Silvermint: Scalable and Secure Blockchain Platform with Smart Contracts

Michael A. Stay, Ph.D
Pyrofex Corporation
stay@pyrofex.net

Nash Foster
Pyrofex Corporation
leaf@pyrofex.net

March 16, 2022

Abstract

We present Silvermint: a vision for implementing a **leaderless, scalable, and secure** proof-of-stake blockchain with smart contracts and censorship resistance. Current blockchain platforms suffer from significant performance and security limitations. Proof-of-work is inherently slow and forces node operators to build unusual hardware systems. Existing proof-of-stake blockchains sacrifice decentralization to improve performance. This can limit reliability and force users to trust network operators. They also neglect the opportunity for concurrency inherent in the network structure. Existing smart contract systems have difficulty ensuring both performance and security of decentralized applications due to low overall transaction throughput and limitations in virtual machine design. Silvermint resolves these issues by providing a platform that is secure and performs well.

Silvermint is based on the *Casanova* protocol [7], a lightweight distributed consensus algorithm that uses a directed acyclic graph (DAG) of blocks to establish consensus on both financial transactions and messages sent between smart contracts. *Casanova* scales linearly on the number of nodes and the bandwidth provided by those nodes. This results in a network that offers orders of magnitude more transaction throughput than existing blockchains. *Casanova* also provides very rapid confirmation and finalization of transactions, which helps ensure a satisfying end-user experience comparable to centralized transaction networks.

Silvermint includes a formally specified fast and secure virtual machine and associated programming language, both named *Symmetry*. These technologies ensure that smart contracts can be formally verified to an appropriate level of security. Silvermint further provides tools and support for sharding and shard management. Silvermint's unique sharding architecture allows the network to scale heterogeneously, thereby supporting a wide variety of decentralized applications with varying demands on the blockchain. The Silvermint architecture is constructed to make each component lightweight and independent, so that projects can adopt key features and technologies incrementally.

1 Introduction

Blockchain technology is a promising alternative to existing centralized financial systems, but adoption has been limited by performance, cost, and the difficulty of developing fast and secure applications. Silvermint addresses these challenges with innovative new technology capable of unlocking a promising future for decentralized financial systems.

The most popular blockchain systems, Bitcoin and Ethereum, are both based on proof-of-work consensus algorithms. While these systems ensure a trustless network, their performance has been disappointing. Bitcoin processes fewer than 10 transactions per second and Ethereum processes only 10-15 transactions per second. By comparison, the centralized Visa network is known to process nearly 2,000 transactions per second on an average day and may be able to handle up to 50,000 transactions per second at peak load. Hence, blockchain payment systems must scale up by a factor of about 10,000 if they are to be viable for global commerce. Smart contract systems will need to scale even larger to handle global-scale application traffic.

Proof-of-work is central to the performance problem. Node operators, known as "miners," have a strong incentive to build bespoke hardware solutions that maximize compute, or "hash rate," rather than building systems that help create a high performance network. Bitcoin miners are not encouraged to deploy systems that have high network throughput and low network latency. On the Bitcoin network, such bandwidth is mostly overhead that produces no additional profit to the miner. As a result of this design flaw, many people now incorrectly believe that decentralized networks face a trade-off between performance and centralization.

Even existing proof-of-stake systems suffer from performance issues because they typically use a linear block-based chain. All the major proof-of-stake chains are based on consensus protocols that produce a single block at a time, including Ethereum 2 [4], Cardano [3], TRON [2], Tendermint [6], Algorand [1], Solana [10], and Tezos [5]. A notable exception to the construction of linear blockchains is IOTA's Tangle [8]. Unfortunately, Tangle, like Bitcoin and Ethereum, has only probabilistic finality.

Silvermint's unique network architecture and leaderless consensus algorithm overcome these issues and are designed to maximize transaction throughput by ensuring that all nodes produce blocks simultaneously. Rather than competing with each other to produce blocks or producing blocks in rotation, Silvermint's protocol requires every node to produce blocks on a fixed time schedule. Throughput therefore scales linearly with the number of nodes in the network. And rather than consuming ever more compute through higher and higher "hash rates," the Silvermint network increases throughput primarily by consuming bandwidth. This is sensible, since an increase in transaction flows necessarily requires increased bandwidth consumption.

Silvermint also accepts or rejects individual transactions independently of the block that contains them. This "line item veto" of individual transactions means that attackers experience slow-downs when they double-spend, but the rest of the network does not. Silvermint is able to maximize the throughput of honest transaction flows, while attackers must pay transaction fees to be processed at slower rates.

