# Perlin: Blind computing for the masses.

Kenta Iwasaki

April 28, 2018

**Abstract**

Perlin is a decentralized network which aggregates, schedules, and frames a liquid market around underutilized computing resources to make massively parallel privacy-preserving supercomputing economically viable and accessible to enterprises, startups, students, and academics working on groundbreaking research and industrial applications.

## 1 Introduction

Supercomputers and other forms of high-performance computing have played a significant role in advancing scientific research and industry work; leading us to achieving a greater overall quality of life.

However, as more powerful forms of compute are introduced to alleviate the ever-growing computational costs necessary for the advancement of technology, they become far more economically inaccessible to those whom are the forefront of innovation. This stems from the fact that the market dynamics behind compute power and research in recent years has started to come under the influence, development, and control of just a few select centralized computing oligopolies around the world.

Such a problematic case has already affected researchers, students, and advocates in numerous academic fields such as machine learning, protein folding, genetic sequencing, supply-chain logistics management, and several others whose practical applications lie only economically viable and accessible to enterprises that can effortlessly expend large sums of capital. Large internet giants for example hold access to millions of computers and servers around the world and are slowly becoming the only ones financially capable of advancing and utilizing the state-of-the-art in computing (ex: quantum computing).

Perlin disintermediates traditional centralized computing oligopolies and replaces them with an openly accessible peer-to-peer network consisting of smartly aggregated underutilized computing resources that can be utilized to perform massively parallelized compute work. By framing a trustless, liquid market around said resources with an economic incentive, *perls*, we increase the maximum potential upper bound for parallelized computation to deliver egalitarian blind supercomputing at a feasible scale.

Perlin enables and democratizes access to large amounts of underutilized compute power for startups, enterprises, researchers, developers, students and advocates working with large amounts of potentially sensitive data for the sake of hastening the advancement of technology.

## 2 The Network

As an autonomously governed trustless system, Perlin consists of three core parties: the *validators*, the *miners*, and the *clients*. End-users of Perlin may partake in multiple roles at any given moment. Discussions with regards to how users partake in/withdraw from roles, how roles interact with one another, how economic incentives are minted, capped, and utilized, and how users are incentivized to responsibly adhere to their roles is discussed in the next section.

The *validators* are nodes of an asynchronous byzantine fault-tolerant distributed ledger that stores immutable records of events that occur within Perlin's network. Validators are responsible for verifying, timestamping, ingesting, and forwarding events over to other actors within Perlin's system, and thus are crucial for Perlin to converge into an eventually-decentralized system. The state of Perlin's network at any moment is derived by replaying through the records of events. Validators operate in such a manner such that the responsibilities of a single node when it comes towards querying and updating data is entirely segregated (command-query-responsibility-segregation).

The *miners* are nodes that provide underutilized computing resources in the form of virtualized container instances. Miners sell their computing resources with respect to the amount of time a client reserves/utilizes said resources for and solve cryptographic puzzles which allows them to prove to the network that they truly have provided the computing resources they claim to be able to supply for.

The *clients* are nodes that develop, utilize, and deploy decentralized applications and computing tasks within Perlin's network. Clients purchase available computing resources off miners, defining specific resource requirements they require for tasks that they wish to run on Perlin's network. Clients are the arbiters for how tasks are defined and executed off provisioned computing resources and are responsible for generating and assessing cryptographic puzzles which miners are to complete.

## 3 The Market

At the heart of Perlin's liquidity lies a virtual currency, **perls**. The dynamics with regards to how Perlin adopts liquidity, garners a supply of compute power, and encourages differing actors within Perlin's market to interact with one another are resultant of a basic supply and demand model.

We define Perlin's total supply of computing power $S_p$ to be a subset of the total external computing market supply $S_m$, and for both markets to have a

demand of $D_p$ and $D_m$ respectively.

Necessary assumptions are made that $S_m$ is relatively elastic with respect to $D_m$, and as a result initially leaves $D_p$ to be a miniscule proportion of $D_m$. Assumptions are additionally made that small price increases over the computing market supply $S_m$ despite lower marginal costs in existing underutilized devices leads to movement of sales towards smaller-scale suppliers. This movement of sales can be derived from how the economies of scale is invocated over the computing market.

