action is greater than the undesired action for $A$ starting at $L_1$. By the infinite time horizon the starting layer $L_m$ becomes irrelevant. Hence, $A$ has the same utility for performing a desired action at layer $L_m$ as at $L_1$.

Second, by Theorem 1, if agent $A$ performs a desired action at the highest layer, $A$ also performs desired action at previous layers. Further, the utility at the lowest layer $L_1$ is smaller than the utility at the subsequent layers. If $A$ only considers a single time step, and $A$ has performed a desired action at layer $L_1$, $A$ can only increase its utility by performing a desired action in the next layer. □

**Property 4 Transparency**

*Balance is transparent to a receiving agent $B$.*

## 5.6 Comparison and social welfare

Next, we show that adding Balance increases social welfare for agents who perform desired actions.

**Theorem 5.** For a distribution of agent types $p(x)$, where $x \in \{\mathcal{T}_d, \mathcal{T}_u, \mathcal{T}_r\}$, adding Balance increases social welfare.

Proof. As in (4), if performing agents perform desired actions, the utility is equal to $\sum_{x=1}^{X}(v_{x,B} - c_{x,B} - c_{x,A} - \mathrm{E}[rD_{x,A}])$, where $x$ is a pair of agents in the set of pairs $X$. without Balance, each pair is required to have deposit $D_1$ each period. Over a cycle length $t = \omega$, the total utility for all pairs of agents over the cycle is given by

$$\sum_{t=0}^{\omega-1} \sum_{x=1}^{X} (\delta/(1+r))^t ((v_{x,B} - c_{x,B} - c_{x,A} - \mathrm{E}[rD_{x,1}])) \tag{21}$$

In contrast, with Balance, total welfare is given by

$$\sum_{t=0}^{\omega-1} \sum_{x=1}^{X} (\delta/(1+r))^t ((v_{x,B} - c_{x,B} - c_{x,A} - \mathrm{E}[rD_{x,t+1}])) \tag{22}$$

For agents performing desired actions, Balance improves welfare when (22) is greater than (21). Yet, since in all but $t = 0$, the opportunity cost term ($[rD_{x,t+1}]$) is smaller in the latter equation, with Balance permitting agents to post smaller deposits at higher layers, (22) must be the larger. Therefore, Balance improves social welfare for agents performing desired actions. For agents who always perform undesired actions, social welfare is the same with Balance as without it. □

**Property 5 Social welfare increasing**

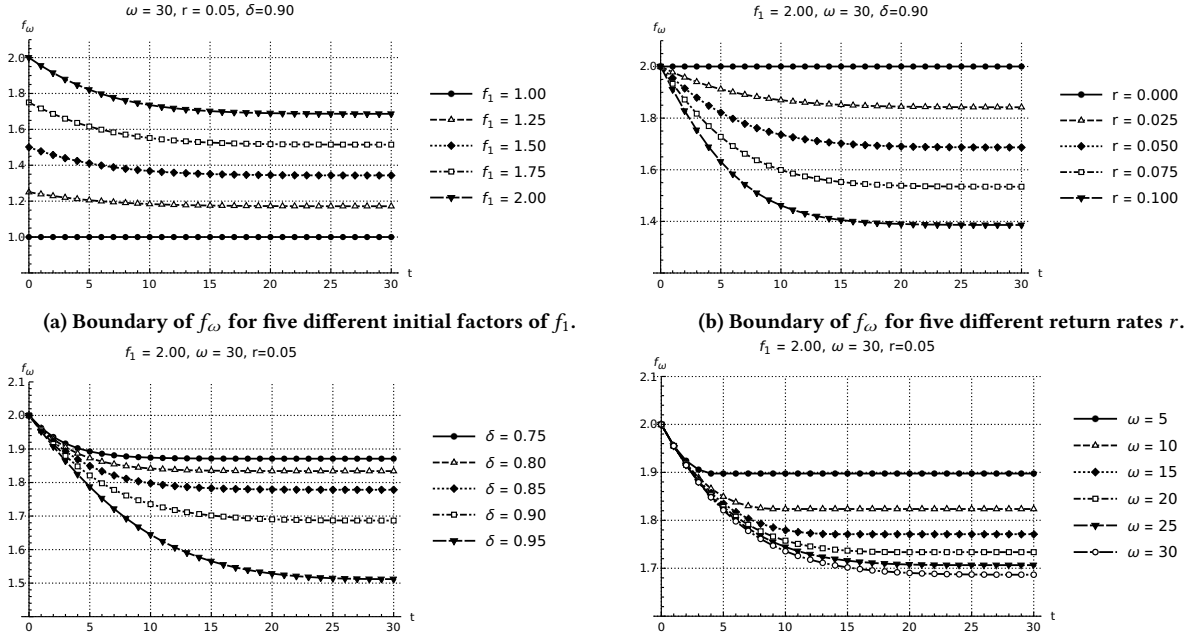
*Balance is social welfare increasing.*

## 5.7 Parameter behaviour

Balance has four main parameters that influences the boundary $f_{\bar{\omega}}$ over which $f_\omega$ must be set to ensure that agents of type $\mathcal{T}_r$ perform desired actions. Figure 4 demonstrates how $f_\omega$, the factor corresponding to the highest layer boundary, changes as the parameters $f_1$, $r$, $\delta$ and $\omega$ vary. The purpose of this section is to provide guidance on the behaviour of for a given implementation when the parameter values are varied. The values of $f_1$ and $\omega$ can be configured by the designer of the protocol $\pi$, whereas $\delta$ and $r$ are external factors that require careful consideration.

- Figure 4 (a) shows that, all else equal, $\forall t$, opting for a higher value of $f_1$ results in a relatively higher value of $f_\omega$ than lower values of $f_1$. Starting with a higher layer factor $f_1$ permits a greater reduction in $f_\omega$ over time relative to the starting value of $f_\omega$ at $t = 0$, but results in a higher absolute value of $f_\omega$ relative to lower values of $f_1$.
- Figure 4 (b) shows that, all else equal, $\forall t$, an increase in $r$ results in a lower value of $f_\omega$. To offset a higher interest rate, $f_\omega$ has to fall more quickly since a higher interest rate results in higher opportunity costs of deposits.
- Figure 4 (c) shows that, all else equal, $\forall t$, an increase in the discount factor $\delta$ allows $f_\omega$ to be relatively lower. As the discount factor increases, such that future utility is more valuable today, the factor $f_\omega$ corresponding to the highest layer can be reduced.
- Figure 4 (d) shows that, all else equal, $\forall t$, using more layers $\omega$ permits a greater reduction in $f_w$ through time. However, this increase is not linear. Using Eq. 20, we can show that if $\omega \to \infty$, the reduction of deposit has a lower bound.

We note an interesting property for protocol designers when choosing parameters for Balance. If the required deposit factor at each layer are set to the boundary $f_{\bar{\omega}}$, $\pi_{\text{Balance}}$ enjoys the same




**(a) Boundary of $f_\omega$ for five different initial factors of $f_1$.**

**(b) Boundary of $f_\omega$ for five different return rates $r$.**

**(c) Boundary of $f_\omega$ for five different discount factors $\delta$.**

**(d) Boundary of $f_\omega$ for six different numbers of layers $\omega$.**

**Figure 4: The boundary $f_{\bar{\omega}}$ depending on the parameters of $f_1$, $\omega$, $r$, and $\delta$. If $f_\omega$ is chosen above $f_{\bar{\omega}}$, an economically rational agent has a higher utility to choose a desired action. Below $f_{\bar{\omega}}$, the agent chooses an undesired action.**

level of security. However, if a designer sets the deposit factors for layers $m$ such that $f_1 > f_m > f_{\bar{\omega}}$, $\pi_{\text{BALANCE}}$ has a relatively higher level of security with respect to economically rational agents *and* increases social welfare.

## 6 SECURITY ARGUMENTS

We discuss a range of attack strategies against BALANCE and how to mitigate them.