Because Silvermint's resource consumption is better aligned with the network's requirements, Silvermint is far more environmentally friendly and sustainable without sacrificing decentralization.

1.1 Payments and Accounts

Silvermint uses a hybrid payment system that is based on unspent transaction outputs (UTXO), but also tracks information at the account level to help control the accumulation of "dust," which are UTXOs that are not worth spending individually. Silvermint supports a variety of user accounts types including single signature, multi-signature, hash time locked, and so forth. There is no specific transaction to create an account. They are defined entirely by the account type, account parameters, and the associated keypairs. In the simplest case, there is a single keypair and transactions must be signed with that keypair to be valid. More complicated accounts exist as noted.

The following is a list of possible account types, but as Silvermint is developed, more may be added as needed:

- **Single signature.** The most basic account type, with a single key holder.
- **Multi-signature.** There are multiple key holders, and a quorum must sign the transaction for it to be valid.
- **Hash time locked.** For the purpose of making half of a cross-shard or cross-chain payment.
- **Tumbler.** Allows a limited form of transaction privacy.
- **Multi-subject.** Arbitrary combinations of the accounts above.

1.2 Smart Contracts

Silvermint includes a virtual machine (VM) for smart contracts and associated programming language with a familiar syntax and a sophisticated type system. The design of the Silvermint VM and its programming language are based on an onion-like structure. The smallest language at the center implements the simply-typed lambda calculus, but is the most secure and the easiest to reason about. Subsequent layers expand the features available in the VM and its language, making programs more expressive and flexible at the cost of some security and performance guarantees. Silvermint includes a rich toolset for working with smart contracts.

Uniquely among smart contracting platforms, Silvermint seeks to offer a full and familiar programming environment, including a rich standard library, secure import semantics, code management and migration tools, and a natural syntax that is easy to learn.

Unlike other smart contacting platforms, Silvermint does not involve a separate currency or "gas" for executing smart contract transactions. Smart contract transactions are paid for in the native currency of a Silvermint shard. Since the capacity of shards is expected to dominate transaction demand, most smart contract messages should be delivered inexpensively. However, validators are permitted to terminate and ignore long-running computations, if the new state of a smart contract cannot be determined quickly enough to be cost-effective.

1.3 Efficiency

Silvermint is a highly efficient architecture designed to minimize the cost of transaction security and the overall maintenance of the network. Bitcoin mining has become dominated by those who can engineer bespoke ASICs that maximize hash rate per unit of power consumed. As a result, Bitcoin mining machines have no utility in any other computing domain. Ethereum uses an algorithm that requires both compute and fast memory, which puts pressure on node operators to build machines with large arrays of expensive graphics cards. Naively, one might assume that Ethereum mining machines could be repurposed to provide access to large-scale graphics compute in the cloud. In practice, however, this is not the case because miners have an incentive to choose the least expensive CPU, motherboard, and other components in order to maximize their mining rewards per dollar of capital expense. However, real-world workloads for AI require a fast CPU, a motherboard with a fast bus, and fast GPUs. Unavoidably, proof-of-work protocols create perverse hardware design incentives while providing disappointing network performance.

Blockchains can do much better. Silvermint aligns hardware design with network performance without sacrificing decentralization or transaction security. Because Silvermint does not rely on proof of work, node operators are encouraged to design and deploy systems that are efficient at handling blocks, which is the useful work of a decentralized blockchain. Every node operator in the Silvermint network needs to provide some compute and storage, but the operational workload for handling Silvermint blocks is quite similar to traditional IT workloads. Moreover, improved efficiency of the nodes themselves serves to improve the operator's profits without sacrificing security. We expect node operators will be able to profitably use typical IT hardware and cloud systems for Silvermint nodes.

While some compute and storage is necessary to operate a node, the most important limitation on throughput is related to the minimum bandwidth requirements for nodes. Silvermint requires all nodes to maintain a minimum useable bandwidth, which ensures that the aggregate decentralized network has the ability to exchange blocks at a scale that can meet global demand.

1.4 Scalability

Silvermint scales fantastically. It has been designed to handle hundreds of thousands of transactions per second on a single shard while maintaining a high level of decentralization. Each shard can support thousands of validators. At peak throughput, a single Silvermint shard can handle many times the maximum throughput available on the centralized Visa network. There are necessary tradeoffs between the number of validators, throughput, latency, operational costs, and security, but Silvermint's shard design supports a heterogeneous operating environment, allowing different trade-offs to be made on each shard.