Given that Perlin recycles underutilized computing resources such as long-idling laptops, smartphones, and tablets, $S_p$ inherently holds lower costs.

This initially leaves Perlin to be a price taker with respect to the external computing market and sets Perlin as a market with low opportunity cost. We label this supply curve as $S_p$.

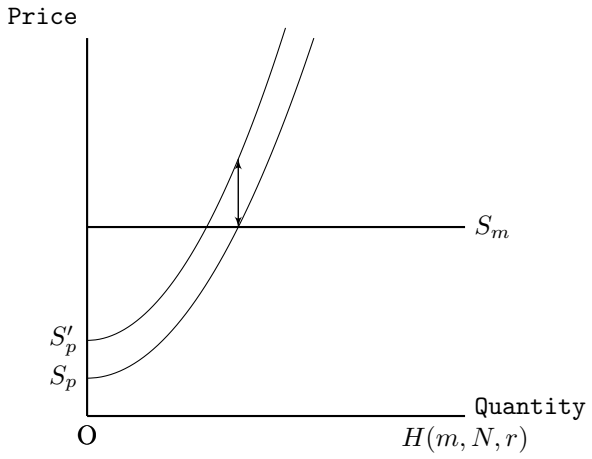


Figure 1: Supply-demand curve of the price of *perls* w.r.t the quantity of computing power measured in terms of $H(m, N, r)$.

As computing power in the hands of the consumer trendily becomes more powerful and available however, Perlin does require justification with regards how it's able to suave over the global computing market demand $D_m$ such that $S_p$ may further saturate $S_m$.

To accomplish the following, Perlin introduces a virtual currency, *perls*, which acts as an economic incentive to suppliers of $S_p$ which is logarithmically minted over time with respect to Perlin's present computing power supply and demand. *perls* are intended to have a higher market value compared to the fees expended by a supplier to maintain their computing power supply, such that suppliers are motivated to provide computing power despite there being lower demands at the beginning of Perlin's regime.

*perls* also act as an economic policy in which inflation is autonomously controlled, as *perls* are steadily minted over time from a pool with a maximum

capacity of 1 billion *perls* instantiated at the beginning of Perlin.

Thus, with the introduction of an economic incentive to wager against fees necessary to manage supply, we assume the supply curve will shift in. We label this new supply curve with the introduction of these incentives as $S'_p$, with respect to the global computing market supply curve as an equilibrium $S_m$.

## 3.1 Proof of Computing Resources

To establish a trustless computing market, a supplier must be able to prove they truly own a batch of available computational resources before delivering and selling them to another party. Perlin makes use of **diodon**, which is an asymmetric memory-hard cryptographic hash function to quantitatively determine that supplier truly holds some amount of computing resources.

### 3.1.1 Parameters of diodon

diodon's parameters $M$, $L$ and $\eta$ is configurable to allow for one to represent some amount of RAM capacity and CPU clock frequency available on a device. The work load factor $M$ enforces a block mixing process on diodon to have $M$ sequential round results sequentially stored and gated with one another in memory, and thus requires truly random memory access requiring long wall clock times and large amounts of memory.

Parameters $L$ and $\eta$ enforces longer periods of CPU time required for each block mixing round as it configures diodon to achieve larger time complexity from hashing mixed blocks $L$ times over $2^\eta$ random memory accesses.

diodon may be supplemented with an additional parameter, $p$, to be attuned to represent some level of parallelism which is possible on a device. Since the introduction of a scheduler requiring multiple block mixing rounds to coordinate one another introduces more work, $p$ can somewhat be used to enforce a longer CPU/wall clock time off a given device.

Although $p$ could be used to measure some number of virtualized/real cores available on a device, we choose to leave it out of Perlin's proof of computing supply due to Perlin's network inherently provisioning parallelism from millions of underutilized computing instances to clients/consumers. We leave $p$ for Perlin's use case to be 1.

With the parameters $M$, $L$ and $\eta$, we can estimate a lower bound memory requirement for diodon being $M * 128$ bytes. To simplify notation, we define the diodon function to hash a message m with the salt provided by a user as $H(m, M, L, \eta)$, given $p = 1$.