### 6.1 Single-shot attack

In single-shot attack strategy, $S_{\text{single-shot}}$, the only objective of agent $A$ is to attack $B$ through committing an undesired action. $A$ would then consider how to execute such an attack with least cost. In particular, $A$ may be able to reduce the cost of attack by playing a modification of $S_{\text{layer-cycling}}$ where $A$ progresses through the layers and performs the undesired action at the highest layer. We show that if $f_\omega$ is set above a bound $f_{\bar{\omega}}$, $A$ would not gain additional utility by waiting until it is in the highest layer before performing the attack.

**Lemma 1.** If $A$ plays $S_{\text{single-shot}}$ it cannot gain additional utility given that the $f_\omega$ is set above $f_{\bar{\omega}}$.

PROOF. This strategy has two implications. First, $A$ performs $\omega - 1$ desired actions that contribute to the social welfare of the protocol. Second, $A$ increases its utility at $t = \omega$ for the undesired action with being punished with the smaller $D_\omega$ instead of $D_1$. However, for each $\omega - 1$ round, $A$ has incurred an opportunity cost. As shown in Lemma 2 in Appendix B.1, $S_{\text{single-shot}}$ results in the same utility expression and boundary for $f_\omega$ as $S_{\text{layer-cycling}}$. Hence, if $f_\omega$ is set to be greater than $f_{\bar{\omega}}$, $A$ does not gain additional utility from playing $S_{\text{single-shot}}$. $A$ should commit the undesired action at $D_1$ if $A$ intends to perform a undesired action. Otherwise, $A$ should perform the desired action. □

BALANCE provides in the single-shot attack the same security as protocols without BALANCE, however, offers desired agent types to reduce their deposit over time.

### 6.2 Reputation boosting

An agent $A$ could create Sybil identities to request performance of actions and fulfil them itself. This results in improving the agent's score and reducing its deposit. We denote this $S_{\text{boosting}}$, a common attack vector in reputation like systems [23, 25].

**Theorem 6.** Reputation boosting $S_{\text{boosting}}$ is not rational in BALANCE if the cost of this strategy, $c(S_{\text{boosting}})$, is higher than the expected saving from the reduction of deposit.

PROOF. We consider a scenario where $A$ can take the role of a receiving agent $B$. Further, we assume that the cost for requesting a service within an agreement $\mathcal{A}$ by $c(S_{\text{boosting}}) = c_B$. Without loss of generality, we assume that an instance of BALANCE includes two layers, $L_1$ and $L_2$. $A$ performs a desired action by fulfilling its own request (per role $B$). In the next step, $A$ can either execute a desired or an undesired action. In either case, the increase of utility is determined by the reduction in opportunity cost at each layer, i.e. $(\frac{\delta}{1+r})\text{E}[rD_2] - \text{E}[rD_1]$. A sufficient condition for $S_{\text{boosting}}$ to not be a rational strategy is:

$$c_B > \mathrm{E}[rD_1] - \left(\frac{\delta}{1+r}\right)\mathrm{E}[rD_2] \tag{23}$$

Therefore, we prevent $S_{\text{boosting}}$ by limiting the maximum deposit that can be provided by a single identity of $A$ to $D_{\max}$. If $A$ wants to exceed $D_{\max}$, it needs to commit to an agreement with an additional identity $A'$. □

The *commit* stage of cryptoeconomic protocols typically requires an on-chain transaction which incurs a cost, e.g. [27, 44]. If requesting is *free* in a protocol, boosting would be rational. Thus, to prevent *griefing* by $B$, cryptoeconomic protocols typically require a small amount of deposit [12] as friction. Since $B$ also has an expected loss of interest on this deposit, this incurs a cost expressed by $\mathrm{E}[rD_B]$.

**Property 6 Sybil resistance**

*Balance is resistant to the creation of Sybil identities.*

### 6.3 Action delay

In strategy $S_{\text{delay}}$, $A$, who has already performed sufficient desired actions to move up a layer, delays a desired action from $t$ until $t+1$. In the new round, with a reset score, the delayed action would count towards the new score. If there are significantly more requesting agents $B$ than performing agents $A$, a backlog of requests would occur. We propose two remedies. First, $A$ could be required to perform the actions within a time-limit enforced by the cryptoeconomic protocol (e.g. HTLCs). Second, Balance could be modified to use *dynamic time rounds*. Dynamic time could be based on the number of *active* performing agents $A$ and the number of requests by receiving agents $B$. For instance, the next round could be started after a certain fraction (or all) of the performing agents $A$ could have at least fulfilled one request by $B$.

### 6.4 Competition and cooperation

Performing agents compete to fulfil agreements, to receive payments and reduce their deposit requirements. However, it is possible that the number of requests is insufficient for $A$ to maintain its assignment to the high layers of Balance. In such cases, two strategies for $A$ are as follows.

First, $A$ can be *non-cooperative* and play a strategy $S_{\text{compete}}$ in which $A$ (i) tries to fulfil requests as fast as possible by e.g. detecting request transaction early through memory pool sniffing and (ii) actively prevent other performing agents from fulfilling requests by e.g. executing eclipse attacks [19, 21].

Second, $A$ can be *cooperative*, executing a strategy $S_{\text{cooperate}}$ to collaborate with other agents. $A$ could form groups with shared interests through e.g. multi-signature wallets with other agents. A set of agents can act as a single entity to reduce their pooled deposit and fulfil agreements as a whole. This concept is similarly applied to layer-one protocols as so called mining pools. Lewenberg et al. formulate an analysis for cooperative games in the Bitcoin mining network [31]. A cooperative game for cryptoeconomic protocols would consider the number of performing agents $A$, their overall deposit $\mathcal{D}$, and a likelihood of having open requests by $B$.

In both cases, Balance does not change the possible strategies in the cryptoeconomic protocol. Agents can play either strategies with and without Balance being present.

## 7 APPLICATION

The objective of Balance is to reduce deposits up to the boundary $f_{\bar{\omega}}$. Integrating Balance is therefore an exercise in determining external factors and choosing the parameters such that $f_{\bar{\omega}}$ can be optimised. This process includes the following steps:

(1) Determining the return rate $r$ includes finding an approximate value what agents could earn by participating in other protocols than $\pi$.
(2) Determining the discount factor $\delta$ comprises of finding the discount agents apply to future returns.
(3) Choosing the scores $s$ in $\mathcal{A}$ needs careful consideration of the agreement. Reducing deposits introduces a positive incentive for agents. Therefore, only agreements that already have a positive incentive, e.g. payments, should be associated with a positive score.
(4) Choosing a period length $t$ depends on how often a protocol designer expects agents to interact with the protocol. Since agents need to continuously perform desired actions, agents should have the chance to do so by giving them a long enough time period.
(5) Choosing the number of layers $\omega$ determines (i) the overall reduction of deposit possible and (ii) how many time periods it takes to reach the lowest deposit.
(6) Choosing the initial factor $f_1$ is a trade-off between the "buffer" of over-collateralization and the amount of deposit reduction. The higher $f_1$, the higher the reduction possible. However, also the higher the absolute deposit.

We provide more detail on determining external factors $r$ and $\delta$ in Appendix C.2. However, a thorough study of parameters specifically for decentralised ledgers is left as future work.

In the following we describe in detail the application of Balance to the XCLAIM protocol. XCLAIM is a generalised protocol that allows exchange of cryptocurrencies across different blockchains without a trusted intermediary [53]. The protocol employs third parties that provide a deposit to be eligible to participate in the protocol. Further, XCLAIM reduces the cost of atomic cross-currency swaps in comparison to atomic-swaps realised with hashed time-lock contracts (HTLC). We choose XCLAIM as an example of how to apply Balance as it is subject to exchange rate risk (event-dependency) and heterogeneous valuations of individual cryptocurrencies (private information).

### 7.1 XCLAIM protocol

XCLAIM is divided into three sub-protocols. First, the *issue protocol* enables creation of cryptocurrency-based assets (CbAs) from a backing chain onto an issuing chain. Second, in the *swap protocol*, senders and receivers on the issuing chain can exchange the issued CbAs. Finally, in the *redeem protocol*, a redeemer exchanges his CbA for the original coin on the *backing* chain. For example, Alice could issue Bitcoin-backed tokens on the Ethereum chain. Alice could then send her Bitcoin-backed tokens to Bob on Ethereum in exchange for Ether. Bob can take his Bitcoin-backed tokens to receive the equivalent amount of Bitcoins on Bitcoin. XCLAIM includes six roles.