1.5 Sharding & Cross-Chain Interactions

Silvermint supports running multiple shards, each with their own set of parameters. Silvermint shards are organized in a tree-like hierarchy much like a Unix filesystem. Each shard in the tree has its own parameters and security guarantees. The Silvermint top-level or “root shard” is intended to maximize transaction security and decentralization.

The root shard is a payment-only shard that every validator must validate. It is therefore the most secure. It supports the creation of multiple tokens, and for each token type it supports many different kinds of end-user accounts, including single signature, multi-signature, hash time locked, and tumblers. We believe this will handle the vast majority of real-world use cases. Silvermint also supports creation and management of independent tokens without the need for a specific smart contract. This helps ensure that tokens are fast and secure while remaining cost-effective.

Lower level shards may run smart contracts or be payment shards. Exchange between shards is handled by hash time locked contracts or accounts. Value backing currency issued in shards is held in multi-signature “vault” accounts in the parent, which may have arbitrary criteria for withdrawing those funds. Therefore, it can be important to validate all the way up the chain of ancestors to the root node for some applications.

1.6 Performance

As noted, Bitcoin handles fewer than 10 transactions per second and Ethereum handles 10 to 15. Sidechains speed this up somewhat, but at the cost of increased centralization and the possibility of rolling back transactions, both of which abrogate important design goals of the original Bitcoin protocol. We designed Silvermint to handle hundreds of thousands of transactions per second while finalizing transactions very rapidly. Silvermint’s protocol parameters are configurable on a per-shard basis, so performance can be tuned to meet each shard’s specific application demands. For applications that require higher throughput and lower finalization latency, for example, a shard’s block timing can be reduced to ensure that finalization occurs even faster than in the root shard.

Existing proof-of-work blockchains must maintain a “mempool” of eligible transactions that could be published in a newly mined block. Because the total throughput of such blockchains is relatively low, the mempool often contains many transactions and those transactions are communicated among miners through a gossip protocol. Because miners preferentially choose transactions offering the highest rewards for inclusion when they mine a block, users must both offer increased rewards and must wait until their transaction is gossiped to a node that is about to produce a block. This results in increased observable latency on the Bitcoin network.

On the Silvermint network, there is no real need for a mempool and no real need for nodes to gossip transactions. Nodes send transactions to each other only by way of blocks. Throughput is sufficient even at global scale that nodes are expected to often produce empty blocks. Under normal conditions, as soon as a Silvermint validator receives a transaction, it should be able to confirm the transaction will be included in the next block. Instead of using a gossip protocol for mempool transaction, end users can send the same transaction to multiple Silvermint nodes, in which case a confirmation from any of them is sufficient.

1.6.1 Throughput

Silvermint throughput is a function of the mimimum bandwidth provided by each validator, the total number of validators, and the number of blocks produced per unit time. Silvermint's root shard is expected to support more than 140,000 transactions per second when it is fully populated.

Each block contains information about the structure of the DAG and overhead for claiming rewards. When validators produce blocks more frequently, transaction latency can be very fast, but the bandwidth consumed by block overhead grows as a fraction of the total bandwidth. Choice of block timing, minimum bandwidth, and other shard parameters is a delicate choice that must be based on the specific needs of users and application developers who will use the shard.

1.6.2 Latency

Silvermint has extremely low end-user latency and confirmation times. Internet round-trip times notwithstanding, Silvermint accepts transactions in just a few milliseconds and finalizes them in about 2 seconds.

If a validator accepts a transaction, it must be published at the earliest opportunity, or it may be slashed by the user for censorship. Because of this, acceptance is sufficient to consider the transaction "complete" for many uses cases including nearly all retail payments, since the cost of censorship at this stage is typically larger than the value of censoring the transaction. The typical end user will experience latency similar to loading a small web page.

Once a transaction is published in a block, other nodes will begin building on that block. Intuitively speaking, when a fault-tolerant majority of nodes builds on a block two layers deep, then the transactions in that block are considered finalized. This takes about twice the block cycle time, which for the root shard is two seconds. Finalized transactions can never be rolled back and their order relative to other transactions in their conflict domain is fixed forever. All finalized transactions are complete for every use case, though some multi-transaction protocols such as hash time-lock contracts may require that several transactions finalize.

Eventually, blocks can no longer be referred to and those blocks can be archived and deleted from the node. Ethereum refers to this as pruning. This is more or less invisible to end users, but is important to node operators because it determines roughly how much storage is required for a node. Blocks may be archived as quickly as within three minutes, but in practice node operators should expect to keep them around for about a day.

In summary, the following are the relevant expected latencies for a transaction on the Silvermint blockchain:

- **Accepted** - $\approx$ 10 milliseconds.
- **Published** - $\approx$ 500 milliseconds.
- **Finalized** - $\approx$ 2 seconds.
- **Archived** - $\approx$ 1 day.

Comparison to Other Blockchains. For most proof-of-work blockchains, end user transactions must be included in a published block that becomes a part of the longest fork of the network. Typically, once one or more additional blocks have been mined on top of the block in question, the block is unlikely to ever be reverted. At that point, a transaction is described as being confirmed. For Bitcoin, the block cycle time is about ten minutes, and transactions therefore take 20 to 30 minutes to be truly confirmed. Ethereum has a much shorter block cycle time of about 10 to 15 seconds, but transaction confirmation times are still several minutes, typically. On both networks, confirmation can sometimes take many hours. This can occur if the network is congested or the fees offered by the user are too low.

Some more recent blockchains have focused on improving transaction latency and throughput. On a network like Solana, a transaction is considered "confirmed" when it is published in a block. Such blocks may later be abandoned if another concurrent fork is chosen by the cluster, so this concept of "confirmation" is much weaker than for networks like Bitcoin. Eventually, Solana transactions achieve consensus when a supermajority of the stake votes for their fork, but at minimum this takes several block cycle times just like Bitcoin and Ethereum.

Silvermint does not need to wait for a single miner or leader to produce a block to "confirm" transactions. In fact, it doesn't have to wait for any node to produce a block. Silvermint transactions are confirmed in real-time and so long as those transactions do not conflict with any others, this confirmation is sufficient to consider the transaction complete. In the case where a transaction conflicts with another, only the order can be changed.

2 Protocol Synopsis

In 2018, we published *Casanova*, a DAG-based consensus protocol for decentralized payment networks. We have now extended it into a protocol that also supports messages sent to smart contracts. Blocks are organized in a directed acyclic graph (DAG) and security and finalization properties of transactions are derived by performing calculations on the DAG.

2.1 Synchrony Assumptions

Casanova relies on a weak synchrony assumption. To wit, there is some finite upper bound on the time it takes for a message to arrive. The upper bound need not be known, but it must exist. Practically speaking, this means that if the network becomes partitioned, human beings will intervene to resolve the partition in a finite amount of time. Since partitions are typically formed by Internet outages, which must be repaired anyway, this assumption aligns nicely with reality.

2.2 Conflict Domains

In most blockchains, double spends are avoided by placing all transactions into a total order. However, for payment systems this is unnecessary. When *Alice pays Bob* and *Carol pays David*, it makes no difference which transaction is processed "first." Order only matters if *Alice* tries to pay *Bob* and *Carol* with the same UTXO.

This observation led us to develop the concept of a "conflict domain." Transactions can only conflict with one another when they are in the same conflict domain. Transactions that cannot conflict may be processed by the network in any order. For payments, most transactions do not conflict and Silvermint need not form consensus on transaction order in these cases. In the absence of conflict, Silvermint uses an attestation protocol, which is orders of magnitude faster and less computationally expensive. In this way, Silvermint maximizes transaction throughput while ensuring that nodes are rewarded for contributing bandwidth to the network.

In our payment protocol, end users provide a "check number" that orders their own payments. The conflict domain in the payments protocol is *transactions referring to the same UTXO or sent from the same wallet using the same "check number."* In the smart contract protocol, each smart contract is assigned an epoch number and the conflict domain is *messages sent to the same smart contract and same "epoch number" (see below). Casanova* uses a cryptographic hash function to deterministically order payments when there is a conflict. As a result, the order of transactions is deterministic for every block produced by a validator, based on its position in the DAG.

An elegant feature of the *Casanova* protocol is that only conflicting transactions are slowed down by the consensus protocol. Silvermint accepts transactions independently of the block that contains them. So, when two transactions conflict, the slower consensus protocol is run on just the conflicting transactions, rather than entire blocks. When an attacker double-spends, all other transactions proceed at the usual rate of the attestation network. As a result, producing invalid transactions does not cause a denial-of-service for other users. Attackers can make their own transactions slower, but they cannot affect the overall throughput or latency of the network.