### 3.1.2 Cryptographic puzzle protocol

We start off with two potentially malevolent entities, Alice and Bob; with Alice wishing to supply spare computing power, and Bob wishing to purchase Alice's computing power.

For Alice to advertise that she is selling off her power, she emits an event containing a public IP to her computing resources to the Perlin network with proof that she holds access to computing power she claims to be able to provide. The proof is written as an appended signature to the event $H(d_a, M_a, L_a, \eta_a)$, with $d_a$ being a serialized message representing her device specifications in terms of $M_a$, $L_a$ and $\eta_a$, alongside access information to her supplied resources. By submitting this event, Alice has registered herself as a miner on Perlin's network.

Alice must follow this process using a miner node client from Perlin or any other entity, which will automate for Alice the process of segregating and securing underutilized computing resources into a virtualized container instance and automate the process of becoming and being a miner.

To automate choosing $M_a$, $L_a$ and $\eta_a$, Perlin tests upper bounds for $M_a$ and factorizes $M_a$ to be the smallest power of 2 that would saturate Alice's desired deliverable memory and processing power.

Now, any client who has developed a decentralized application or has a computation task which requires significant computing effort may have Alice process/work on a portion or on the entirety of their application.

Bob holds a task $T$ and registers $T$ onto Perlin's network with a fixed amount of *perls* to provide to miners $T_c$. Bob is now officially registered as a client on Perlin's network and may utilize Alice's resources for $T$.

Alice and Bob connect to one another provided Alice's public IP through SSH, and Bob may now use Alice's computing instance.

While Bob is connected to Alice, Alice and Bob must work with one another to solve a cryptographic puzzle such that Bob has constant access to Alice's computing instance, and Alice gets rightfully paid by both Bob and Perlin's network. The cryptographic puzzle goes as follows:

On a constant interval, Bob must generate, stream, and publish to Perlin's network a string of random bytes $b_c$. Alice must publish an event containing $H(m_a|b_c, M_a, L_a, \eta_a)$ as soon as possible with $m_a$ being the health and status of Alice's computing instance.

### 3.1.3 Economic incentives

Should Alice or Bob fail to submit their piece of the cryptographic puzzle in time, either Alice would not be provided pay from Bob and the network, or Bob would get disconnected from Alice and have 5% of his investment into $T$ recycled back into the network's pool of unminted *perls*.

Should one round of the cryptographic puzzle be successful, Alice would be paid $M = M_n + M_c(N, r)$, where $M_n$ is a logarithmically minted amount of *perls* from the network, and $M_c$ represents the total fees Bob pays from $T_c$ which amounts to:

$$M_c(M) = \frac{M}{10^{12}}$$

The minted proportion of Alice's salary $M_n$ is determined with respect to the total number of seconds required for Alice to submit the hash of her payout

request $H(m_a|b_c, M_a, L_a, \eta_a)$, which is an approximation of the number of CPU cycles Alice provided Bob.

As the transacted proportion of Alice's salary $M_c$ paid by Bob is proportional to parameters of $M$, $L$ and $\eta$ to a memory-hard cryptographic function, $M_c$ approximates the device memory specifications of the virtualized container instance Alice provided to Bob.

Given the volatility of underutilized computing resources however, before Alice receives compensation $M_n$, she must serve Bob for a minimum of 3 rounds for free as a bond.

The constant interval in which the cryptographic puzzle executes between Alice and Bob is described in the next section detailing Perlin as a distributed, asynchronous Byzantine fault-tolerant system.

# 4 The Ledger

For Perlin to track, audit, and validate the state of different actors in its network, Perlin requires a decentralized ledger which is immutable in the face of numerous potentially malevolent entities. In addition to that, Perlin requires an immutable source of time to execute its cryptographic puzzle protocol across miners and clients.

Perlin represents streams of events in the form of a historical graph of events synchronized across a quorum of validators.

To ensure Byzantine fault-tolerance of the contents of the graph, Perlin draws in inspiration from other forms of distributed ledger technology to securely achieve fast consensus in the finality of event sequences established by a decentralized autonomous government.

More specifically, with inspiration from distributed ledgers Hashgraph and Dfinity, Perlin models its ledger as a directed-acyclic graph whose consensus mechanism consists of a rotating quorum of a maximum of 1024 staking validators randomly chosen from an entire pool of candidate validators.

## 4.1 Hash Graphs

Drawing inspiration from Hashgraph, we divide events into sequences represented as rounds which are established whenever a single event is strongly witnessed'. Rounds define a measure of time in which a sequence of events were executed and are markers for the frequency in which the cryptographic puzzle between miners and clients should be executed.

We define an event $e$ to be strongly witnessed when:

$$|e_v| \geq \left\lceil \frac{2}{3}|v_r| \right\rceil \tag{1}$$

where $e_v$ is the set of all validators who notarized $e$, and $v_r$ is the set of all validators chosen in a given round $r$. We define an event $e$ to be notarized when a validator signs and gossips out an event.

An entity is considered to have gossiped out an event when an entity forwards an event to another entity; akin to how epidemic messaging protocols work in the context of distributed systems. Signatures are provided by a validator by hashing said event.

$$r_i = \{e_v : |e_v| \geq \left\lceil \frac{2}{3}|v_r| \right\rceil\} \tag{2}$$

To establish Perlin's historical graph of events, we formulate the data structure of the graph to be a directed-acyclic-graph representative of the hashes of all events published by Perlin's prior and present groups of validators.

The hash function utilized in Perlin $H'(m)$ is Keccak SHA-256, whose security comes from its unique preimage hash whose difficulty stems from the discrete log problem in mathematics.

Node identities in Perlin comprise of private/public keys derived from multiplicative and additive operations applied under an elliptic curve algebraic group defined by the Edwards 25519 curve such that public key $K_{pub} = K_{priv} * G$ where $K_{priv}$ is a node's private key, and $G$ is the base/generator point of the Edwards 25519 curve.

To minimize the amount of memory needed to cache and update the graph, hash signatures are appended together through the Schnorr signature scheme where signature $S(m, s, e) = H'(m|s * G + e * y) = H'(m|r) = e$ is represented through the signature components $s$ and $r = s * G + e * y)$ for message $m$.

The line of events which end users of Perlin may trust is a single line of events from said graph which has the greatest number of strongly witnessed events at any present moment.

$$p_g = \{\{e \in r_i\} \in R : max(\sum_{i=0}^{|R|} |\{e_v : |e_v| \geq \left\lceil \frac{2}{3}|v_r| \right\rceil\}|)\} \tag{3}$$

where $R$ is the set of all events, and $p_g$ is the ground-true sequence of events derived from Perlin's historical directed-acyclic event graph.

Hence, the selected group serves to linearize the sequences of events established and gossiped throughout Perlin's network, which will then eventually be immutably ledgered into Perlin's historical graph of events.

## 4.2 Random Oracle

Drawing inspiration from Dfinity, the hash of Perlin's genesis set of events consisting of a centrally selected set of validators will be used to seed a secure random generator $R_s$.

To select an initial group of validators during the genesis around, a Fisher-Yates shuffle is performed to permute over all initial candidate validators into a single group of a maximum of 1024 members that are responsible collecting, linearizing, hashing, and piecing together events which occur on Perlin's network to Perlin's historical graph of events.

For every perl a candidate validator stakes, they are singly ledgered onto a list that is permuted by the Fisher-Yates shuffle to determine a round's group of validators.

**Algorithm 1:** Perform Fisher-Yates shuffle to select validator(s) for round $r$ out of a pool of candidate validators.

**Data:** Array of $n$ candidate validators $a$.
**Result:** Round $r$'s validators $v$.
**for** $i \leftarrow n - 1$ **to** 1 **do**
  $j \leftarrow R_s$ s.t. $0 \leq j \leq i$
  swap a[j] with a[i]
**end**
**return** $v \leftarrow$ *pop from* $a$ *until* $|v| = 1024$

After a set of validators are chosen, provided node software will connect all validators together to form a small distributed cluster that will represent Perlin's network such that all parties of Perlin may monitor the present state of the network.

After a round has ended through the event witnessing scheme discussed in the last section, we concatenate and utilize the set of all validator signatures for the strongly witnessed event of the latest round $e_v \in r$ to reseed the random number generator that will then be utilized to choose the next group of validators managing Perlin.

Since the contents of the next strongly witnessed event is unpredictable and thus entirely stochastic, it is difficult for a malicious attacker to predict who the next round's set of validators will be.

Therefore, Perlin's recording of events by nature is asynchronously Byzantine fault tolerant as potential attackers lose information about the validator selection process over rounds.

Additionally, the unique causality of strongly witnessed events dividing sequences of events into rounds ensures that all finalized sequences of events within Perlin are resultant of democratic finality out of a stochastically determined group of validators.

Given that validation occurs out of a limited set of validators in quorum, Perlin can achieve fast finality of events and undergo rounds in the span of a few seconds.

## 4.3 Staking Mechanisms

To promote decentralization, anyone can register themselves to be a candidate validator in Perlin's system. One may submit a registration event into the network representing a minimum staking of $\frac{1}{1024}$'th of all minted *perls*, which will register them as a candidate validator in the next candidate set of sequences of events.

The reason for staking a minimum proportion of $\frac{1}{1024}$'th of all minted *perls*

is to limit possible candidate validators to a subjective percentile of the masses who can pay the high staking cost.

One may additionally leave the candidate validator pool at any time and unlock their initial stake through the submission of a leave event. Thus, those who were initially registered in Perlin's genesis may not be in the set of all candidate validators in future rounds.

The prime incentive for one wanting to be a validator in Perlin's system is that every honest member of the group of selectors that publishes one sequence of events into Perlin's historical graph gets an equal partition of all transaction fees derived from said published sequence of events.

These fees can rack up to a large sum of cash and does not require users to stake in any computing power or work for the sake of gaining back monetary compensation.

# 5 Blind Supercomputing

For Perlin to preserve the sovereignty of data it handles with developer/researcher flexibility in mind, Perlin chooses to establish a processing pipeline in which encrypted data is distributed and computed on in parallel across a number of possibly insecure miner virtual machines.

Should Perlin choose to secure miner virtual machines such that computations on encrypted data is privacy-preserving, Perlin must sacrifice developer flexibility by enforcing that developers/researchers create their programs and mathematical models in a restricted virtual machine which would constrain developers/researchers to utilizing an esoteric set of programming languages.

However, should an user really wish to establish that computations be privacy-preserving, an user may simply distribute an encrypted virtual machine alongside a batch of encrypted data to ensure that the entire processing pipeline end-to-end on a miner's virtual machine is completely secure.

Thus, Perlin provides developers and researchers complete freedom in which they may choose to protect, or to not protect the sovereignty of both their computations and data.

For Perlin to preserve the sovereignty and computability of data which is to be distributed and processed by a group of miner nodes, Perlin provides cryptographic utilities and software development toolkits (SDK's) implementing a wide plethora of homomorphic cryptosystems and parallel computation schemes.

## 5.1 Differential Programming

Numerous state-of-the-art techniques utilized in a wide variety of academic and industrial fields deal with developing mathematical models whose parameters are obtained through optimization techniques iterated over continuously differentiable objectives.

Examples of such techniques include predictive maintenance over supply chain management systems, phenotype modeling over drug discovery operations condoned by pharmaceutical companies, autoregressive time series prediction models of stock market features utilized by high-frequency trading firms and companies, and predictive user behavior modeling established by advertisement companies.

In broad terms, we can designate said techniques to all fall under an emerging academic field known as differential programming.

$$\min_{E(f(W,b,x),y')} f(W,b,x) \tag{4}$$

where $E(f)$ is a continuously differentiable objective function, $f(W,b,x)$ is a model parameterized over a set of multiplicative weight parameters $W$ and a set of additive bias parameters $b$, and a dataset consisting of a set of inputs $x$ and ground truth labels $y'$.

Perlin provides flexibility to its users on partitioning the dataset $(x,y')$, or the model $f(W,b,x)$ across miner instances with respect to potential gains in performance and throughput achievable out of massive parallelism.

To allow for differential programming over censored data, one may simply derive gradients computed off of $E(f)$ with respect to $f(W,b,x)$ given that additive and multiplicative homomorphisms are preserved on cipher text generated by Perlin's cryptographic utilities.

Hence, with homomorphic cryptosystems, one may train differential models for numerous objectives given that gradients necessary to update parameters for model $f$ with respect to encrypted cipher text $x'$ is equivalent to gradients with respect to raw inputs $x$ due to preserved homomorphisms.

Utilizing gradients $\frac{\partial E(f(W,b,x),y')}{\partial p}$ for parameter $p$, an user may then apply a $n$th-order iterative optimization algorithm over model $f$ to update parameter sets $W$ and $b$.

$$W_{t+1} = W_t - \mu \frac{\partial E(f(W_t,b_t,x),y')}{\partial W_t} \qquad b_{t+1} = b_t - \mu \frac{\partial E(f(W_t,b_t,x),y')}{\partial b_t}$$
$$\frac{\partial E(f(W_t,b_t,x),y')}{\partial p_t} = \frac{\partial E(f(W_t,b_t,x'),y')}{\partial p_t} \tag{5}$$

where $t$ represents the $t$'th iteration of a first-order optimization scheme with gradients scaled by $\mu$ being applied to a parameter $p$ such that model error indicated by $E$ collectively minimizes.

## 5.2 Distributed Systems

Distributed system components utilized in enterprises and startups such as databases, persistent messaging queues, stream processors, and virtualized container orchestrators are expensive to maintain and operate.

Monolithic variants of said components on the other hand hold a problematic issue of only being vertically scalable, and thus are more expensive to maintain.

Not only are these systems costly, but such centralization of technical infrastructure components makes achieving high-availability and fault-tolerance of one's distributed technical infrastructure subjective to the quality of computing power, hardware, and devops team one has.

Perlin introduces a novel application of decentralized ledgers in which it disintermediates centralized backbones underlying distributed systems deployed in centralized clusters such that establishing high-availability, consistency, and throughput of said systems is much more cheaper and easier to achieve.

Utilizing Perlin's network, one for example may host an eventually-consistent fault-tolerant database on potentially thousands of miner nodes for low costs. Data hosted over such a massively distributed system may additionally easily be censored, encrypted, and operated on given Perlin's homomorphic cryptosystem implementations.

Thus, with Perlin, any enterprise or startup may easily build fault-tolerant, highly available decentralized systems using Perlin's cryptographic utilities, parallelism utilities, and network of idle computing power.

### 5.3 Homomorphic Cryptography

Perlin provides two state-of-the-art asymmetric homomorphic cryptosystem variants: the Paillier cryptosystem, and CryptoNets. Paillier is a fully homomorphic cryptosystem which preserves multiplicative and additive operations; however, being computationally expensive to utilize on very large datasets and models. Paillier's security with respect to differential privacy is achieved through the discrete log problem in mathematics; akin to RSA and Diffie-Hellman.

CryptoNets on the other hand is a partially homomorphic cryptosystem which utilizes two noisy polynomial terms which phase one another out to partially preserve homomorphisms on data after numerous additive and multiplicative operation.

Using the properties of superpositions of high-degree noisy polynomials exponentially increases decryption error because of operations accumulating noise when sequentially applied to cipher text, though remains much cheaper to compute in comparison to the factorization of discrete logs necessary for Paillier.

In the face of machine learning problems which utilize stochastic mini-batch first-order optimization algorithms for updating model parameters, the error imposed over these superpositions however would be much less of a problem.

## 6 Conclusion

With Perlin's three major components being a self-audited distributed ledger, a cryptographic proof of computational resources, and a framework for highly parallel blind computing, Perlin commodifies underutilized computing resources

into highly parallel batches of computing power which can be utilized for advancing research meaningful to a plethora of industries.

Apart from creating a framework over these computing resources, Perlin also establishes itself as a foundation for both privacy advocates, communities, and industries with sensitive data to utilize state-of-the-art research results with minimal risk and cost.

By commodifying such resources into a decentralized virtual currency, Perlin can balance the disheveled supply and demand for computing resources across first world and third world countries.

In doing so, Perlin aims to shed new light to researchers and developers about the numerous practical applications decentralized systems have yet to provide for the world.