Truebit Light's Incentive System

Christian Reitwießner
`chris@ethereum.org`

Blockchains can reasonably ensure that programs are executed correctly in a decentralized setting. They do not guarantee that a transaction is accepted, nor do they guarantee that an accepted transaction remains accepted, but the probability of a rollback decreases as time advances.

The main problem of blockchains is their scalability: The amount of computation that can happen per time is more or less fixed. It especially does not increase the more participants join the network, it is rather limited by the slowest node in the network.

TrueBit solves this problem through interactive verification. It allows more or less arbitrarily complex computations to be performed under the assumption that there is at least one honest participant. It does not require that participant to be altruistic, though. TrueBit also includes some drawbacks, especially the drawback that transactions usually take more time to be accepted and they can also be delayed arbitrarily by an attacker, as long as this attacker has enough financial resources. The system favours correctness over liveness, i.e. as long as there is at least one honest participant, it is impossible to include an incorrect computation, but an attacker can cause arbitrary delays to include correct computations.

This article wants to specifically address the Dogecoin-Ethereum bridge, which requires blocks from the Dogecoin blockchain to be verified inside an Ethereum smart contract. This verification is too expensive to be done directly, and because of that, we utilize TrueBit to take the computation off-chain.

We call TrueBit-light the protocol that does not make an effort to bring honest participants to the network. We assume that an honest participant is present who is altruistic to a certain degree. This includes keeping up with the Ethereum network, paying the gas fee and having a certain amount of money to pay for an initial deposit.

We also simplify the system to a degree to only allow one parallel task. This might be extended to a constant amount of tasks but would also make the presentation here more complex.

All timeouts are lower bounds and have to be extended in case of Ethereum network congestion. This means that if you want to claim a timeout to take effect, you have to provide a proof that severeal previous blocks had enough capacity to include a potential response by the other party. Since TrueBit never makes a claim that state transitions take effect in a finite amount of time, this is still consistent with the theory presented below.

The TrueBit contract has the following properties:

TrueBit consists of the fact claiming component and the verification game. In both cases, we fix a mathematical function $f\colon \Sigma^n \mapsto \Sigma^n$ which can be implemented on a given machine in s elementary computation steps. This is not a big limitation, since f can be an interpreter for another machine, thus allowing to run arbitrary programs, which have a certain finite running time. Limiting the running time is a crucial component, although this limit can be magnitudes higher than what is possible to compute in a single block of the underlying blockchain.

We will start with describing a language that helps us treat the smart contract systems.

Definition 0.1. *Let $\mathcal{A}$ be the set of all accounts and $\mathcal{T}$ the set of timestamps (e.g. the natural numbers). We model a smart contract C as a state machine that can receive inputs from $\mathcal{T} \times \mathcal{A} \times I$ (timestamp, sender and input) and acts on this input by changing to a state from S and producing an output from O. Here, I, O and S are specific to each smart contract type. We identify the smart contract with its state transition function $C\colon S \times \mathcal{T} \times \mathcal{A} \times I \to S \times O$. The function is a partial function, i.e. the machine is able to reject certain inputs. Smart contracts reject any input whose timestamp is not larger than the previously non-rejected input. We also identify C with the iterated state transition function $C\colon (\mathcal{T} \times \mathcal{A} \times I)^* \to S \times O$, where we assume an implied initial state s_0 such that $(s_0, o_0) = C(\varepsilon)$. The iterated state transition function is then defined inductively as $C(I_n, (t, a, i)) = C(s', t, a, i)$, where $(s', o') = C(I_n)$.*

As a shorthand, we write $C\colon s \overset{i}{\underset{t,a}{\Longrightarrow}} s', o$ if $C(s, t, a, i) = (s', o)$.

A strategy *for a player $a \in \mathcal{A}$ is a function from $S \to (\mathcal{T} \times \mathcal{I}) \cup \{\bot\}$ (where $\bot$ means that the player does not make a move). For a strategy assignment $\mathcal{S}\colon \mathcal{A} \to (S \to (\mathcal{T} \times \mathcal{I}) \cup \{\bot\})$, a* game *$g$ in a smart contract C according to $\mathcal{S}$ is a sequence of moves, i.e. $g \in (\mathcal{T} \times \mathcal{A} \times \mathcal{I})^*$ such that $C(g)$ is defined and either $g = \varepsilon$ (the empty game) or $g = g' \cdot (t, a, i)$ such that g' is a game in C where $C(g') = (s, o)$, $\mathcal{S}(a)(s) = (t, i)$ and there is no $a' \in \mathcal{A}$ such that $S(a') = (t', i')$, $t' < t$ and $C(g' \cdot (t', a', i'))$ is defined.*

