Bootstrapping a stable computation token

Jason Teutsch
Truebit
jt@truebit.io

Sami Mäkelä
Truebit
mrsmkl@gmail.com

Surya Bakshi
UIUC/Truebit
sbakshi3@illinois.edu

August 9, 2019

Abstract

We outline a token model for Truebit, a retrofitting, blockchain enhancement which enables secure, community-based computation. The model addresses the challenge of stable task pricing, as raised in the Truebit whitepaper, without appealing to external oracles, exchanges, or hierarchical nodes. The system's sustainable economics and fair market pricing derive from a mintable token format which leverages existing tokens for liquidity. Finally, we introduce a governance layer whose lifecycles culminates with permanent dissolution into utility tokens, thereby tending the network towards autonomous decentralization.

1 Initializing Truebit

Bitcoin, from its outset, leveraged an egalitarian distribution method enabling anyone with an Internet-connected computer to obtain its native tokens [19]. The system's automated mining process indiscriminately, albeit probabilistically, distributes digital wealth in the form of bitcoins to individuals who run a Bitcoin client on their local machines. While the practicalities of scaling, demand, and human nature introduced some complexities beyond those conceived in Bitcoin's original specifications [18], the system's remarkable minimization of *politics*, or complex formal dependencies on the actions or judgements of privileged nodes, permit the underlying protocol to function under simple, mathematical assumptions.

Tokens born from and regulated by smart contracts, in contrast to Bitcoin's Nakamoto consensus, present unique bootstrapping challenges with respect to both adoption and distribution. Indeed, Bitcoin's value proposition of "generate your own cash and then spend it" doesn't translate easily into systems where intended *consumers* must supply such cash. We witness an abundance of miners and stakers offering computation power and capital in the blockchain space, however the corresponding consumers for these services, when distinct from the service providers, remain far less ubiquitous. Some protocols, like Livepeer's MerkleMine [20], have dispensed tokens through computational work at the smart contract layer, yet sustainable distribution through apolitical function lingers elusively on blockchain's horizon. In light of the relatively low demand for decentralized services, we concentrate on an economic design which minimizes friction and politics for consumers without sacrificing security.

Consumers generally find convenience in predictable pricing as acquisition of an asset often does not coincide with its consumption. Consider a pilot who purchases sufficient aircraft fuel for a trip from Los Angeles to Tokyo and takes off. Halfway through his journey, the price of fuel increases by 20%. Consequently, a corresponding $1 - 1/(120\%)$ fraction of the pilot's remaining store vanishes from the gas tank, inconveniently diverting his course to Hawaii. While the physical world may prevent this particular scenario from occurring, the volatile world of cryptocurrency consumption provides no such guarantees. The pilot in this example requires a fixed amount of fuel, not a stable amount of fuel relative to the US dollar (USD) (compare with [2] and [6]). This example suggests the need for a kind of affordable, stable token independent of USD (Section 3). Both Truebit's stable token and fiat currency may correlate with the price of electricity (Section 3.2). We highlight that the Truebit protocol model assumes no distinguished authority nodes and, as such, achieves not only a trustless computation system but a decentralized one based on simple security assumptions and hierarchy-free pricing.

Any new network which requires consumers to pay for services with a token for which they *a priori* lack access faces a distribution problem. This fundamental initialization challenge exists even for *mineable* smart contract-based tokens, such as those used in Truebit [23]. Some blockchain projects politically circumvent this utility dilemma through *premining*, or initial distribution to a select group of individuals or institutions, however a private premine alone does not transform the system into a public good. Sections 3.2, 4.1, and 4.2 describe premining alternatives which leverage existing liquid tokens for distribution. This technique reduces friction for consumers,

who use assets readily available to them, while offering a potential source of revenue for project management and enhanced collaboration.

The governance game, as described in Section 4.2, determines in the short run tokens for use in bootstrapping and in the long run incentives for those holding governance tokens to convert them into utility tokens. Upon conversion of all governance tokens, a fully decentralized, yet upgradable system persists (Section 4.3). Teutsch and Reitwießner predicted some use cases for Truebit in early 2017 [23, Section 7]. Armed with a modern imagination, we now bring these ideas to a practical test.

2 Protocol review

This exposition describes a deployment method for the Truebit protocol [23], a blockchain enhancement which enables smart contracts to securely execute larger computations than the minimal gas limit permits [16]. *Task Givers* submit computational tasks, while *Solvers* and *Verifiers* ensure correct results in exchange for token rewards. Each task may have several Verifiers but only one Solver. The protocol runs on a unanimous consensus protocol among all Verifiers; there are no privileged or distinguished nodes. We shall largely treat the Truebit protocol as a black box, however the interested reader may refer to the project whitepaper [23] and code [4] for full details. While acquaintance with these specifications may prove useful, we aim to keep the present discussion streamlined and self-contained.

Tokens have three functions in Truebit: paying for tasks, staking to participate, and rewarding Solvers and Verifiers (Figure 1). Just as the identities of Task Givers, Solvers, and Verifiers may either be disjoint or overlap, so too may the actual *tasking*, *staking*, and *reward* tokens. The operating model, depicted in Figure 2, enables the possibility of disjoint tokens by minting rewards.

Solvers and Verifiers in Truebit must stake token deposits in order to participate in the network. This reserves financing for dispute resolution, in case some Verifier disagrees with a Solver's solution. Staking penalties, enforced in the framework of a *verification game*, disincentivize intentionally false or spammy solutions from Solvers as well as false alarms from Verifiers. When n Verifiers verify a task, then total token reward decreases by a multiplicative factor of 2^{n-1}, an *exponential dropoff* which dissuades Sybil attacks, and the protocol distributes the reward evenly among all participating Verifiers. In Section 5.2, we discuss an alternative mechanism with linear payouts rather than exponential dropoff. The Truebit protocol as described

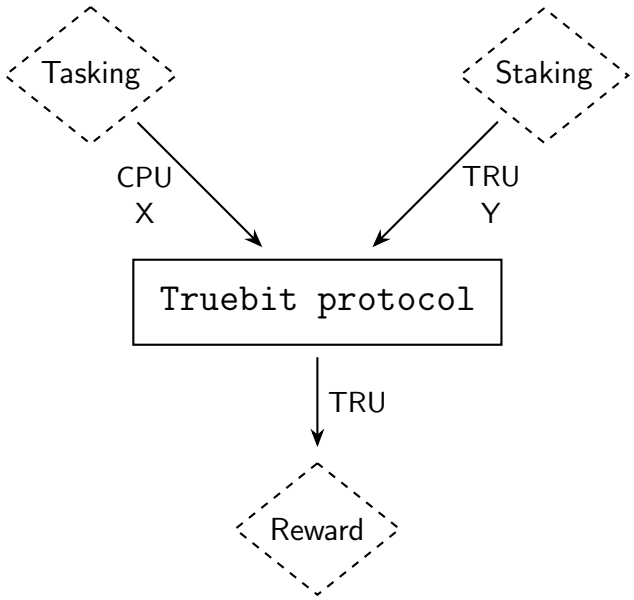

Figure 1: Simplified token functions in Truebit. CPU and TRU are primary tokens (Sections 3), whereas "X" and "Y" denote indeterminate, or "variable," tokens.

in the whitepaper uses a probabilistic reward scheme which pays Verifiers for discovering bugs. This *forced error* mechanism occasionally rewards Solvers for providing wrong answers and ensures that Verifiers get some rewards. The whitepaper describes a *jackpot repository* which cumulatively stores a portion of fees paid from each task in order to fund reward payouts in the case of forced errors.

The current Truebit implementation [4] can process task data from various sources, including IPFS [3]. If the raw data corresponding to the input IPFS hash is not made publicly available [21], then Truebit effectively processes private data via some subset of Solvers and Verifiers. A task which makes use of such a private input is called a *private task*. Goddard [10] first pointed out this innate privacy feature of interactive verification, and the Arbitrum project later popularized it [14]. The current Truebit implementation [4] processes both public and private inputs uniformly, cost-effectively, and faster than the authors of the original whitepaper could have imagined [17].

3 Basic token operations

We express the components of the token model through a series of self-contained but cumulative updates to the original Truebit whitepaper [23]. As indicated in Figure 1, the system primarily relies on TRU tokens for staking and uses CPU for task payments. A hard-wired preference for TRU rewards aids predictable task pricing, and the multi-token model achieves predictable economic effects by isolating the properties and functions of each system component.

The Truebit whitepaper describes an *underpricing attack* in which a Task Giver issues a task at a negligible price so that *rational miners*, or Solvers and Verifiers motivated primarily by token rewards, lack incentive to solve it [23, Section 5.5]. The Task Giver then solves the task himself, gets a bogus answer, and breaks the system's security. More simply, a Task Giver might, if left to his own devices, underprice a task simply to cut costs. Hence the protocol cannot allow Task Givers to price tasks under the assumption of rational miners. For similar reasons, due to potential Sybil identities, individual Solvers and Verifiers cannot securely set task costs for other participants.

In light of these requirements, Truebit relieves Task Givers from pricing responsibilities by fixing the cost for one computation cycle at one CPU. Thus Task Givers holding CPU may consume them at any time without exposure to price variation, while Solvers and Verifiers absorb potential market fluctuations through the TRU token (Section 3.2). This two token model decouples stability and market functions in a way reminiscent of GasToken [13] and without the protocol or governance system determining inflationary or deflationary rates; we assume that Task Givers holding CPU condone indeterminate token supply so long as it does not inhibit their ability to issue tasks and obtain correct computational results.

As a guiding design principle, the value of tokens paid into the Truebit protocol must not exceed the value of tokens paid out as rewards, lest Task Givers accumulate TRU rewards for spamming the network by solving their own tasks. Indeed, one can efficiently pass entire task rewards to oneself via private tasks as detailed in Section 3.1. Regardless of the quantity of TRU received as a reward, the reward received per computation step is always equivalent to one CPU according to the pricing provided by the reward recipient.

We expect that many Verifiers participate in the network in order to guarantee correctness of their own tasks, however these same Solvers and Verifiers may add security to other Task Givers by leaving their Verifier

or Solver client running between solving their own tasks. Thus participants who help the Truebit community may effectively use the system at negligible cost (ignoring network gas and real electricity expenses).

3.1 Pricing rewards

Rational miners wish to receive no less than the market rate for their services. We elaborate on Truebit's staking method in order to meet this requirement. Each participating Solver/Verifier, V, does the following prior to solving tasks.

1. V specifies a *local price* per computation step in TRU, p, to be paid to Solvers and to Verifiers, and

2. stakes *both* CPU and TRU tokens in a ratio of 1 to p.

For conceptual clarity, we have denominated reward payouts with respect to number of computation steps, however in practice the local price might represent *expected* price per computation step when taking into account the sporadic, cumulative nature of jackpot payouts.

Whenever a Solver/Verifier V earns a reward, the protocol *mints* to V the TRU reward allocation at the median price among all local prices, while the protocol *burns*, or permanently destroys, the Task Giver's corresponding payment. Median pricing, as opposed to localized pricing, avoids a potential tragedy of the commons wherein parties maximize their own rewards at the detriment of the tasking conversion price discussed in the next section. The system expects the local price to equal the "true" cost of a computation step in TRU, but what's to stop V from collecting unbounded rewards by choosing p as large as possible? The protocol incentivizes each Solver/Verifier V to provide correct market pricing through *Staking Monitors* who can *swap* tokens against V's stake at V's local price p. In other words, anyone holding a CPU token can exchange it for p TRU tokens against V's deposit, less a fixed, *universal staking conversion fee* of, say, 20% to dissuade swapping against reasonable pricing. In more detail, a *staking swap* proceeds as follows.

(a) A Staking Monitor M targets the deposit of some Solver/Verifier V with local price p.

(b) M submits n CPU (resp. TRU) to the Staking contract.

(c) Assuming V had staked at least np TRU (resp. n/p CPU):

i. M obtains $(1-c)np$ TRU (resp. $[1-c]n/p$ CPU) from V's stake, and

ii. V retains the n CPU (resp. TRU) submitted by M.

The protocol requires a *price bonding delay* of, say, 24 hours between the time that a Solver/Verifier initially commits or updates a price per computation step and the time that the Solver/Verifier can participate and receive rewards. This delay enables Staking Monitors to confirm the Solver/Verifier's local price before it applies in the system. At all times, the Solver/Verifier must have stake in the system in order to participate, however stake may be withdrawn at any time, including during the price bonding delay period. This allows Solvers and Verifiers to protect their deposits against potential price volatility.

In Section 3.2, we shall construct a more cost efficient means of converting from TRU to CPU tokens; staking swaps are only preferable in cases of outlier local pricing. Thanks to TRU's designation as reward token, one can achieve the reverse exchange direction, from CPU to TRU, via the method of private tasks.

Private tasks. In cases where only a hash of the input data appears on-chain, some nodes may never have access to a task's raw input. Private tasks offer rudimentary but effective security by obscurity and arise whenever the input to a Truebit task points to data outside the blockchain or permanent contract storage, e.g. through reads and writes to IPFS. For example, a Task Giver can supply an IPFS hash as input to Truebit's on-chain filesystem without actually uploading any public data to IPFS.

A Task Giver who submits a private task can effectively guarantee that he will be the only Solver and Verifier by ensuring that no one else has access to the raw input data. As no other party can correctly perform the requested computation, the Task Giver could win any verification game [23] that arises and therefore destroy the deposit of any opposing party. Thus, by issuing a private task and also solving it, the Task Giver can efficiently burn CPU and in its place receive TRU as reward.

Jackpot spiral. The astute reader may notice that minting TRU rewards "on-the-fly," permits periodic payment to Verifiers without requiring the existence of a jackpot repository to accumulate the necessary funding. In the Truebit whitepaper [23], the finite quantity of tokens stored in a jackpot repository bounds the maximum incentive for Verifiers and hence the size

of the largest, processable, secure computation. With TRU minting, in contrast, the Truebit protocol can, in theory, reward Solvers and Verifiers for performing tasks of any size. Since the construction above vanquishes the incentive bounding parameter, the TRU token itself now becomes a scalability solution.

Viewed another way, the design mitigates risks by eliminating politics of replenishing and growing the jackpot repository. Karen Teutsch pointed out that a finite jackpot repository without active, altruistic management is susceptible to *jackpot spiral*. Suppose that the Truebit protocol were to use a finite jackpot repository instead of a mintable TRU. If the value of TRU according to the pricing contract were to decrease for any reason, or if the jackpot repository's drops due to normal variance, then a previously executable task could become out of reach for the Truebit protocol. In all likelihood, this means that some DApp which relies on Truebit would stop working, which causes the price of TRU to drop further, which causes additional DApps to fail, which causes the price of TRU to drop further, resulting in an undesirable, hyperinflationary feedback loop.

3.2 Tasking token economics

Solver/Verifiers may wish to redeem rewards by issuing tasks. While Solvers and Verifiers receive rewards in TRU, Task Givers pay for computations with CPU. The system therefore requires an efficient means of converting TRU into CPU. Let us inspect the economics of this process.

The system prices *tasking conversions* from TRU in CPU using the median over all bonded Solver/Verifiers' local prices (Section 3.1). In short, Solvers, Verifiers, and Task Givers can burn TRU into a *tasking conversion contract* which instantly mints back an equivalent quantity of CPU tokens in the sense that one CPU token always pays for one computation step. More precisely, the protocol mints a quantity of CPU equal to the number of TRU tokens burned divided by the median local price. Note that a smart contract can efficiently maintain this median price in contract storage.

We now turn our attention to the system's relationship with external tokens and fiat currency. The protocol supports tasking conversions from tokens other than TRU as illustrated in Figure 2. A *Holder* in possession of XYZ tokens, who wishes to obtain CPU

1. stakes both XYZ and TRU in the tasking conversion contract, and
2. provides an exchange rate between XYZ and TRU.

Analogous to the procedure for Staking Monitors (Section 3.1), any Tasking Monitor can swap the Holder's deposit from XYZ to TRU or TRU to XYZ before the end of pricing bonding period. If no Tasking Monitor swaps the Holder's deposit in the allotted time, then the tasking contract proceeds to convert the XYZ deposit into CPU at the rate in Item 2 composed with the median price described in the previous paragraph.

The above construction assumes that the XYZ token has sufficient liquidity that a Tasking Monitor can not only estimate the correct price for XYZ but also has access to XYZ tokens with which to swap. Secondly, we assume that the Holder has some access to a modest number of TRU tokens, either from friends, through a Uniswap contract [5], or via whitelisted activities (Section 4). Lastly, the amount of TRU posted must be enough to make conversion worthwhile to the Tasking Monitor, e.g. equal to the Verifier TRU deposit for some standard task. The total value of the TRU deposited, however, need not equal the value of the XYZ deposit. For example, suppose that an XYZ Holder wants to mint 100 CPU tokens and that the Holder's declared exchange rate is 2 XYZ per 1 TRU. The Holder deposits 400 XYZ tokens but only 10 TRU tokens into the external conversion contract. A monitoring agent may then exchange 20 XYZ for the the 10 TRU tokens, or vice-versa.

We remark that an XYZ holder may burn XYZ tokens repeatedly into the tasking conversion, using the same TRU deposit, to obtain additional CPU tokens. Since this operation increases the net supply of tokens in the system, care must be taken to ensure that the supply grows in a controlled manner. We shall explore this matter further in Section 4.1.

The value of CPU in fiat. While one CPU token always pays for one Truebit task, the exchange rate between CPU and USD may fluctuate over time. On one hand, a CPU price increase may incentivize Task Givers to adjust their CPU purchases, task consumptions, and TRU reward accumulations, but it also attracts rational miners and therefore maintains system integrity. A CPU price decrease, on the other hand, requires more detailed analysis to justify profitability for Solvers and Verifiers.

A TRU Holder has options to hodl, sell, convert to CPU, or, following the latter case, issue a task. Consider the following pair of basic economic strategies for an active Solver/Verifier.

Strategy 1. Convert CPU as soon as it's earned. Use a price per computation step close the median *plus* 20% in order to maximize the amount of CPU obtained during conversion without attracting the attention

of a Staking Monitor.

Strategy 2. Hodl earned TRU. Use a price per computation step close the median *minus* 20% in order to drive the median price of TRU relative to CPU as high as possible without attracting the attention of a Staking Monitor.

Whether or not TRU inflation occurs relative to CPU depends on whether the majority of Solver/Verifiers select Strategy 1 or Strategy 2.

Below we describe reasonable conditions under which Strategy 2 is the long-run, dominant strategy for rational miners. In a nutshell, the set of hodlers following Strategy 2 eventually ends up with all the tokens while others are spending them on tasks. This decrease in market CPU supply ultimately drives up the price of CPU relative to USD.

Proposition. *Suppose that initially there exist n CPU tokens, including the equivalent value in TRU tokens according to the median price, and*

(a) at least p fraction of Solver/Verifiers follow Strategy 2,

(b) on average, no more than (the equivalent of) $x < p$ new CPU tokens are generated during each epoch of n tasks (as a result of either TRU deflation or external token conversions).

Then the number of tasks until the set of Solver/Verifiers following Strategy 2 hold all of the tokens is

$$\frac{n}{p-x}.$$

Proof. The expected time, as measured in tasks for the set of Solvers following Strategy 2 to acquire n tokens is n/p. During this epoch, xn/p new CPU tokens (including TRU equivalent) are created, and it takes xn/p^2 time to acquire them. By iterating this calculation, we see that the time required to acquire all tokens converges to the geometric series

$$\frac{n}{p} + \frac{xn}{p^2} + \frac{x^2 n}{p^3} + \cdots = \frac{n}{p} \cdot \prod_{k=0}^{\infty} \left(\frac{x}{p}\right)^k = \frac{n/p}{1 - x/p} = \frac{n}{p-x}$$

whenever $0 \leq x/p < 1$. If $x < 0$, then the expected time to collect all the tokens is bounded above by n/p. □

Due to the median pricing scheme for tasking conversions, at least half of Solver/Verifer nodes must follow the deflationary scheme from Strategy 2

in order for TRU value to increase relative to CPU. Solvers and Verifiers must each weigh pros and cons in deciding local prices. A median increase in TRU value relative to CPU effectively increases the available CPU supply. While an increase in TRU price grants TRU Holders greater purchase power and attracts Task Givers via reduced USD-equivalent prices, the change also reduces USD-equivalent rewards for Solvers and Verifiers. Thus local pricing brings into play both short-term economic and long-term network effects. We emphasize that the protocol's market values for TRU and CPU tokens neither appeal to external price oracles nor exchanges, nor do they depend on specific details of the reward mechanism (e.g. number of Verifiers per task).

Effect of exchange markets. We have argued that the Truebit token system functions as intended without exposure to external markets, however in general the protocol cannot guarantee that such markets will not materialize. We now consider potential effects of liquid exchanges and argue that they do not disrupt the construction. There are four possibilities depending on which subsets of the two tokens, TRU and CPU, are *tradeable* on public exchanges. Recall that a TRU holder may convert tokens to CPU by using the protocol's built-in options contract as opposed to trading TRU on an *exchange*.

We have already considered the case where neither TRU nor CPU is tradable. Now let us assume a liquid market for both TRU and CPU, and assume that an exchange value exists for each token. Let USD(TRU) [resp. USD(CPU)] denote the exchange market value for TRU [resp. CPU], and let the number r be the *tasking conversion rate* such that r TRU's convert to 1 CPU.

Claim. Assume a fixed tasking conversion rate r. Then USD(CPU) tends towards $r \cdot \mathsf{USD}(\mathsf{TRU})$.

Proof. If ever $\mathsf{USD}(\mathsf{CPU}) > r \cdot \mathsf{USD}(\mathsf{TRU})$, then rational actors will convert rather than trade, hence the market does not support this pricing. On the other hand, if $\mathsf{USD}(\mathsf{CPU}) < r \cdot \mathsf{USD}(\mathsf{TRU})$, then rational actors will use exchanges rather than converting. Task Givers continue to burn CPU while no new tokens are created, hence the CPU supply decreases, and therefore USD(CPU) increases. The claim follows. □

The remaining two cases, where exactly one of TRU or CPU is tradable are less interesting, however we include them for completeness. In either

case, exchanges add a sell option, but TRU and CPU prices remain incomparable. In case only TRU is tradable, then CPU can only be obtained through the task conversion contract. Then CPU's prescribed functionality persists, and the TRU price provides little distraction. On the other hand, a tradable CPU might make Solvers and Verifiers more inclined to convert from TRU into CPU, in which case they can simply modify their local prices according to the logic described above.

4 Bootstrapping the network

The CPU/TRU lifecycle steps described in the previous subsection presumes the existence of tokens, yet none are present at network initialization. It remains to derive the circumstances of genesis. We describe a bootstrapping method which conveniently minimizes friction for participation and takes advantage of flexible token supply. Sections 4.1 and 4.2 provide mutually compatible options for initiating and maintaining a supply of TRU and CPU. The methods described herein result in net inflation.

4.1 Staking with external tokens

For the purposes of initial distribution, in the short run, the protocol may permit Solvers and Verifiers to stake other tokens in place of TRU, using the same method outlined in Section 3.1, and Task Givers to burn tokens other than CPU as payment for issuing tasks under ad-hoc pricing. While the protocol permits withdrawal of stakes, Solvers and Verifiers would still retrieve deposits in the same currency in which they originally staked. In case the external tokens used already have a broad, liquid distribution, Truebit immediately becomes an accessible, public system. In order to facilitate Solver and Verifier participation and legitimate use, the system restricts private tasks to those issued with CPU. Once a sufficient initial distribution has been established so that Task Givers, Solvers, and Verifiers can access small quantities of TRU, this initial process would end.

Offering staking services opens a potential source of revenue with which to finance the development team's operations. The team can benefit from various business transaction structures, including fiat currency payments, value-in-kind services, and investment. Through this short-term program, external tokens enjoy greater demand through increased utility, visibility, and obliteration of supply via tasking.

Since the only way to obtain TRU tokens is through Solver and Verifier rewards, the post-whitelist staking mechanism above enables an apolitical

distribution of tokens devoid of management decisions, centralized trust, or passive income, in contrast to other common techniques such as exchanges, premines, initial coin offerings, [22], or airdrops [20]. Because "an airdrop may constitute a sale or distribution of securities" [12], the apparent lack of investment contract in this distribution method may offer regulatory benefits. Even for cases where an external token use is restricted or "locked" for regulatory compliance reasons, whitelisting it in Truebit can potentially increase its utility, thereby making the external token less like a security.

Following the remark made in Section 3.2 regarding external tasking conversions, the development team chooses relevant tokens XYZ on the same blockchain as Truebit and may bound tasking and/or staking functionality in at least one of the following parameters: the number of XYZ tokens useable per unit time, total time allowed for whitelisting XYZ tokens, and total quantity of XYZ whitelisted. Figure 2 illustrates the context for these governance interactions which, as we shall discuss in Section 4.2, need not be controlled by a centralized entity.

4.2 The governance game

Let us now consider a tokenized version of the governance mechanism outlined in Section 4.1. Assume some initial distribution of governance tokens, called DAO, with democratic voting power (e.g. [1], [7]) restricted to the following set of items:

1. whitelisting of variable tokens for use in the tasking conversion contract, and

2. assigning a maximum useable allotment of each such variable token.

As DAO tokens convert over time, the protocol roughly tends towards decentralization as the Truebit protocol's political hierarchy, along with the net TRU/CPU token supply variance, fully dissolve upon the ultimate DAO conversion. On the other hand, the last remaining DAO tokens, which could belong to anyone who acquired even a small holding, increasingly resemble unique souvenirs and may be slow to disappear.

The governance token, or DAO, expressly does not manage Truebit protocol upgrades (see Section 4.3), and in particular cannot assign TRU or CPU minting rights to any new smart contract. We enumerate the DAO token's crypto-idiosyncratic features which resemble the independent, concurrent material in [9].

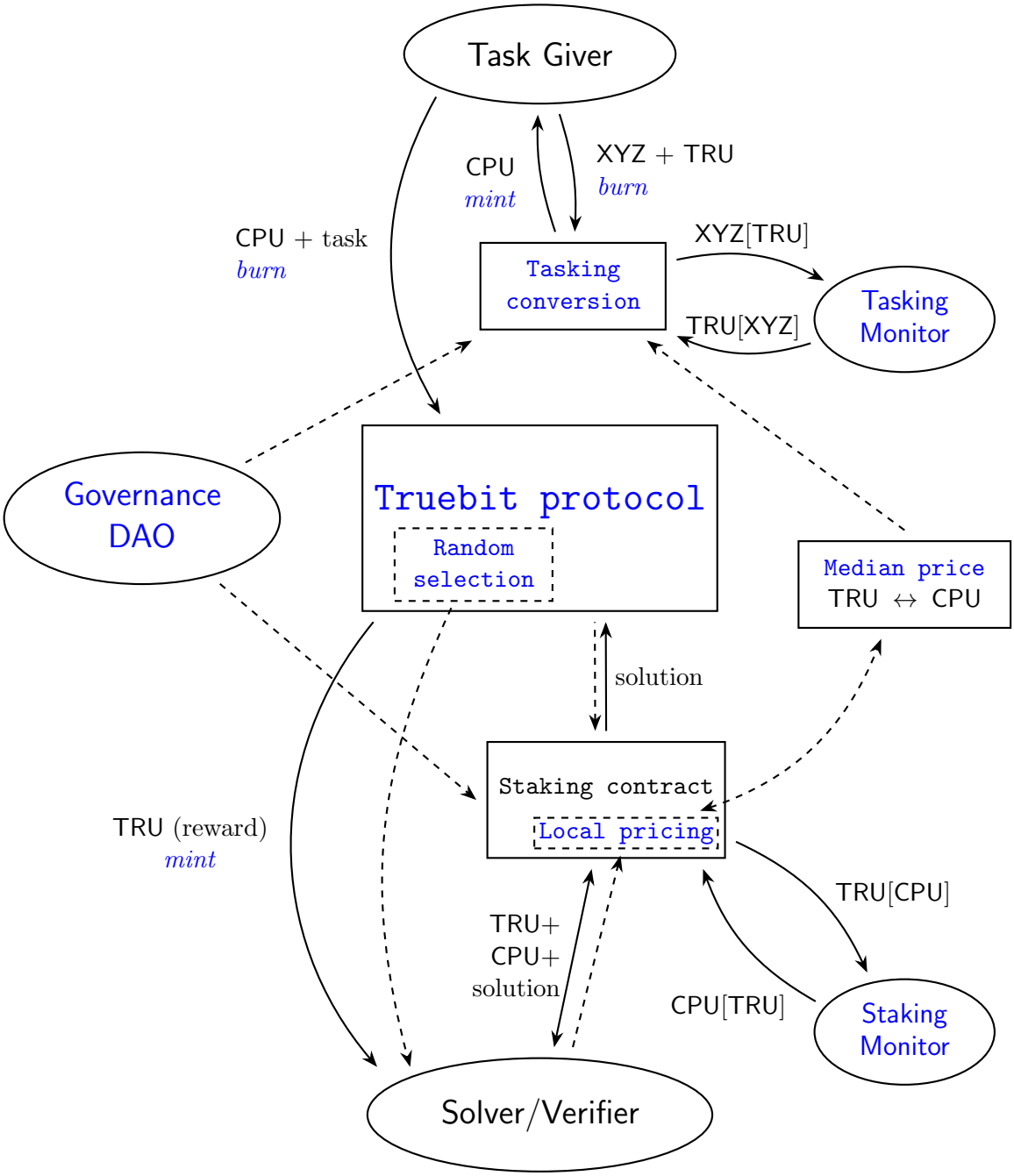

Figure 2: Illustration of Truebit token model. Solid lines indicate data and token transfers while dotted lines indicate control features. "XYZ" is placeholder symbol for external, whitelisted tokens, and square brackets adjacent to Monitors should be read as "respectively." Governance actions for tasking and staking include whitelisting external tokens and fixing their maximum useable allotments.

(a) Each DAO token, at the token holder's discretion, is eligible for a one-time conversion into either TRU or CPU.

(b) Each DAO token increases in its governance power over time because conversion decreases the total supply of DAO tokens.

(c) Later conversions receive a higher percentage of TRU or CPU tokens. More quantitatively, suppose that a DAO Holder has p fraction of the original DAO token supply and that c fraction of the DAO tokens have thus far been converted. Without loss of generality, assume that the Holder converts to TRU and that N TRU tokens currently remain. Then the token holder receives

$$f(p, c) = (p + 5c^2 p) \cdot N \text{ TRU}, \tag{1}$$

where "$5c^2p$" denotes the monotonic *bonus* for holding DAO long-term.

Let consider what might happen if p represented the fraction of currently remaining DAO tokens rather than the fraction relative to the original supply of DAO tokens. As a simple illustration, assume $f(p) = pN$, with $N = 100$ TRU, and suppose that the entirety of DAO holders decide to convert 50% of their tokens to TRU. They get 50 TRU, bringing the total TRU supply to 150. Now the DAO holders again convert 50%. This time, they receive $50\% \cdot 150$ TRU $= 75$ TRU. Hence, by converting 50% and then 50% again they receive a grand total of

$$50 \text{ TRU} + 75 \text{ TRU} = 125 \text{ TRU},$$

whereas if they converted 100% at first, they would only have received 100 TRU. Clearly obtaining an unbounded number of tokens from a finite amount of DAO is undesirable. We remark that the squared term "$5c^2$" in Equation 1 incentivizes long-term holding by back-loading the conversion bonus, while the number "5," on the other hand, is a somewhat arbitrary constant.

Don Gossen acutely noted that one could effectively apply the governance conversion process above without a whitelisting feature.

4.3 The upgrade game

Over time, network software may require amendments due to new fixes or features. While governance (Section 4.2) enables some foreseeable modifications, such as additions to the staking whitelist, others may require changes

to the underlying, immutable smart contracts. In the long run, the system requires a versatile update mechanism which preserves network effects and minting functionality without resorting to management decisions from a central authority. To this end, we propose a simple framework which invites new fixes and features without breaking legacy contract code.

1. A new version of Truebit protocol smart contracts is deployed.
2. The new contracts accept both CPU and TRU tokens but mint new CPU′ and TRU′ tokens for conversions and rewards respectively. The latter maintain tasking and staking functionality alongside legacy tokens.
3. DAO tokens can convert to the new CPU′ and TRU′ using the new contracts as well as to legacy tokens using the old contracts (Section 4.2).

For purposes of greater governance diversity, a DAO′ token might exist as well. Upgrades can also introduce new governance rules, e.g. imposing a limit on the maximum number of simultaneous tokens available for the staking whitelist. We remark that TRU, CPU, and DAO holders all have incentive to follow this upgrade pattern which preserves the original features of their tokens while enhancing functionality. In a successful upgrade, TRU (resp. CPU) gradually phases out of circulation in favor of TRU′ (CPU′), while TRU and CPU remain useable on both legacy and upgraded systems. When for example "CPU′ = GPU" represents a distinct hardware application, CPU and CPU′ might incomparably coexist with distinct pricing contracts. For additional flexibility, one can even transfer DAO tokens, and hence network effects, across blockchains via a two-way peg [8] or similar mechanism.

Practical example. Concurrently with the production of this paper, Harz, Gudgeon, Gervais, and Knottenbelt published a description of a reputation-based scheme called Balance which dynamically adjusts cryptocurrency deposits [11]. The authors mention that the Truebit protocol could benefit from including this design feature. The upgrade mechanism above incentivizes developers to experiment with Balance's adaptations to Truebit's staking mechanism.

5 Aleatorics

We conclude with two upgrade candidates for the Truebit protocol (Section 2) which crucially interface Verifiers with randomness and private tasks.

5.1 Martingale strategy

Let us examine the effect of private tasks on network security. Recall that both the Task Giver's cost and Verifier's reward depends on the computational complexity of the underlying task (Section 3.1, [23, Section 5.4]), and consider a Task Giver who submits a private task which costs 1 CPU followed by a private task which costs 2 CPU, then 4 CPU, and so on doubling the cost each time. Since the task is private, the Task Giver can verify it himself without concern for splitting rewards with other Verifiers. The Task Giver can even provide bogus answers through a Solver Sybil without spending real compute cycles or risk of getting caught. Eventually a forced error occurs, at which point the Task Giver more than recoups his expenses for this sequence by challenging the forced error with a payout proportional to the complexity of the last task times the reciprocal of the forced error rate.

The forced error method saves *gas*, or execution resources from the underlying blockchain network, in comparsion to distributing rewards after each task. Forced errors, however, must be sufficiently infrequent so as to make execution of the martingale strategy above prohibitively expensive because the martingale vulnerability exists whenever some rational attacker has enough capital to sustain the rounds of doubling capital. On the other hand, if gas is not a concern, the protocol can avoid Martingale attacks entirely by setting the forced error rate equal to 1, that is, imposing a forced error on every single task.

5.2 Random selection

In parallel to the traditional exponential dropoff payout for Verifiers discussed in Section 2, we add a linear reward payout scheme in the spirit of [15]. In addition to facilitating predictable reward payouts, this gas-efficient approach also improves security. This *random selection*, which we outline below, is uniformly compatible with private tasks. The Truebit protocol requires that tasks pass verification for both traditional exponential dropoff and linear random selections.

1. When a Task Giver submits a task requesting k Verifiers, priced proportionally, interested Solvers and Verifiers each execute it locally.

2. The protocol then randomly selects, according to the mining nonce, $k + 1$ Solver/Verifiers among those with registered deposits. Each selected Solver/Verifier has the opportunity to commit the hash of a

solution plus some committed random bits. The latter random bits prevent lazy copying of others' solutions.

3. Among the responding subset from Step 2, the protocol randomly selects a Solver, and the remaining nodes become Verifiers.

4. Solvers and Verifiers reveal their solutions. Verification games ensue in case of disagreement until either one Verifier wins or all Verifier challenges have been refuted. Unselected Solvers and Verifiers who submit solutions suffer penalties.

5. Absent any dispute, Solvers and Verifiers split the reward equally.

The protocol never penalizes selected Solver/Verifiers who cannot access the input data. Indeed, participation is optional in Step 2. Moreover, The construction achieves gas efficiency since only Solver/Verifiers selected in Step 2 who commit solutions actually interact with the blockchain. The protocol guarantees a predictable reward due to a bounded number of selected participants, and the only real cost for non-selected Solvers and Verifiers are watching events in a contract and performing off-chain computations.

Dandelion, Sam Moelius, and the Arbitrum paper [14] each described variants of the following vulnerability on Truebit's exponential dropoff scheme. An attacker convincingly, publicly declares to all Verifiers,

> "I will challenge all tasks of the form ABC roughly 100 times in case of a forced error. Don't bother verifying these, as the reward is negligible."

The attacker's goal is to get bogus answers onto the blockchain, and this becomes easier when he convinces other Verifiers not to participate. Random selection makes such an attack less effective, even under the assumption of entirely rational miners, as the attacker's Sybil attack has no effect on the incentives of chosen Solver/Verifiers.

Acknowledgements. We thank Ian Kovalenko for help with the DAO token design.

References

[1] Aragon wiki: Voting. https://wiki.aragon.org/dev/apps/voting/.

[2] The Dai stablecoin system. https://makerdao.com/en/whitepaper.

[3] IPFS is the distributed web. https://ipfs.io/.

[4] Truebit OS. https://github.com/TrueBitFoundation/truebit-os.

[5] Uniswap exchange protocol. https://uniswap.io/.

[6] Tether: Fiat currencies on the Bitcoin blockchain. https://tether.to/wp-content/uploads/2016/06/TetherWhitePaper.pdf, June 2016.

[7] DAOStack: An operating system for collective intelligence. https://daostack.io/wp/DAOstack-White-Paper-en.pdf, April 2018.

[8] Jason Teutsch Michael Straka Dan Boneh. Retrofitting a two-way peg between blockchains. https://people.cs.uchicago.edu/~teutsch/papers/dogethereum.pdf, October 2018.

[9] DAO Community & Friends. DAO - a next-generation decentralized organization that that coordinates the resources of a community (human, capital) to sustainably deliver value for members. https://gist.github.com/rzurrer/f5db72f427902300a2e030ccdfda641c, 2019.

[10] Tim Goddard. AdversariallyVerifiableMachine. https://www.reddit.com/r/ethereum/comments/51qjz6/interactive_verification_of_c_programs/d7ey41n/, 2016.

[11] Dominik Harz, Lewis Gudgeon, Arthur Gervais, and William J. Knottenbelt. Balance : Dynamic adjustment of cryptocurrency deposits. https://eprint.iacr.org/2019/675s, 2019. Cryptology ePrint Archive, Report 2019/675.

[12] Bill Hinman and Valerie Szczepanik. Framework for 'investment contract' analysis of digital assets. https://www.sec.gov/news/public-statement/statement-framework-investment-contract-analysis-digital-assets, April 2019.

[13] IC3. Gastoken.io – cheaper Ethereum transactions, today. https://gastoken.io, 2018.

[14] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. Arbitrum: Scalable, private smart contracts. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, pages 1353–1370, Berkeley, CA, USA, 2018. USENIX Association.

[15] Julia Koch and Christian Reitwießner. A predictable incentive mechanism for truebit. http://arxiv.org/abs/1806.11476, 2018.

[16] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 706–719, New York, NY, USA, 2015. ACM.

[17] Sami Mäkelä. JIT for Truebit. https://medium.com/truebit/jit-for-truebit-e5299afc72d8, December 2018.

[18] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

[19] Satoshi Nakamoto. Bitcoin P2P e-cash paper. https://www.mail-archive.com/cryptography@metzdowd.com/msg09959.html, November 2008.

[20] Doug Petkanics. Introducing the MerkleMine. https://forum.livepeer.org/t/introducing-the-merklemine/204, April 2018.

[21] Jason Teutsch. On decentralized oracles for data availability. http://people.cs.uchicago.edu/~teutsch/papers/decentralized_oracles.pdf, December 2017.

[22] Jason Teutsch, Vitalik Buterin, and Christopher Brown. Interactive coin offerings. https://people.cs.uchicago.edu/~teutsch/papers/ico.pdf, 2017.

[23] Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. https://people.cs.uchicago.edu/teutsch/papers/truebit.pdf, 2017.