(1) *CbA Requester*: creates a backed token $i(b)$ on an issuing chain from locking a coin $b$ on the backing chain.

(2) *CbA Sender*: sends a backed token $i(b)$ on the issuing chain.
(3) *CbA Receiver*: receives a backed token $i(b)$ on the issuing chain.
(4) *CbA Redeemer*: destroys the backed token $i(b)$ on the issuing chain to redeem the coin $b$ on the backing chain.
(5) *CbA Backing Vault (vault)*: a non-trusted third party enabling the issue protocol and executing the redeem protocol. The CbA Requester locks her coins $b$ with the vault while the CbA Redeemer upon destruction of $i(b)$ redeems $b$ from the vault.
(6) *Issuing Smart Contract* (`iSC`): implements the sub-protocols on the issuing chain.

The vault only becomes active during the redeem sub-protocol as any actions in the issuing sub-protocol are executed by the CbA Requester and the `iSC`. To ensure correct behaviour, the vault has to provide a deposit on the issuing chain which allows the issuing and redeeming of CbAs. The `iSC` serves as an escrow for the deposit of the vault. The deposit is a promise by the vault to release the backing tokens $b$ when a valid redeem request is made by destroying $i(b)$. Under the assumption of an economically rational vault, the vault releases $b$ as otherwise its deposit is refunded to the CbA Redeemer. Further, the `iSC` verifies correct execution of the issue and redeem process by verifying transactions occurring in the backing chain on the issuing chain via a chain relay.

## 7.2 Deposit dilemma for XCLAIM vaults

Vaults in XCLAIM provide a deposit in the currency of the issuing chain to insure against a risk on the backing chain. Exchange rate fluctuations are external events that affect the security assumption of a vault fulfilling a redeem request. Sudden drops in the exchange rate results in insecure protocol states where the provided deposit is less than the value of the coin $b$ on the backing chain. Hence, rational vaults have an incentive to refuse redeem requests even under the threat of having their deposit taken away. Further, vaults have a private valuation for the outcome of the redeem protocol. If a vault values the backing coin $b$ higher than the required deposit, the vault has an incentive to not fulfil the redeem request. XCLAIM mitigates exchange rate fluctuations and, implicitly, the potential detrimental effects of private information through staged deposits.

(1) In *secure operation*, a CbA Requester can issue new CbAs as $D_A$ is greater than an the ideal deposit $\lceil D_A \rceil$.
(2) In the *buffered deposit* stage, $D_A < \lceil D_A \rceil$ but $D_A$ is still above a lower bound deposit $\lfloor D_A \rfloor$. In this stage, a CbA Requester cannot issue new CbAs based on this vaults deposit. The vault can provide additional deposit to increase its deposit rate back to the ideal rate.
(3) In the *liquidation* stage, a vault's deposit is automatically liquidated if it falls below $\lfloor D_A \rfloor$. The deposit rate is *dynamically* adjusted using an oracle $O$ importing the exchange rate.

## 7.3 Application of Balance

We integrate Balance with XCLAIM by defining the agreements of vaults. These include their specification, payments, and deposits as well as the score. Further, we define the length of the time period $t$ in which an agent can perform actions. Last, we set the initial deposit factor $f_1$ and lower bound $f_\omega$. Our definitions are described in Figure 5. We leave the quantification of scores for actions and determining lower and upper bounds of layers as future work. We argue that Balance does not affect the security arguments of XCLAIM in Appendix C.3.

*Determining a minimum time.* XCLAIM assumes a minimum number of blocks to consider transactions confirmed depending on the security parameter $k$ [17]. As outlined in Section 4.4, we determine the period by considering the agreement with the most transactions. Agreements $\mathcal{A}_{1,3,4}$ depend on one issuing chain transactions while $\mathcal{A}_2$ depends on one backing chain and two issuing chain transactions. We use a buffer of 2 such that the time $t = 2(k_b\Delta_b + 2k_i\Delta_i)$. For the case where Bitcoin is the backing and Ethereum the issuing chain, we assume $k_{\text{ETH}} = 12$, $\Delta_{\text{ETH}} = 15$ s, $k_{\text{BTC}} = 6$, $\Delta_{\text{BTC}} = 10$ min. This results in $t \approx 132$ min or $t \approx 528$ blocks on Ethereum.

*Determining scores.* Within XCLAIM, the issue commitment $\mathcal{A}_1$ is equivalent to the commit stage. We choose $s = 0$ for $\mathcal{A}_1$ to prevent agents from splitting up their deposit into smaller parts. Further, we exclude a reward for $\mathcal{A}_3$ resolving $D_A < \lfloor D_A \rfloor$ as ideally $D_A$ should always be greater than $\lceil D_A \rceil$. We define a positive score for the desired action in $\mathcal{A}_2$. Further, we argue that having a positive score in $\mathcal{A}_4$ provides an additional incentive for $A$ to re-balance its deposit in case of exchange rate fluctuations. In turn, this improves the chance of CbA Requesters being able to participate in the issue sub-protocol.

*Determining a number of layers.* We determine the number of layers $\omega$ by considering which time period a performing agent $A$ is accounting for in its decision to decide between $\Sigma_d$ and $\Sigma_u$. We assume that $A$ considers a 24 hour time period. Next, we calculate $\omega$ by dividing 24 hours by the time representation of $t$. Based on $t = 528$ blocks, we set $\omega = 12$.

*Determining factors.* XCLAIM suggests deposit stages of 2.0 for $\lceil D_A \rceil$ and 1.05 for $\lfloor D_A \rfloor$. We leave $\lfloor D_A \rfloor$ as suggested. We focus on reducing $\lceil D_A \rceil$ for a concrete implementation of XCLAIM for Bitcoin and Ethereum. To verify the suggested deposit $\lceil D_A \rceil$, we perform an analysis of the daily exchange rate fluctuations and private valuations by parsing order books of exchanges (cf. Appendix C) for Bitcoin and Ethereum. Based on our analysis, and assuming $r = 0.05$ and $\delta = 0.9$ as well as a linear relation between layer factors, we set $f_1 = 2.06$ and $f_\omega = 1.85$. Thereby, we achieve a reduction of 10% of the deposit.

## 7.4 Implementation

We implemented Balance for Ethereum in Solidity v0.5.0[11]. The implementation consists of around 170 lines of code. The core functions of Balance are *update* and *curate* defined by (9) and (10). The cost experiments are conducted under the assumption that all agents in the current registry change their layer (i.e. maximum updates in the smart contract)[12]. We conducted experiments with up to ten layers. The `update` function amounts to 55,287 gas costing

[11] Implementation available at https://github.com/nud3l/layered-tcr.
[12] All USD conversions are made with an Ethereum exchange rate of USD 230 and a gas price of 6 GWEI.

**Vault agreements**

*Roles:* A mapping from XCLAIM to Balance roles.

(1) **Performing agent** $A$: Vault
(2) **Receiving agent** $B$: CbA Requester and CbA Redeemer
(3) **Registry** $\mathcal{R}$: Integrated in the `iSC`

*Events:* An update to the exchange rate at time $t$ changes the required ideal deposit $\lceil D_A \rceil$ and $\lfloor D_A \rfloor$. The current deposit $D_A$ might fall below $\lceil D_A \rceil$ or $\lfloor D_A \rfloor$.

*Private information:* $A$ might value a currency higher such that even if $D_A > \lfloor D_A \rfloor$, $A$ behaves economically rational when performing an undesired action.

$\mathcal{A}_1$: **Issue commitment**

- $\phi$ **True**, if $A$ provides a deposit $D_A \geq \lceil D_A \rceil$.
  **False**, if $D_A < \lceil D_A \rceil$.
- $p$ If $\phi$ of $\mathcal{A}_{1,4}$ is true, $A$ receives $p_A$ paid by $B$ in its issue request.
- $\mathcal{D}$ $A$'s deposit $D_A$ is refunded to $A$ if $\phi$ of $\mathcal{A}_1$ is false.