For $\mathcal{S}\colon \mathcal{A} \to (S \to (\mathcal{T} \times \mathcal{I}) \cup \{\bot\})$, we write $C \rightsquigarrow_{\mathcal{S}} o$ if for any game g in C according to $\mathcal{S}$ there is some s such that $C(g) = (s, o)$. For a single strategy function $s\colon S \to \mathcal{T} \times \mathcal{I} \cup \{\bot\}$ for a player $a \in \mathcal{A}$ we write $C \rightsquigarrow_s o$ if $C \rightsquigarrow_{\mathcal{S}} o$ or any $\mathcal{S}$ that satisfies $\mathcal{S}(a) = s$.

Theorem 0.2. *For any function f taking s steps to compute, there is an interactive game with two participants a and b implemented by a smart contract $G[a, b, \cdot, \cdot, \cdot]$ with the following properties:*

1. *it takes at most $\log_2 s$ rounds and at most $t_G \log_2 s$ time (assuming no network congestion) for some intra-round timeout t_G*
2. *for any x and y, there is always a strategy s for player a such that $G[a, b, x, f(x), y] \rightsquigarrow_s = f(x)$*
3. *for any x and y, there is always a strategy s for player b such that $G[a, b, x, y, f(x)] \rightsquigarrow_s = f(x)$*

We now model the interactive game $G[a, b, x, y]$ that is played between two participants a and b and which either decides a claim that $f(x) = y$ with certainty in at most $\log_2 s$ rounds or results in a timeout. Let t_G be the intra-round timeout for the game. In case where there is no network congestion, G takes at most $t_G \cdot \log_2 s$ time.

Initially, a claims that $f(x) = y$ and the point of disagreement between a and b is between steps 0 and s and the game started at time t_0:

$G[a, b, x, y, t_0](\varepsilon) = (a, t_0, (0, x), (s, y))$

All following messages have to have a timestamp larger than the one in the state, i.e. we have an implicit requirement that $t > t_p$. Furthermore, we will omit the parameters of G in the following.

The claimant, a, has to start with providing a representation of the internal state (it can be a hash thereof) of f at step $\lfloor \frac{s}{2} \rfloor$:

$$G[\dots]\colon a, t_p, (l, s_1), (h, s_2) \underset{t,a}{\overset{s_3}{\Longrightarrow}} b, t, (l, s_1), (h, s_2), (\lfloor \tfrac{l+h}{2} \rfloor, s_3), \quad \text{if } h - l > 1.$$

Next, the challenger responds by choosing to either search in the left area (if he or she things that the internal state at step $\lfloor \frac{l+h}{2} \rfloor$ is not represented by s_m) or the right area (otherwise):

$$G[\dots]\colon b, t_p, (l, s_1), (h, s_2), (m, s_3) \underset{t,b}{\overset{\uparrow}{\Longrightarrow}} b, t, (m, s_3), (h, s_2)$$

$$G[\dots]\colon b, t_p, (l, s_1), (h, s_2), (m, s_3) \underset{t,b}{\overset{\downarrow}{\Longrightarrow}} b, t, (l, s_1), (m, s_3)$$

Note that $h - l$ is roughly halved each round. As soon as it reaches 1, the smart contract can actually perform the computation:

$$G[\dots]\colon a, t_p, (l, s_1), (l+1, s_2) \underset{t,a}{\overset{p}{\Longrightarrow}} \begin{cases} a & \text{if in the algorithm that computes } f\text{, the transition from step } l \text{ to } l+1 \text{ turns the internal state from } s_1 \text{ to } s_2\text{, potentially utilizing the auxiliary proof data } p\text{, and} \\ b & \text{otherwise} \end{cases}$$

Furthermore, a timeout can be claimed any time by anyone.

$$G[\dots]\colon a, t_p, \dots \underset{t,\cdot}{\Longrightarrow} b, \quad \text{if } t > t_p + t_G \text{ and}$$
$$G[\dots]\colon b, t_p, \dots \underset{t,\cdot}{\Longrightarrow} a, \quad \text{if } t > t_p + t_G.$$

The strategy for the honest actor: