The Boundless Protocol

The RISC Zero Team

Abstract

We propose *Boundless*, a decentralized market, aggregation, and smart contract settlement protocol designed to scale ZK-powered verifiable compute for every chain.

In Boundless, operators of GPUs and other commodity hardware called "provers" compete for on or off-chain requests for zero knowledge proofs issued by "requestors" via an efficient and open market. Such requests might occur either in the "spot" market or as part of a "service agreement" contract. In either case payment is settled in conjunction with the verification of the proof on the destination chain specified by the requestor.

We describe the following contributions: *Proof of Verifiable Work*, a fraud and spam-resistant mechanism for measuring the complexity, and therefore relative cost, of a zero knowledge proof which Boundless uses to incentivize proving via a token reward issued to provers as they engage in useful "ZK mining;" the *Financialization of Compute*, which leverages crypto-economic incentives to provide a reliable, capital-efficient solution to the problem of matching provers and requestors; and *Boundless*, a practical system built upon these ideas.

Contents

1 Introduction

The idea that computational resources can be traded is as old as the computing industry itself. Several models have emerged over the decades, culminating in modern approaches such as cloud computing and on-chain execution.

However, while cloud computing enjoys outstanding economies of scale, the same cannot be said for on-chain execution.

The difference is one of trust. A user of a cloud service implicitly trusts the service provider to perform the computation faithfully. In contrast, a user of a blockchain does not trust any particular node; instead they trust the protocol, which has mitigations in place for dealing with dishonest nodes. Unfortunately, the mechanism by which these mitigations work (re-execution by skeptical, incentivized validators) necessarily imposes limits on the amount of computation that can be performed on-chain. Demand for on-chain compute exceeds these limits and so the chain must have yet another mechanism to decide which of the requested computations it will perform. In Ethereum and others this is decided by the gas market, which simply auctions-off the chain's limited compute resources. While this mechanism has many desirable properties, it does not address the underlying problem (the scarcity of on-chain compute); instead, it "prices-out" all but the most high-margin requests, effectively sacrificing one workload for another rather than serving them both.

Modern approaches based on zero knowledge proofs (ZKPs) offer a better solution. ZKPs change the trust dynamic by making it possible for a node to prove that a given computation results in a given output. This makes it possible for a user, or other observer, to trust the result of the computation even if they don't trust the node who performed it. By changing the trust dynamic in this way, ZK obviates the need for skeptical re-execution, providing a path towards scalable, trustless, decentralized compute.

1.1 The road to ZK adoption

Though the foregoing argument has been known for some time, real-world adoption of ZK has been limited until recently.

During its early history, the field of zero knowledge proofs consisted of an eclectic bag of specialized tricks, each capable of solving a narrow class of problems. Public key digital signature schemes such as RSA and ECDSA are perhaps the most recognizable examples from this early era.

As time went on, advances in computational complexity theory, particularly in the study of probabilistically checkable proofs, provided strong theoretical evidence that ZK should be capable of performing general purpose computations. Practical algorithms – and a Turing Award – soon followed.

But these algorithms, while practical, were still a far cry from being easy to use. By their nature they were only accessible to people with deep expertise in applied cryptography and a bleeding-edge awareness of theoretical results. Still, the launch of Zcash in 2016 demonstrated that the technology was viable. This led to an explosion of research aimed at making ZK more performant, versatile, secure, and easy to use, which led to new technologies such as Bulletproofs (2018), Circom (2019), Halo (2019), and others. Concurrently, the development of STARKs, together with the release of the CairoVM in 2020, showed that ZK could be secure and programmable without requiring a trusted setup.

In 2022, RISC Zero released the zkVM, the first general purpose zero knowledge virtual machine based on a standard architecture (RISC-V). The zkVM revolutionized the industry: whereas ZK adoption previously required the use of exotic programming or circuit languages, the zkVM made it possible to use any programming language that supports RISC-V. This notably includes Rust, which has an outstanding ecosystem of tools and libraries, and which is seeing widespread adoption in blockchain and other industries.

One year later, RISC Zero showcased the power of the zkVM with the release of Zeth, the first "Type 1" zkEVM. Zeth took the industry by surprise. Previously, conventional wisdom had it that zkEVMs took years to build. With Zeth it was shown that a team of 3 people could do it in under a month. This disruption, together with the cumulative performance improvements seen in zkVM since its release, has triggered changes to project roadmaps across the industry: not only is ZK viable, but with zkVM it's fast and easy.

1.2 The last remaining challenge

Unfortunately, adopting ZK is rarely as simple as writing some Rust and running it in the zkVM. True adoption forces confrontation with the last remaining challenge: end-to-end integration. The essence of the challenge is that, whereas the gas market for on-chain compute is enshrined into the chain's protocol, access to ZK proving is not. Thus, teams seeking to adopt ZK must build their own pipeline for proofs. This is a non-trivial challenge, encompassing the lack of suitable decentralized infrastructure, differences in the control flow between standard on-chain execution and ZK-powered execution, incentive management, and issues associated with bootstrapping and retaining proving capacity.

The zkVM revolutionized the way people write ZK apps. What's needed is a similar revolution in how they are deployed and integrated.

2 *Boundless*

To address the challenges associated with ZK adoption we propose *Boundless*, a fully decentralized ZK-powered protocol containing smart contracts, novel cryptographic primitives, and off-chain infrastructure whose permissionless marketplace facilitates secure, efficient, and reliable execution for any protocol or blockchain via *the financialization of compute*:

- Its core is implemented by smart contracts deployed across any compatible blockchain.[1] Each deployment is independent and inherits the properties of the chain to which it is deployed, including decentralization, liveness, and crypto-economic security.
- It enables infrastructure providers to seamlessly engage with all of these deployments at same time, ensuring there's ample *proving liquidity* across all chains.
- It leverages the security properties of *zero knowledge proofs* and *proof of verifiable work* to enable trustless exchange of payment for compute.
- It is a *marketplace*. It solves problems such as price discovery, liveness, and censorship resistance by treating ZK-powered verifiable compute as a commodity that can be traded either directly (in the *spot market*) or indirectly (via *service agreements*).
- Its compute capacity scales linearly with the compute resources operated by provers: double the hardware, double the capacity.

2.1 The financialization of compute

In this section we explain the foundational ideas behind Boundless: (1) verifiable compute can be regarded as a commodity, and (2) proof of verifiable work makes it possible for that commodity to be trustlessly metered and traded. Boundless leans into these properties, leveraging market forces to facilitate permissionless and trustless trading and utilization of verifiable compute – and in doing so provides an efficient and reliable solution to the end-to-end integration challenges described above.

2.1.1 An abundant commodity

ZKP technology has reached a state where it is now possible to generate non-trivial ZKPs on consumer off-the-shelf hardware such as MacBook Pro and gaming PCs. Furthermore, the software for doing so is free and open source. With such a low barrier to entry, it is expected that verifiable compute will generally behave like an abundant commodity.

For users and protocols this means simple, dependable access to verifiable compute, at a stable price.

For industrial provers this means adding value through resource management, capacity planning, pricing, and reliability; by selling service agreements and other differentiated offerings; and by managing or coordinating pools of residential/amateur provers.

Lastly, for traders this means adding value by contributing to price discovery and by providing instruments that allow market participants to hedge against volatility (see *service agreements* below).

2.1.2 Trustless trading

Unlike physical commodities (where practically every dispute boils down to a *concurrently-observative intersubjective truth*), all of the relevant information regarding ZK-powered verifiable compute falls within the realm of *objective truth with bounded uncertainty* (i.e. probabilistic soundness under standard cryptographic assumptions). With properly implemented circuits and software (and with suitable choice of cryptographic parameters), this is practically the same as *objective truth*, which is why ZK-powered verifiable compute is so well suited for use in trustless contexts.[2]

[1]For didactic reasons, this paper restricts its attention to Ethereum and its EVM-based L2s.

[2]We take this opportunity to call for greater adoption of formal verification at all levels – from properties of the underlying theories all the way up to guarantees about real world

But this property alone does not allow verifiable compute to be efficiently traded. For that one also requires a trustless and secure way to meter the amount of compute that went into a particular task. In Boundless this metering is implemented by *proof of verifiable work*, a simple but robust mechanism that, among other things, allows provers to reveal the amount of work that went into a particular proof.

These properties – the objectivity of the result together with the integrity of the metrics reported by proof of verifiable work – enable trustless trading of verifiable compute.

2.2 A spot market for verifiable compute

We begin with a simple example. Alice wants to use an on-chain app that requires her to submit a ZKP. The ZKP can be generated by anyone with sufficient compute resources and access to the necessary data. Alice doesn't want to generate the ZKP herself and so begins looking for someone who can generate and submit it for her. Bob and Charlie are both interested in generating the ZKP for Alice as long as she's willing to pay enough to cover costs plus a reasonable profit.

Alice is able to describe the required computation and from this description it is possible for Bob and Charlie to estimate the amount of work that would be needed to fulfill her request.[3] Thus, from a market perspective, the only question is: who is willing/able to do it for the lowest price?

The Boundless Marketplace answers this question by facilitating a *reverse Dutch auction.* In a reverse Dutch auction, price discovery works as follows:

1. The requestor (Alice) announces their request in a public forum (the Boundless Marketplace), either on-chain or offchain.
2. Initially, the requestor is not willing to pay very much to have their request fulfilled. They are willing to let some time pass to see if anyone will fulfill it for a low price.
3. As time goes on, if the request is not fulfilled, the requestor increases the amount they are willing to pay.
4. Eventually, either:
    (a) The price rises to a level where someone (Bob, Charlie, or someone else) is willing to fulfill the request; or
    (b) The price rises until it's so high that the requestor is unwilling to go any higher. If there's still no takers, the request goes unfulfilled.

This approach has several benefits:

- Dutch auctions conclude once the first bid is confirmed.[4] This property makes Dutch auctions particularly well suited to low-latency situations as well as on-chain implementations.
- The "reverse" aspect of the auction results in a simpler protocol than an "ordinary" auction. In an "ordinary" auction provers would be required to advertise their prices and capacity. Doing this in a trustless, generic manner is not easy. The reverse auction avoids those challenges.
- The reverse Dutch exhibits desirable efficiency and stability properties. Crucially, it gets Alice the lowest possible price with low overhead.

Thus, when Alice needs a ZKP, she publishes a request to the Boundless Marketplace containing the following data:

implementations – and to thank our partners at Nethermind, Veridise, Runtime Verification, and the Ethereum Foundation for their significant contributions in this area.

[3]They can do this by fetching the necessary data and running the requested computation in a non-proving executor. This is relatively inexpensive to do, de-risks the request by ensuring the given data really leads to a successful result, and tells them exactly how much work is needed to generate the proof.

[4]To see why, note that the price function is monotonically non-decreasing. This means the first bid is guaranteed to be the lowest and thus the requestor has no reason to consider any later bids.

- A machine-readable description of the ZKP she needs generated (together with any other metadata needed to fulfill the request).[5]

- A timestamp indicating when the auction is scheduled to begin.

- A monotone, non-decreasing price function (denominated in ETH or ERC20).

- An optional bond (staked by provers seeking to lock the request).

- An expiration date/deadline.

After picking her request parameters, Alice publishes her request. For this she has several options:

- She can publish her request on-chain to the Boundless Marketplace contract (on the chain of her choosing).

- She can sign her request and publish it offchain to one or more separate protocols (a third-party gossip protocol, exchange, listing service, etc).

In either case her request will ultimately be fulfilled via the Boundless Marketplace contract on the chain specified by her request.

Once Alice's request has been published, Bob and Charlie both begin evaluating the work necessary to fulfill it. They evaluate the request based on the description (see footnote 3) the stated expiration/deadline, their current workload, other opportunities available in the market, their desired profit margin, the likelihood that someone else will be willing to do it for less, etc.[6]

Based on these factors they both pick a price above-which they would be willing to fulfill the request. Without loss of generality, suppose Charlie picks a lower price than Bob.[7]

As time goes on and the request remains open, the price programmatically increases as per the price function. Eventually the price approaches Charlie's threshold. When it does, he has two choices:

1. *Immediately fulfill the request.* Charlie can do this by racing to generate the proof and fulfill the request before the price increases to such a point as to attract competition.

2. *Immediately lock the request.* This operation grants Charlie the exclusive right to fulfill the request. This lock allows him to engage in the task of proof generation without fear that a competitor will swoop in and fulfill the request at the last second. To secure this exclusivity, he must lock-up some stake. He will lose this stake if he fails to fulfill the request before the given deadline.

In some situations Alice might prefer to disallow locking. She can express this as part of her request. In this case path (2) above would be prohibited.

When Charlie fulfills her request he is programmatically paid the *prover's fee*, which is equal to the *request's price* minus the *market's fee* (which is determined by the market's *take rate*, and which accrues to the market's *vault*). The request's price is determined by the price function at either the time of lock (if Charlie locked the request prior to fulfillment) or fulfillment (if he didn't). To save on gas fees associated with proof verification (and if permitted by Alice's request), Charlie can deliver the proof as part of an aggregated batch.

2.2.1 Fulfillment guarantees

When everything goes right, Alice pays the lowest possible price to receive her proof before her specified

[5]To save on on-chain data costs, the request metadata may include references to offchain data such as magnet links, IPFS links, HTTPS links, S3 links, links to DA platforms, etc. Long-term data availability is not generally required; in the spot market, it is expected that prospective provers will attempt to fetch the data before attempting to lock or prove a request.

[6]This evaluation can be automated through the use of models.

[7]Maybe Bob has a higher cost basis than Charlie, or maybe he just seeks a higher profit margin. Maybe Bob doesn't expect Charlie to pick a lower price, or maybe Bob has decided that the request won't be profitable even at its max price. In what follows, the reasons why Bob picked a higher price target than Charlie won't matter.

deadline. But what if somebody locks her request but fails to meet the deadline (possibly on purpose)?

If Charlie locks Alice's request but misses the deadline, then he will lose his stake, and Alice will be refunded. What happens next depends on how Alice configured her request:

- In the simplest case, the lost stake is sent to the market contract (where it can be distributed to token holders via the points mechanism) and the request's lifecycle is considered complete.

- Alternatively, if her request allows it, the request enters into a "retry" phase. During this phase, locking is disabled and provers are rewarded with a portion of the original prover's lost stake (the remainder being sent to the market contract). Assuming Alice tuned the staking requirement appropriately, provers will recognize the premium being offered and race to fulfill her request.

Other strategies are also available to Alice. For example, if Alice needs the proof *right now*, she can submit a request with a short deadline, and locking disabled, and a high, constant price. When Bob and Charlie see the request, they'll recognize the high premium being offered and race to fulfill it.

Alternatively, suppose Alice needs her request to be fulfilled within 3 days (i.e. some period of time that greatly exceeds the time required to generate the proof). In this case she could submit a request with a much shorter deadline (say, 12 hours), a reasonable initial price, an attractive maximum price, and a moderately high staking requirement. If her request fails to attract any interest, she still has plenty of time remaining to re-issue the request with a higher price or different parameters.

2.2.2 Decentralization

Proof generation at scale is an industrial process, and like other industrial processes, it has economies of scale. All else being equal, a data center next to a river is going to be able to generate more proofs at a lower cost than a pool of residential gaming PCs. In light of this, we now consider the following: what's to stop a monopoly from forming?

To answer this, we return to Bob and Charlie, the provers from our previous example.

Suppose Charlie out-scales the competition and grows to a point where he is responsible for the vast majority of proving. Bob is rarely able to lock a request and is effectively pushed out of the market.

If Charlie continues to fulfill requests at a reasonable price, then all is well. But what if Charlie starts raising rates or refuses to fulfill certain kinds of requests?

As Charlie abuses his monopoly, Bob starts to see an opportunity. Charlie's grip on the market is not absolute: he may have a monopoly on the proof market, but he does not have a monopoly on the hardware needed to *engage* in the proof market. Indeed, Bob already has the necessary hardware (a MacBook Pro or gaming PC will do); if the opportunity is great enough, he can even spin up a fleet of cloud instances for maximum effect. The software is easy to deploy and run. So, if Bob can make a profit by undercutting Charlie's prices or fulfilling the requests Charlie is ignoring, then he will step into the market and do so.

In the Boundless Marketplace, the goal is to provide users with a reliable service at a low cost. This goal is achieved by ensuring that people like Bob can easily use the hardware they already have to jump in and correct the market. This reliance on market forces (and permissionless chains) is the key sense in which Boundless is decentralized. The barriers to joining the market as a prover are low: the software is open source and works with commodity hardware, so that the proving supply is permissionless and ownerless. This ideally will provide sufficient "compute liquidity" to correct aberrations in the marketplace. Consequently, a request made to the Boundless Marketplace will be fulfilled at a fair price when there is at least one party for whom it is profitable to do so.

2.3 A token

To facilitate reliable, low-friction trading and trustless governance, the Boundless Marketplace includes a novel token called $ZKC with the following properties:

- A cap on total supply.
- A variable minting rate, ultimately controlled by tokenholders via on-chain governance, which determines how quickly the supply cap is reached. Newly minted tokens are distributed to provers as part of the *proof of verifiable work* reward mechanism (see below).
- Ability to generate points, which confer access to the fees collected by the market.
- Ability to stake in the spot market and/or service agreements.
- A variable reward frequency (controlled by governance) that determines how often tokens and points are minted/distributed.

We describe these concepts in greater detail in the sections below.

2.3.1 ***Proof of verifiable work***

As requests are fulfilled, the Boundless Marketplace inherently tracks various metrics related to reward distribution. Specifically, for each reward epoch, Boundless tracks:

- The cumulative fees collected by the market. This is a measure of value provided.
- The cumulative number of cycles proven by the market. This is a measure of work performed.

It also tracks these metrics on a per-prover basis. Thus, for any given epoch, for any given prover, it is easy to calculate the percentage of fees (resp. cycles) attributed to requests fulfilled by that prover.

Using these metrics, Boundless' smart contract programmatically rewards provers by distributing newly minted tokens to them in proportion to their contributions to the market. Consequently, the market provides provers with two sources of revenue: the payments they receive from requestors upon fulfillment, and the native rewards provided by proof of verifiable work.

But this begs the question: what does it mean to "contribute" to the market? While the definition of "contribution" will ultimately fall to tokenholder governance, at launch there will be two key metrics:

- Value delivered. By this metric, provers who are responsible for a large percentage of the fees collected by the market should receive a large reward.
- Work performed. By this metric, provers who are responsible for a large percentage of the cycles proven should receive a large reward.

These metrics are synthesized into a reward via the following rule: each prover is rewarded according to either their proportion of fees collected by the market[8] or their proportion of cycles proven – whichever is smaller.[9]

For example, suppose Bob is responsible for 25% of fees collected by the market but only proved 10% of the cycles. In this case he would receive 10% of the newly minted tokens. Conversely, suppose Charlie is responsible for 75% of the fees collected by the market but proved 90% of the cycles. In this case he would receive 75% of the newly minted tokens.

In the example above, we see that Bob and Charlie account for all of the fees and cycles, but only wind up receiving 10% + 75% = 85% of the newly minted tokens. The remaining 15% are simply burned, thereby reducing the minting rate.

The ability to track the number of cycles proven depends critically on a unique feature of RISC Zero's zkVM called *proof of verifiable work*. This feature gives provers the ability to include (in their proofs) metadata about the number of cycles that went into the proof. This metadata is cryptographically secure,

[8]Boundless supports payments in ETH or ERC20 tokens. However, for the purposes of the reward mechanism, only those fees denominated in ETH are considered.

[9]The fee proportion is ignored in the special case where governance has set the market's take rate to 0%. In this case provers are rewarded exactly in proportion to the number of cycles they proved.

meaning that the prover cannot manipulate/alter it without invalidating the proof,[10] and is also stamped with an unique immutable value (a *nonce*) that prevents the proof from being "double counted" by the reward mechanism.

2.3.2 The *Vault*

As alluded to above, the market takes a fee on all fulfilled requests. This fee is calculated as a percentage (called the *take rate*) of the request price. The take rate is configurable by tokenholder governance. Fees collected by the market in this manner are ultimately distributed to $ZKC holders. In this section we describe the mechanism by which that happens.

Recall that the market is deployed on multiple chains. Each of these deployments has its own fee pool, into which the fees collected by the market are deposited.

Each deployment also has a *vault* (implemented by an immutable smart contract) containing locked-up tokens. $ZKC holders can permissionlessly lock-up their tokens in the vault. While locked-up, tokens earn (non-transferable) points which can later be burnt to extract fees from the pool and/or unlock tokens held in the vault. The extraction/unlock mechanism is based on proportionality: an entity who burns 1% of all outstanding points will be rewarded with 1% of the fees in the pool (at time of burn); similarly, an entity who burns 1% of their points has the option to unlock 1% of their tokens at that same time.

In addition to point generation, tokens locked-up in the vault can also be staked into a service agreement (see below) or used to lock a request in the spot market. As with all tokens locked-up in the vault, tokens staked in this manner continue to generate points programmatically.

[10]Intuitively, this feature works by adding constraints to the zkVM circuits that effectively implement a monotonic counter whose value correlates with the number of cycles of (ZK) execution needed to generate the proof. The full details, such as the protections against malleability and the means of accounting for the cost of different circuits, are beyond the scope of this paper.

2.4 Verifiable service agreements

A *verifiable service agreement* is a binding commitment between a prover and a requestor, typically laying out terms by which the requestor can oblige the prover to perform some verifiable work.

Mechanically, service agreements are implemented by smart contracts that make use of Boundless' market, vault, aggregation, delivery, and reward mechanisms. By design, the universe of potential service agreements is left open: Boundless allows anyone to permissionlessly bring new kinds of service agreements to market simply by deploying a compatible contract. This flexibility allows for the creation of instruments tailored to different use cases while still enjoying the economies of scale and intrinsic rewards provided by the Boundless marketplace.

We illustrate the utility of service agreements with two examples.

First we return to Alice and Charlie. Recall that Alice, an end user, used the spot market to purchase a proof from Charlie (a prover). She knows she's going to make more requests in the future, so she reaches out to Charlie and offers a deal: she's willing to pay a small up-front sum in exchange for the ability to purchase 100 more similar requests, one at a time, at a limited frequency, sometime over the next year, at a predetermined price per cycle. They memorialize this agreement with a smart contract – a service agreement – that gives Alice trustless, disintermediated recourse in the event that Charlie fails to uphold his end of the bargain.

Charlie likes this arrangement because it establishes an easy payment rail with Alice while also helping him with his long-term capacity planning. Alice also likes this arrangement. On the one hand, the service agreement reduces her exposure to unpredictable increases in the (spot) price of proving. On the other hand, if she doesn't make as many requests as she initially planned, then she's out the cost of the service agreement but doesn't need to pay for her unused requests. Furthermore, if the terms of the service agreement allow for it, she can recover some of her costs by selling the service agreement to another user.

But Alice isn't the only party looking for a service agreement. For our second example we turn to Pam, a protocol developer who seeks to ensure there's ample proving supply for her core protocol. Specifically, she is looking for a service agreement that provides:

- A bunch of proving over the next year, possibly with sudden spikes, possibly with significant growth before the end of year.
- Timely delivery of proofs (to ensure the protocol runs smoothly.)

Pam reaches out to Charlie to purchase a service agreement, but this alone is not enough: no matter how reliable Charlie has been in the past, no matter how much stake he puts behind the contract, there is always a chance that Charlie might fail to uphold his end of the bargain. If this should happen, Pam always has the option of using the spot market as a backstop (Sections 2.2 and 2.2.2). However, given the enormity of her workload, she'd prefer to avoid "on demand" pricing.

So Pam seeks out others like Charlie and purchases service agreements from them as well. She *accumulates* these service agreements into a basket. This addresses her reliability concerns: by construction, no single point of failure can prevent this basket from delivering proofs. But the basket is also quite large, possibly several times larger than the expected workload. Thus she takes one last step and *fractionalizes* the basket. This gives her the ability to sell-off the excess capacity while also giving buyers access to the kind of high-end SLAs that are traditionally only available to large protocols.

2.4.1 A note on data availability

As discussed in Section 2.2, provers in the spot market engage with requests at their own discretion and are therefore free to ignore requests for which the data is unavailable (footnotes 3 and 5).

For service agreements, the situation can be more subtle. For example, suppose Alice buys a service agreement from Charlie that obliges him to fulfill her requests (and which slashes him should he fail to do so). What happens if Alice issues a request to Charlie but fails to provide the data needed to fulfill it? In general, there is no way for Charlie to *prove* that Alice withheld the data, and so it would appear (to the service agreement's smart contract) that Alice issued a valid request and Charlie failed to fulfill it. By exploiting this, Alice can cause Charlie to be slashed through no fault of his own.

To defend against this, provers must consider the data availability requirements implied by an agreement before entering into it. For example, if the agreement only applies to requests that can be served using data from permissionless sources (such as Ethereum, EigenDA, Celestia, etc), then the risk to the prover stemming from data (un)availability is relatively low. Luckily, most decentralized applications should naturally satisfy this requirement, as the relationship between data availability and progress is not unique to ZK.

3 Implementation

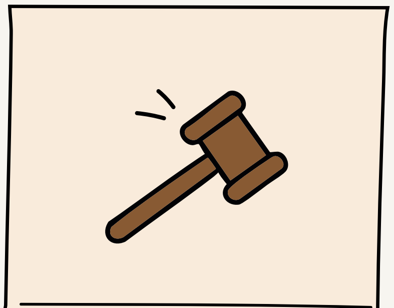
Boundless is being built in the open. See the requestor, prover, and developer docs, as well as the code itself, at `https://beboundless.xyz`.

4 Conclusion

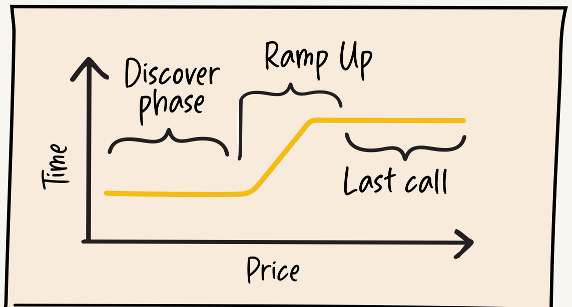
Much has changed since Satoshi published his 9 page masterpiece more than 15 years ago, but many things have stayed the same. The majority of the on-chain ecosystem is still rooted in 20th century cryptography. Recent advances in zero knowledge proofs represent a significant step forward and we propose that Boundless is the right way to realize their potential.

Requestor

Alice wants a proof

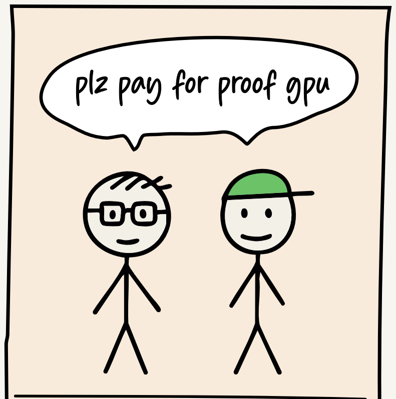


To match with a prover, she conducts a reverse Dutch auction

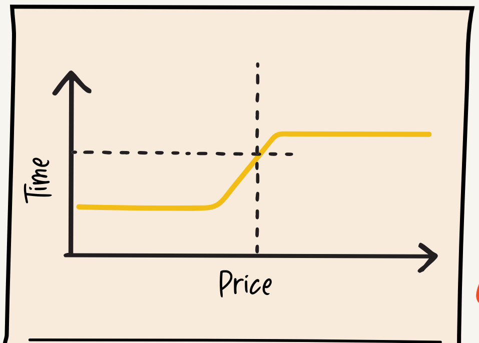


The price Alice offers goes up until either (a) someone agrees to do it or (b) Alice gives up

Provers

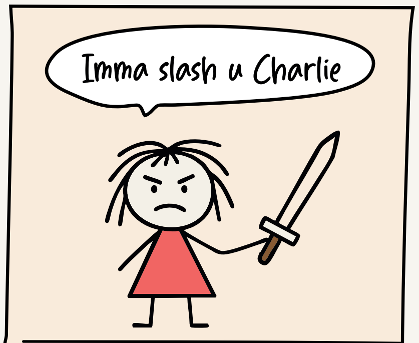


Bob & Charlie want to get paid to make proofs



Charlie locks Alice's request. He will lose his stake if he fails to deliver on time

Charlie misses the dealine

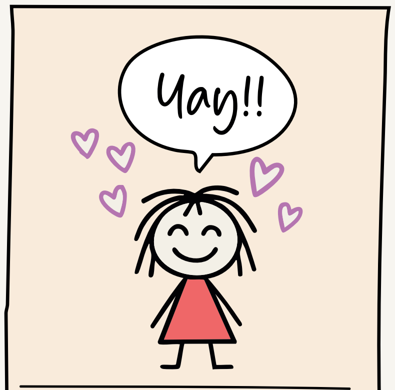


Charlie misses the deadline Charlie gets slashed; his stake is used to fund a new auction

Charlie delivers on time

Fallback auction succeeds

Result



Happy Alice