$\mathcal{A}_2$: **Redeem request** (*Precondition:* $\phi$ of $\mathcal{A}_1$ is true.)

- $\phi$ **True**, if $A$ fulfils a *valid* redeem request by $B$ within a time $w$.
  **False**, if $A$ fails to provide a proof of executing $B$'s request within $w$.
- $p$ If $\phi$ of $\mathcal{A}_2$ is true, $A$ receives payment $p_A$[a].
- $\mathcal{D}$ $A$'s deposit $D_A$ is transferred to $B$ if $\phi$ of $\mathcal{A}_2$ is false.

$\mathcal{A}_3$: **Liquidation** (*Precondition:* $\phi$ of $\mathcal{A}_1$ is true.)
Trigger: Exchange rate update provided by oracle $O$.

- $\phi$ **True**, if $D_A$ is above $\lfloor D_A \rfloor$.
  **False**, if $D_A$ is below $\lfloor D_A \rfloor$.
- $p$ There are no payments for performing this action.
- $\mathcal{D}$ $A$'s deposit $D_A$ is used to perform a redeem request as stated in $\mathcal{A}_2$.

$\mathcal{A}_4$: **Buffered deposit** (*Precondition:* $\phi$ of $\mathcal{A}_1$ is true.)
Trigger: Exchange rate update provided by oracle $O$.

- $\phi$ **True**, if $D_A$ is above $\lceil D_A \rceil$.
  **False**, if $D_A$ is below $\lceil D_A \rceil$.
- $p$ There are no payments for performing this action. $A$ cannot participate in the issue protocol while $\phi$ is false.
- $\mathcal{D}$ There is no impact on the deposit.

[a] XCLAIM does not define how payments are made during redeem (cf. Section VII-F [53]). We assume that a fee is already paid during issue.

**Parameters of Balance**

*Assumptions*:

- $r$ 0.05. $A$ earns 5% of its deposit by participating in other protocols.
- $\delta$ 0.9. $A$ discounts future income by 10%.

*Time constraints:*

- $t$ 528 blocks (on Ethereum)

*Actions:* Actions depend on the agreements and have an associated score.

- $\mathcal{A}_1$ **Desired** $\sigma_d$: $A$ provides $D_A \geq \lceil D_A \rceil$.
  **Undesired** $\sigma_u$: $A$ provides $D_A < \lceil D_A \rceil$.
  **Score** $s$: 0. $A$ should not receive a reduction of deposit for adding deposit in small quantities to reduce its overall deposit. By setting $s = 0$ for $\mathcal{A}_1$ we prevent such behaviour.
- $\mathcal{A}_2$ **Desired** $\sigma_d$: $A$ fulfils $B$'s redeem request within $w$.
  **Undesired** $\sigma_u$: $A$ fails to execute $B$'s redeem request within $w$.
  **Score** $s$: $> 0$. $A$ receives an additional incentive to execute redeem requests and can in return reduce its deposit.
- $\mathcal{A}_3$ **Desired** $\sigma_d$: If $D_A < \lfloor D_A \rfloor$, $A$ adds additional deposit $D'_A$ such that $D_A + D'_A \geq \lfloor D_A \rfloor$.
  **Undesired** $\sigma_u$: $A$ does not add deposit new deposit or an insufficient amount such that $D_A + D'_A < \lfloor D_A \rfloor$.
  **Score** $s$: 0. $A$ should not receive a reward for being temporary below the liquidation bound.
- $\mathcal{A}_4$ **Desired** $\sigma_d$: If $D_A < \lceil D_A \rceil$, $A$ adds additional deposit $D'_A$ such that $D_A + D'_A \geq \lceil D_A \rceil$.
  **Undesired** $\sigma_u$: $A$ does not add deposit new deposit or an insufficient amount such that $D_A + D'_A < \lceil D_A \rceil$.
  **Score** $s$: $> 0$ should receive an incentive for staying above the ideal deposit, however, this might result in purposely falling under the ideal deposit.

*Layers:* The layer factors express the boundary for $\lceil D_A \rceil$. We leave $\lfloor D_A \rfloor$ as suggested by XCLAIM at 1.05.

- $\omega$ 12. Based on considering a time window of 24 hours and the minimum time of 528 blocks.
- $f_1$ 2.06. Based on our analysis of exchange rate and order books.
- $f_\omega$ 1.85. Based on applying $f_1$ to (20) and considering the exchange rate and order book analysis.

**Figure 5: An overview of the agreements of a vault in XCLAIM and parameters for an integration of Balance into XCLAIM.**

around USD 0.07. The execution of the `curate` function costs 54,948 gas which is equivalent to around USD 0.07. We reduce the linear complexity of the `curate` function (10) to constant by executing the assignment of agents for the next round in the `update` function. The `curate` function updates the round counter and activates the mapping for the next round.

## 8 RELATED WORK

To the best of our knowledge, Balance is the first reputation-based system for the dynamic adjustment of cryptocurrency deposits. There are three discernible strands of related literature.

The first strand relates to Token Curated Registries (TCRs) [37], inspiring the layered aspects of Balance [34]. A TCR formally represents a set $\mathcal{R}$ in which elements $n$ can be included in a set through a token-based voting mechanism. A variety of different TCR types have been proposed [10, 16, 34, 37, 45]. Notably, ranked TCRs $\mathcal{R}_O$ enable agents to vote on the rank (i.e. position) of an element in a set of elements such that $n_i \prec n_{i+1}$. Further, layered TCRs are a set $\mathcal{R}_\mathcal{L}$, consisting of distinct subsets where $\cup_{i=1}^{n} R_i \subseteq \mathcal{R}_\mathcal{L} \wedge \cap_{i=1}^{n} R_i = \emptyset$. However, while Balance takes inspiration from a ranked and layered TCR construction, TCRs require voting by individuals with tokens, adding significant and potentially unwarranted complexity [2].

The second strand concerns *reputation* aspects of Balance. Yu et al. uses a notion of reputation to define a miner's power in terms of its work performed over the lifetime of a blockchain, as opposed to instantaneous mining power, in order to mitigate the vulnerability of blockchains to 51% attacks, where an adversary momentarily possesses more than 50% of the total mining power [52]. Another system for reputation management in decentralised systems, but this time for users of the system as opposed to miners, is [32]. The mechanism uses cryptocurrencies to measure trust: using deposits between different agents, the authors construct a web-of-trust like architecture. Balance is different, creating only direct trust relationships between agents and actions are directly evaluated through the agreements in a smart contract. Taxonomies of reputation based systems [22, 23, 43] indicate comparators for Balance and other reputation based systems. Balance is a *quantitative* as opposed to qualitative trust system, expressing a reputation in terms of a deposit factor. Reputation itself is accrued by agents through direct experience, i.e. direct interaction with a smart contract. In particular, there is no transitive reputation between peers: the trust that *others* place in an agent does not confer trust onto the agent [13]. Some

[13] One caveat is that in one sense trust is transitive across agreements, since an agent may participate in multiple agreements within the same smart contract, retaining the same deposit factor.

systems such as Pisa [35] do allow for transitive trust, allowing a reputation to be indirectly established.

A third strand of literature is pursued by [7]. Seeking to guard against losses of users' funds by centralised exchanges [39], the work focuses on providing a privacy-preserving system for proving the solvency of an exchange, i.e. that an exchange controls sufficient reserves to settle the account of each user.

## 9 CONCLUSION

Balance is an application-agnostic system, intended as an extension to existing cryptoeconomic protocols, that allows for the reduction of cryptocurrency deposits without compromising security. By explicitly modelling agents' utilities, we show that it features an incentive-compatible mechanism that rewards agents for the performance of desired actions by reducing their required deposits, and therefore the opportunity costs of those deposits. Moreover, we show that the addition of Balance increases social welfare. We also implement Balance, integrating it with XCLAIM.

The primary motivating force for agents in our construction is the expected reduction in opportunity costs for the agent resulting from forgone returns on deposits. If we modify the assumption of perfect competition in Section 5, such that agents receive a positive payment from performing a desired action, payments $p$ could also constitute a motivating force. For protocols where a reduction in the deposit is not practical or not desired by the protocol designers, the factor in each layer could be used to calculate the payment. In this case, the factor would increase with every layer so that agents in the lowest layer receive a payment of $pf_1$, where e.g. $f_1 = 0.6$.

To the best of our knowledge, this paper is the first to use a curated registry to enable dynamic deposit requirements. One question that arises is that given $v$ is private information, at what level should deposits and factors be set such that the proportion of agents who find it incentive compatible to perform desired actions is optimal for the protocol (or society) as a whole? In addition, we plan to explore different parameter configurations such as overlapping boundaries and restrict the number of agents per layers, as well as extend the model to cover probabilistic formulations of the specification.

## ACKNOWLEDGMENTS

The authors would like to thank Zeynep Gurguc for discussions on game theory and mechanism design as well as the cryptoeconomics team at Outlier Ventures and Aron van Ammers for their continued support. Further, the authors thank the reviewers for their valuable comments to improve the paper. This research is funded by Outlier Ventures and EPSRC Standard Research Studentship (DTP) under Grant EP/R513052/1.

## REFERENCES

[1] Amitanand S Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. 2005. BAR fault tolerance for cooperative services. *ACM SIGOPS Operating Systems Review* 39, 5 (2005), 45. https://doi.org/10.1145/1095809.1095816

[2] Aditya Asgaonkar and Bhaskar Krishnamachari. 2018. Token Curated Registries - A Game Theoretic Approach. (2018), 16 pages. arXiv:1809.01756 http://arxiv.org/abs/1809.01756

[3] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. 2018. Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security - CCS '18*. 913–930. https://doi.org/10.1145/3243734.3243848

[4] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. 2017. Bitcoin as a Transaction Ledger: A Composable Treatment. In *Advances in Cryptology - CRYPTO 2017*, Vol. 10401 LNCS. 324–356. http://link.springer.com/10.1007/978-3-319-63688-7{_}11

[5] Lorenz Breidenbach, Phil Daian, Floriantra Er, and Ari Juels. 2018. Enter the Hydra: Towards Principled Bug Bounties and Exploit-Resistant Smart Contracts. In *27th USENIX Security Symposium (USENIX Security 18)*. 1335–1352. https://thehydra.io/paper.pdf

[6] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*. 173–186. https://doi.org/10.1.1.17.7523 arXiv:arXiv:1203.6049v1

[7] Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. 2015. Provisions: Privacy-preserving proofs of solvency for Bitcoin exchanges. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*. ACM Press, New York, New York, USA, 720–731. https://doi.org/10.1145/2810103.2813674

[8] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2019. Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges. (2019). arXiv:1904.05234 http://arxiv.org/abs/1904.05234

[9] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *Advances in Cryptology - EUROCRYPT 2018*, Jesper Buus Nielsen and Vincent Rijmen (Eds.), Vol. 46. Springer International Publishing, Cham, 66–98. https://doi.org/10.1007/978-3-319-78375-8_3

[10] Simon de la Rouviere. 2017. Introducing Curation Markets: Trade Popularity of Memes & Information (with code)! https://medium.com/@simondlr/introducing-curation-markets-trade-popularity-of-memes-information-with-code-70bf6fed9881

[11] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*. IEEE, 1–10. https://tik-old.ee.ethz.ch/file//49318d3f56c1d525aabf7fda78b23fc0/P2P2013_041.pdf

[12] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. 2018. FairSwap: How To Fairly Exchange Digital Goods. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security - CCS '18*. ACM Press, New York, New York, USA, 967–984. https://doi.org/10.1145/3243734.3243857

[13] Ittay Eyal and Emin Gün Sirer. 2014. Majority Is Not Enough: Bitcoin Mining Is Vulnerable. In *Financial Cryptography and Data Security 2014*, Vol. 8437. Berlin, Heidelberg, 436–454. https://doi.org/10.1007/978-3-662-45472-5_28

[14] St. Louis Fed. 2019. 10-Year Treasury Constant Maturity Rate, DGS10. Available at: https://fred.stlouisfed.org/series/DGS10.

[15] Shane Frederick, George Loewenstein, and Ted O'Donoghue. 2002. Time Discounting and Time Preference: A Critical Review. *Journal of Economic Literature* 40, 2 (2002), 351–401. http://www.jstor.org/stable/2698382

[16] Sebastian Gajek. 2018. Graded Token-Curated Decisions with Up-/Downvoting. https://medium.com/coinmonks/graded-token-curated-decisions-with-up-downvoting-designing-cryptoeconomic-ranking-and-2ce7c000bb51

[17] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology-EUROCRYPT 2015*. Springer, 281–310. http://courses.cs.washington.edu/courses/cse454/15wi/papers/bitcoin-765.pdf

[18] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 3–16.

[19] Arthur Gervais, Hubert Ritzdorf, Ghassan O Karame, and Srdjan Capkun. 2015. Tampering with the delivery of blocks and transactions in bitcoin. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 692–705.

[20] Dominik Harz and Magnus Boman. 2019. The Scalability of Trustless Trust. In *2nd Workshop on Trusted Smart Contracts (Financial Cryptography and Data Security)*. 279–293. https://doi.org/10.1007/978-3-662-58820-8_19 arXiv:1801.09535

[21] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In *24th USENIX Security Symposium (USENIX Security 15)*. 129–144. https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-heilman.pdf

[22] Ferry Hendrikx, Kris Bubendorfer, and Ryan Chard. 2015. Reputation systems: A survey and taxonomy. *J. Parallel and Distrib. Comput.* 75 (jan 2015), 184–197. https://doi.org/10.1016/j.jpdc.2014.08.004

[23] Audun Jøsang, Roslan Ismail, and Colin Boyd. 2007. A survey of trust and reputation systems for online service provision. *Decision Support Systems* 43, 2 (mar 2007), 618–644. https://doi.org/10.1016/j.dss.2005.05.019

[24] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. 2018. Arbitrum: Scalable, private smart contracts. In *Proceedings of the 27th USENIX Conference on Security Symposium*. USENIX Association,

1353–1370.
[25] Sepandar D Kamvar, Mario T Schlosser, and Hector Garcia-Molina. 2003. The Eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the twelfth international conference on World Wide Web - WWW '03*. ACM Press, New York, New York, USA, 640. https://doi.org/10.1145/775152.775242
[26] Rami Khalil and Arthur Gervais. 2017. Revive: Rebalancing Off-Blockchain Payment Networks. Cryptology ePrint Archive, Report 2017/823. http://eprint.iacr.org/2017/823.pdf Accessed:2017-09-26.
[27] Rami Khalil, Arthur Gervais, and Guillaume Felley. 2018. NOCUST–A Securely Scalable Commit-Chain. *IACR Cryptology ePrint Archive* 2018 (2018), 642.
[28] Rami Khalil, Arthur Gervais, and Guillaume Felley. 2019. TEX-A Securely Scalable Trustless Exchange. *IACR Cryptology ePrint Archive* 2019 (2019), 265.
[29] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Symposium on Security & Privacy*. IEEE. http://eprint.iacr.org/2015/675.pdf
[30] Ranjit Kumaresan and Iddo Bentov. 2014. How to use bitcoin to incentivize correct computations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 30–41. http://www.cs.technion.ac.il/~iddo/incentivesBitcoin.pdf
[31] Yoad Lewenberg, Yoram Bachrach, Yonatan Sompolinsky, Aviv Zohar, and Jeffrey S Rosenschein. 2015. Bitcoin Mining Pools: A Cooperative Game Theoretic Analysis Categories and Subject Descriptors. *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)* (2015), 919–927.
[32] Orfeas Stefanos Thyfronitis Litos and Dionysis Zindros. 2017. Trust Is Risk: A Decentralized Financial Trust Platform. (2017), 156 pages.
[33] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. 2015. Demystifying incentives in the consensus computer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 706–719. http://www.comp.nus.edu.sg/~prateeks/papers/VeriEther.pdf
[34] Trent McConaghy. 2018. The Layered TCR. https://blog.oceanprotocol.com/the-layered-tcr-56cc5b4cdc45
[35] P McCorry, S Bakshi, I Bentov, Andrew Miller, and ... 2018. Pisa: Arbitration Outsourcing for State Channels. (2018). https://www.cs.cornell.edu/{~}iddo/pisa.pdf
[36] Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. 2018. Smart Contracts for Bribing Miners. In *Financial Cryptography and Data Security. FC 2018.*, Vol. 10958. Springer Berlin Heidelberg, 3–18. https://doi.org/10.1007/978-3-662-58820-8_1
[37] Mike Goldin. 2017. Token-Curated Registries 1.0. (2017), 10 pages. https://medium.com/@ilovebagels/token-curated-registries-1-0-61a232f8dac7
[38] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 31–42.
[39] Moore, Tyler and Christin, Nicolas. 2013. Beware the middleman: Empirical analysis of Bitcoin-exchange risk. In *International Conference on Financial Cryptography and Data Security*. Springer, 25–33.
[40] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf. https://bitcoin.org/bitcoin.pdf Accessed: 2015-07-01.
[41] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. 2007. *Algorithmic Game Theory*. Vol. 1. Cambridge University Press, Cambridge. 1–754 pages. https://doi.org/10.1017/CBO9780511800481 arXiv:0907.4385
[42] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the Blockchain Protocol in Asynchronous Networks. In *Advances in Cryptology - EUROCRYPT 2017*, Vol. 10211. 643–673. https://doi.org/10.1007/978-3-319-56614-6_22
[43] Isaac Pinyol and Jordi Sabater-Mir. 2013. Computational trust and reputation models for open multi-agent systems: A review. *Artificial Intelligence Review* 40, 1 (2013), 1–25. https://doi.org/10.1007/s10462-011-9277-z
[44] Joseph Poon and Thaddeus Dryja. 2016. The bitcoin lightning network. https://lightning.network/lightning-network-paper.pdf. https://lightning.network/lightning-network-paper.pdf Accessed: 2016-07-07.
[45] Achill Rudolph. 2018. Ranking Token Curated Registries. https://medium.com/coinmonks/ranking-token-curated-registries-e9a92dc85b31
[46] Tuomas W. Sandholm and Victor R. Lesser. 2001. Leveled Commitment Contracts and Strategic Breach. *Games and Economic Behavior* 35, 1-2 (2001), 212–270. https://doi.org/10.1006/game.2000.0831
[47] Tendermint. 2016. Introduction to Tendermint - Tendermint. https://tendermint.com/intro
[48] Jason Teutsch and Christian Reitwießner. 2017. A scalable verification solution for blockchains. (2017), 44 pages.
[49] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. 2018. Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 67–82.
[50] Gavin Wood. 2014. Ethereum: a secure decentralised generalised transaction ledger. (2014), 32 pages.
[51] Michael Wooldridge. 2009. *An Introduction to MultiAgent Systems* (2nd ed.). Wiley Publishing.
[52] Jiangshan Yu, David Kozhaya, Jeremie Decouchant, and Paulo Verissimo. 2019. RepuCoin: Your Reputation is Your Power. *IEEE Trans. Comput.* (2019), 1–1. https://doi.org/10.1109/TC.2019.2900648
[53] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William J. Knottenbelt. 2019. XCLAIM: Trustless, Interoperable, Cryptocurrency-Backed Assets. In *Proceedings of the IEEE Symposium on Security & Privacy, May 2019*. 1254–1271. https://doi.org/10.1109/SP.2019.00085
[54] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. 2016. Town Crier: An Authenticated Data Feed for Smart Contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 270–282. https://eprint.iacr.org/2016/168.pdf
[55] Guy Zyskind, Oz Nathan, and Alex Pentland. 2015. Enigma: Decentralized computation platform with guaranteed privacy. arXiv preprint arXiv:1506.03471. https://arxiv.org/pdf/1506.03471.pdf Accessed: 2017-08-22.

# A CRYPTOECONOMIC PROTOCOLS

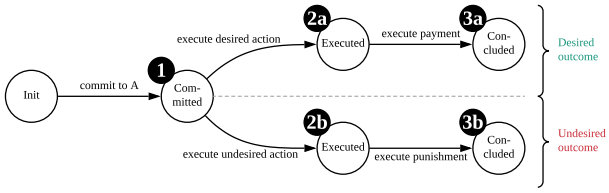



**Figure 6: A cryptoeconomic protocol consists of three states. (1) an agent commits to the fulfilment of a specification $\phi$. (2) the agent executes an action that either evaluates to true or false w.r.t. the specification. (3) the protocol is concluded by rewarding or punishing the agent for its action.**

# B INCENTIVE PROOFS

The rational agent type $\mathcal{T}_r$ needs to decide which action to perform at the highest layer. From Figure 3, we see that at the highest layer the agent has either the choice to perform the desired action which results in a utility of $p - c_A - \mathrm{E}[rD_3]$ or an undesired action which is equivalent to a utility of $v - c_A - \mathrm{E}[rD_3] - D_3$. In this simplified figure, there are three layers. However, we can generalise this by assuming $\omega$ layers. By Definition 4, the rational type cannot come to a decision just considering these two utility functions. Reaching a decision requires $\mathcal{T}_r$ to consider several time-steps. By integrating Balance, we transform the decision process into a *sequential game*.

## B.1 Decision boundary for type $\mathcal{T}_r$

The total utility received by an agent performing desired actions in the highest layer, for a sequence that lasts $t = \omega$ rounds, is expressed by:

$$u_A = \sum_{t=0}^{\omega-1} \left(\frac{\delta}{1+r}\right)^t \left(p - c_A - \mathrm{E}[rD_\omega]\right) \tag{24}$$

Summing over cycles to infinity yields the following expression ($\tau$ swapped for $t$ to express 'within' cycle time):

$$u_A = \sum_{\gamma=0}^{\infty} (\frac{\delta}{1+r})^{\gamma\omega} \left( \sum_{\tau=0}^{\omega-1} (\frac{\delta}{1+r})^{\tau} (p - c_A - \mathrm{E}[rD_\omega]) \right) \tag{25}$$

The total utility corresponding to an agent who commits an undesired action in the highest layer $\omega$, followed by $\omega - 1$ desired

actions to return to that layer, can be expressed as a sequence with $t = \omega$ rounds as follows. We express the deposit $D$ of layer $L_m \in \mathcal{L}$ at time $t$ with $D_t$.

$$u_A = v - c_A - \mathrm{E}[rD_\omega] - D_\omega + \sum_{t=1}^{\omega-1} \left(\frac{\delta}{1+r}\right)^t \left(p - c_A - \mathrm{E}[rD_t]\right) \tag{26}$$

Summing over cycles to infinity yields the following expression ($\tau$ swapped for $t$ to express 'within' cycle time):

$$\begin{aligned} u_A = \sum_{\gamma=0}^{\infty} (\frac{\delta}{1+r})^{\gamma\omega} \Big( & v - c_A - D_\omega - \mathrm{E}[rD_\omega] \\ & + \sum_{\tau=1}^{\omega-1} (\frac{\delta}{1+r})^{\tau} (p - c_A - \mathrm{E}[rD_\tau]) \Big) \end{aligned} \tag{27}$$

The decision boundary for agents of type $\mathcal{T}_r$ is found by equating equations (25) and (27). This is as follows.

$$\begin{aligned} & \sum_{\gamma=0}^{\infty} (\frac{\delta}{1+r})^{\gamma\omega} \Big( v - c_A - D_\omega - \mathrm{E}[rD_\omega] \\ & \quad + \sum_{\tau=1}^{\omega-1} (\frac{\delta}{1+r})^{\tau} (p - c_A - \mathrm{E}[rD_\tau]) \Big) \\ & = \sum_{\gamma=0}^{\infty} (\frac{\delta}{1+r})^{\gamma\omega} \Big( \sum_{\tau=0}^{\omega-1} (\frac{\delta}{1+r})^{\tau} (p - c_A - \mathrm{E}[rD_\omega]) \Big) \end{aligned} \tag{28}$$

The following lemma simplifies this boundary condition.

**Lemma 2.** Comparing payoffs of desired as opposed to undesired actions into the infinite horizon, with $\gamma \to \infty$, is the same as comparing the payoffs of these actions over a single cycle.

Proof. Inspection of (28) shows that the two sides of the equation are equal when the terms inside the summation operator (summing from $\gamma = 0$ to $\infty$) are equal. □

Therefore the equation becomes

$$\begin{aligned} & v - c_A - D_\omega - \mathrm{E}[rD_\omega] + \sum_{t=1}^{\omega-1} (\frac{\delta}{1+r})^t (p - c_A - \mathrm{E}[rD_t]) \\ & = \sum_{t=0}^{\omega-1} (\frac{\delta}{1+r})^t (p - c_A - \mathrm{E}[rD_\omega]) \end{aligned} \tag{29}$$

Which simplifies to:

$$v = p + D_\omega + \sum_{t=1}^{\omega-1} (\frac{\delta}{1+r})^t (\mathrm{E}[rD_t] - \mathrm{E}[rD_\omega]) \tag{30}$$

If we equate (24) and (26) we get the decision boundary for an agent to perform a desired or undesired action. We express this by the factor $f$. By (7), $f_m$ determines the deposit of a layer $m$ in relation to the base deposit $D_{\text{base}}$.

**Lemma 3.** Introducing Balance creates a sequential game in which factors $f$ determine the deposit at each layer. The factor of the highest layer $f_\omega$ in a sequential game with at least two layers can be set lower than $f_1$ and still achieves a higher utility for a desired action compared to an undesired action.

Proof. First, we equate (24) and (26) characterizing the decision boundary. Next, we set $D_\omega = f_\omega D_{\text{base}}$ and $D_t = f_t D_{\text{base}}$ (where $t$ and $m$ are are substitutes for a single cycle). This leaves us with the following equality.

$$\begin{aligned} & \sum_{t=0}^{\omega-1} \left(\frac{\delta}{1+r}\right)^t \left(p - c_A - \mathrm{E}[rf_\omega D_{\text{base}}]\right) \\ & = v - c_A - \mathrm{E}[rf_\omega D_{\text{base}}] - (f_\omega D_{\text{base}}) \\ & \quad + \sum_{t=1}^{\omega-1} \left(\frac{\delta}{1+r}\right)^t \left(p - c_A - \mathrm{E}[rf_t D_{\text{base}}]\right) \end{aligned} \tag{31}$$

We assume that the agent is able to progress to the next layer after every time step $t$. Also, we apply our initial assumptions that $p - c_A$ for performing a desired action is 0, and that $c_A$ for the undesired action is 0. We then simplify (31).

$$\begin{aligned} & \sum_{t=1}^{\omega-1} \left(\frac{\delta}{1+r}\right)^t \left( - \mathrm{E}[rf_\omega D_{\text{base}}]\right) \\ & = v - f_\omega D_{\text{base}} + \sum_{t=1}^{\omega-1} \left(\frac{\delta}{1+r}\right)^t \left( - \mathrm{E}[rf_t D_{\text{base}}]\right) \end{aligned} \tag{32}$$

$$v = f_\omega D_{\text{base}} + \sum_{t=1}^{\omega-1} \left(\frac{\delta}{1+r}\right)^t \left(\mathrm{E}[rD_{\text{base}}(f_t - f_\omega)]\right) \tag{33}$$

When we equate $v$ and $D_{\text{base}}$, we imply the decision boundary without Balance. We can then express $f_\omega$ in relative terms by setting $v = D_{\text{base}} = 1$, and rearranging for $f_\omega$.

$$f_\omega = \frac{1 - \sum_{t=1}^{\omega-1} \left(\frac{\delta}{1+r}\right)^t \mathrm{E}[rf_t]}{1 - \sum_{t=1}^{\omega-1} \left(\frac{\delta}{1+r}\right)^t \mathrm{E}[r]} \tag{34}$$

From (34), we observe that if $f_t$ is smaller than 1, then $f_\omega$ would become greater than 1. This would allow the agent to perform undesired actions in the first layer at a positive utility since $v$ would become larger than $D_1 f_1 - p$. Therefore, the factor at the lowest layer must be greater than 1. If we set any factor $f_t$ equal to 1, $f_\omega$ becomes also 1. Consequently, allowing a factor $f_t$ to be greater than 1 allows us to set a deposit factor $f_\omega$ smaller than $f_1$. □

**Lemma 4** (Minimum deposit at the lowest layer)**.** The deposit factor at the lowest layer $f_1$ must be greater than 1 to allow the factor at the highest layer $f_\omega$ to be less than 1.

Proof. The proof follows from the proof of Theorem 3. □

## B.2 Linear factor adjustment

We assume a linear relationship between the smallest factor $f_\omega$ and the other factors $f_t$. Next, we assume that at $D_1$ the factor $f_1$ is $> 1$ and that at $D_\omega$ the factor $f_\omega \geq 1$ is the lowest factor. From this, we can calculate linear relation between $f_1$ and $f_\omega$ as follows.

$$f_t = f_1 - \left(\frac{f_1 - f_\omega}{\omega - 1}\right)(t - 1) \tag{35}$$

We can then replace $f_t$ in (33) by (35) to express the term purely in terms of $f_\omega$. Further, we assume that the valuation $v$ is lower than the deposit $D_1$, i.e. the deposit is higher than the desire for performing an undesired action.

This leaves us with the following equation:

$$f_\omega = \frac{v\omega - v - \sum_{t=1}^{\omega-1} \left(\frac{\delta}{1+r}\right)^t \mathrm{E}[r f_1(\omega - t)]}{\omega - 1 - \sum_{t=1}^{\omega-1} \left(\frac{\delta}{1+r}\right)^t \mathrm{E}[r(\omega - t)]} \tag{36}$$

# C XCLAIM

## C.1 Currency fluctuations and valuations

*ETH-BTC fluctuation.* In order to determine the buffer for the deposit to account for exchange rate fluctuations we collect the daily high and low prices of BTC to USD and ETH to USD from a period of one year (May 3, 2018 to May 3, 2019). The data is collected from the Poloniex exchange API[14]. We are interested in the *drop* of the exchange rate, i.e. from a the high price to the low price. Therefore we calculate the drop as $\frac{p_{\text{low}} - p_{\text{high}}}{p_{\text{high}}}$. Using the exchange data, we find that the highest drop within the year is 17.06% and the average per day drop is 3.90% with a standard deviation of 2.73%. The maximum daily exchange rate drop is visualised in Figure 7.

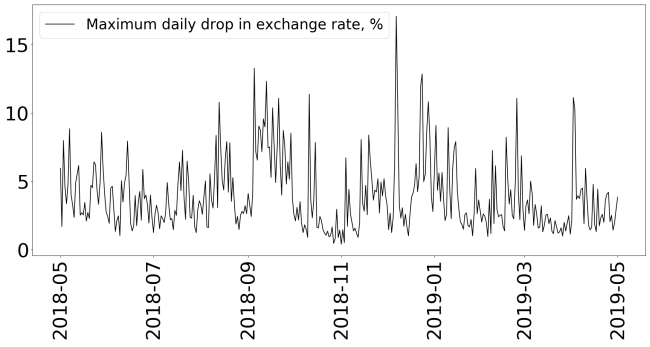



**Figure 7: Using daily data from the Poloniex exchange, this figure plots the percentage decrease (in absolute terms) of the low price relative to the high price for a given day.**

*ETH-BTC valuations.* Agents can express a different preference for cryptocurrencies. To quantitatively approximate this difference we scanned the order books of the Binance exchange for bids and asks for a period of one week from May 3 to May 9, 2019[15]. We calculated the valuation difference by comparing the bid/ask price to the exchange rate at the time the order is placed. We note that most agents place orders close to the actual exchange rate as the mean difference between price and exchange rate is 0.64% with a standard deviation of 2.73%. However, there are significant outliers with up to 76.34% in this data. The valuation difference serves as an orientation to calculate the security buffer for the cross-currency deposit.

We note several shortcomings of our approach. Orders placed on an exchange may not reflect the honest valuation of the agent. Agents might place extreme orders to influence the exchange rate price instead of actually valuing the currency at that level, and more generally market manipulation (such as front-running and transaction re-ordering) is cause for concern [8] [16]. Further, we did not consider a *weighted* approach where the difference between price and exchange rate also considers the size of the order. Further, large difference can also be caused by human error when entering the order.

## C.2 Factor calculation

We can calculate the factors taking the highest deviations from the exchange rate fluctuation and the valuation differences. We define the highest exchange rate fluctuation as $\Delta_{ex} = 0.1706$ and the highest valuation difference as $\Delta_v = 0.7634$. We calculate a buffer $b_{\text{deposit}}$:

$$b_{\text{deposit}} = (1 + \Delta_{ex}) * (1 + \Delta_v) \tag{37}$$

We then determine the factor $f_1$ by using a base deposit of 1: $f_1 = 1 * b_{\text{deposit}} = 2.06424$. We assume $r = 0.05$[17], $\delta = 0.9$ [18], $\omega = 12$ and linear relationship between factors. Further, we apply the buffer $b_{\text{deposit}}$ to all subsequent factors $f$. We apply these assumptions and results to (20). This yields $f_\omega = 1.85$. Our results are visualised in Figure 8.

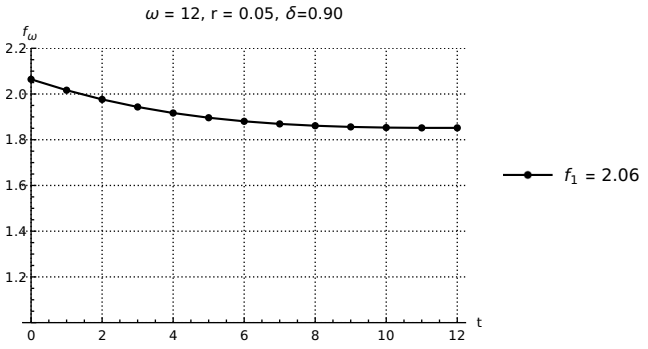



**Figure 8: Boundary of $f_\omega$ for the initial factor of $f_1 = \{2.06\}$ with an additional buffer for exchange rate fluctuations and valuation differences. The function is the decision boundary for an economically rational agent to decide between a desired and undesired action given a number of time steps $t$. Assuming $r = 0.05$, $\delta = 0.9$, $\omega = 12$.**

## C.3 Security arguments

We discuss possible effects on XCLAIM by integrating Balance.

[14] Taken from https://docs.poloniex.com/

[15] Taken from https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md

[16] However, if the order is matched, an agent will need to fulfil it, providing a constraint on behaviour

[17] A risk free rate of 5% approximates, for instance, the long term average rate on 10 year Treasuries of 6.14% [14]

[18] [15] supports a discount factor of 0.9

*Vaults A take multiple roles to reduce their deposit.* There is no guarantee that $A$ has an opportunity to perform a desired action within a given period $t$. $A$ would thus fall to a lower layer. To prevent this, $A$ could ensure a constant flow of redeem requests by creating an artificial demand as $A$ can act as CbA Requester, Vault, and CbA Redeemer, called strategy $S_{\text{boosting}}$ as described in Section 6.2. Assuming that expected interest on the deposit $\mathrm{E}[r] * D_{\text{layer}(A)}$ is higher than the cost $c_A(S_{\text{boosting}})$, this would be a dominant strategy. This lead to an issue: $A$ could ensure that its actions reduce its deposit and could perform the undesired action, i.e. not executing a redeem request of $B$ with a lower deposit and punishment. XCLAIM can prevent this by requiring a *maximum deposit* per vault $A$. Given an expected interest E$r$, the maximum allowed deposit is defined by $D_{max} = \frac{c_A(S_{\text{demand}})}{\mathrm{E}[r]}$. This also holds true under Sybil identities, since deposit reductions are not transitive between multiple identities.

*Conflicting minimum and maximum deposits.* XCLAIM suggests to prevent Sybil attacks by either (i) requiring a minimum deposit from vaults through the `iSC`, or (ii) a fee for issuing based on the total amount and not per issue request. The minimum deposit amount is in conflict with the Sybil resistance requirement for Balance as defined in Section 6.2. Hence, XCLAIM would need to adopt a fee model based on issue amounts rather than requiring a minimum deposit per vault when integrating Balance. As such, Sybil resistance in both XCLAIM and Balance is maintained.

*Length of time period t motivates agents to delay actions by a vault.* Assuming that an agent can fulfil more than one desired action within a time period $t$ and $A$ has already a high enough score to progress to a higher layer or remain in the highest. In that case, $A$ can play a strategy $S_{\text{delay}}$ to delay performing redeem requests ($\mathcal{A}_2$) and balancing its deposit ($\mathcal{A}_3$) until the next period (see also Section 6.3). In XCLAIM's case $S_{\text{delay}}$ is limited by enforcing time limits on agreements $\mathcal{A}_{2,3,4}$. Thus, vaults do not delay their actions.

*Artificial redeem requests.* Since the score for the redeem request $\mathcal{A}_2$ is positive, $A$ might try to obtain $i(b)$ to execute redeem requests to maintain it's ranking. This would require that the cost for redeem is lower than the opportunity cost of remaining at a higher layer. This argumentation is similar to Section 6.2 as a maximum deposit will prevent a vault from having a higher gain in opportunity cost compared to the cost of executing redeem requests.

*Deposit adjustments to improve score.* The score to resolve a state where $D_A < \lceil D_A \rceil$ is positive. As such, $A$ has an incentive to let its deposit fall below $\lceil D_A \rceil$ and resolve this state by adding more deposit, expressed by strategy $S_{\text{deposit-bouncing}}$. By having a positive score, Balance encourages $S_{\text{deposit-bouncing}}$. However, we can prevent the impact of this strategy by setting the score for such an action to be below a lower bound of a given layer $L_m$. For example, XCLAIM might allow $A$ to improve its score with $S_{\text{deposit-bouncing}}$, but sets the score such that $A$ is only able to reach a layer $L_m \prec L_\omega$. Thereby, it *limits* $S_{\text{deposit-bouncing}}$ but does not prevent it.

# D NOTATION AND SYMBOLS

**Table 1: Overview of notation and symbols.**

| Symbol | Definition |
|---|---|
| $\pi$ | Cryptoeconomic protocol |
| $P = \{1, ..., n\}$ | Set of $n$ agents |
| $\sigma \in \{\Sigma_d, \Sigma_u\}$ | Desired and undesired action |
| $u_i$ | Utility of agent $i \in P$ |
| $v_i$ | Private valuation agent $i \in P$ attaches to the realisation of a particular outcome |
| $c_i$ | Costs for agent $i \in P$ |
| $r$ | Interest rate |
| $E[..]$ | Expected value |
| $\mathcal{A} = \langle \phi, p, \mathcal{D} \rangle$ | Agreements with specification $\phi$, payment $p$, and set of deposits $\mathcal{D}$ |
| $\pi_{\text{Balance}}$ | Cryptoeconomic protocol with Balance |
| $\mathcal{A} = \langle \phi, p, s, \mathcal{D} \rangle$ | Agreements with additional score $s$ |
| $\mathcal{L} = \{L_1 \prec ... \prec L_\omega\}$ | An ordered list of $\omega$ layers |
| $L_m = \langle l, u, f \rangle$ | Layer with a lower bound $l$, upper bound $u$, and a deposit factor $f$ for layer $L_m \in \mathcal{L}$ |
| $D_m$ | Deposit at a specific layer $L_m \in \mathcal{L}$, e.g. $D_1$ for layer 1 |
| $f_m$ | Deposit factor for layer $L_m \in \mathcal{L}$ |
| $\delta$ | Discount factor |
| $t$ | Time step |