2.3 Messages to Contracts

We extended *Casanova* to support smart contracts, and the updated protocol similarly operates as an attestation protocol unless it observes messages delivered to smart contracts within the same conflict domain. A smart contract's conflict domain is the smart contract's address and an "*epoch number.*" Each smart contract address has its own epoch number, which is tracked by nodes in the network.

When a node produces a block that contains one or more messages for a smart contract, each message is assigned an epoch number that is calculated based on the epoch numbers the validator has already seen. This can be computed deterministically for every block in the DAG, so validators cannot "backdate" messages for smart contracts without being punished by other validators.

Multiple nodes may produce messages in the same conflict domain around the same time. In this case, *Casanova* uses the same consensus protocol as before, but instead of choosing a single message to deliver during the epoch, by default *Casanova* orders messages by their hash and delivers the list in an order that does not vary between nodes. In this way, messages to a smart contract gain a global total order in the DAG and the smart contract's state is deterministic for any block in the DAG.

For some applications, e.g. high frequency DeFi contracts, this may still expose transactions to a limited form of front-running. In those use cases, the smart contract can request messages in the same epoch be delivered unordered by the protocol. The smart contract can then choose the order in which to clear transactions based on application level concerns. E.g., bids can be ordered based on their price. In this case, transaction confirmation is sufficient to ensure that the order will be processed appropriately by the contract. Anyone wishing to front-run your transaction must offer a better price and there is no arbitrage opportunity.

2.4 Finalization

Silvermint's consensus protocol features true finalization. Once a transaction has been finalized, it cannot be rolled back, even by a fault tolerant majority. On networks like Bitcoin and Ethereum, there is always a small chance of transaction rollback, particularly if "miners" decide to fork the network. On the Silvermint network, transactions are considered finalized whenever a validator has attested to a fault-tolerant majority attesting to a fault-tolerant majority attesting to the transaction. In the absence of conflict, this takes twice the block period. See our paper on the arXiv [7] for more details. The block period is intended to be configurable on a per-shard basis.

As noted already, Silvermint also features the ability to archive blocks that can no longer be referred to. This permits validators to save significant disk space in the usual condition where other validators are keeping up with network traffic. At any given point in time, a validator can construct a "snapshot" of the blockchain that contains only that portion of the DAG necessary to begin processing future transactions. In this way, new validators can come online rapidly without having to download the entire blockchain since genesis. This optimization requires some real-world coordination to ensure that the snapshot is correct and in practice such coordination is straightforward and secure.

2.5 Liveness

The Casanova protocol is provably live under the weak synchrony assumption that messages are delivered within a finitely bounded time. Moreover, even under a permanent partition, Casanova is live so long as a fault-tolerant majority of validators are live and honest with respect to each other. Since transaction finalization requires a fault-tolerant majority of nodes observing each other, Casanova can continue finalizing transactions even when substantial portions of the network are Byzantine.

Even when the number of live nodes in a partition drops below a fault-tolerant majority, transactions can still be accepted and published in blocks. It simply isn't possible in to ensure that they don't conflict with other transactions that will be finalized. In practice, this means that a node can continue accepting transactions even when its Internet connection has gone offline. Those transactions can't be sent to any other nodes until the connection is restored, but for many end-user transactions, this is an acceptable level of degraded service.

2.6 Failure Modes

Every consensus algorithm has failure modes. The safety and liveness proofs for *Casanova* assume that there is a bound f on the number of faulty validators; we then define a fault-tolerant majority in terms of f. The value of f is a shard parameter, typically one third of the validators. The proofs also assume that there is a maximum time delay for the delivery of messages; this bound on the delay must exist, but it need not be known. Practically, this means that partitions are noticed and eventually repaired.

Unlike the probabilistic finality of Bitcoin and Ethereum, Silvermint has true finality: once transactions have been finalized they cannot be undone. Under very restrictive circumstances, it is possible for an attacker to cause two parts of the network to independently finalize conflicting transactions. The security of the Silvermint blockchain relies on these conditions being difficult to achieve and economically disadvantageous.

For a detailed analysis of the failure mode, see the forthcoming Silvermint Specification document [9].

2.7 Networking

Like many consensus protocols, the *Casanova* protocol is usually bandwidth constrained. Silvermint requires that validators in the network supply a minimum amount of bandwidth for each shard that they service. In the root shard, this is initially expected to be one Gigabit per second. Silvermint then allocates this bandwith in equal parts to *sending block messages*, *receiving block messages*, and *catchup and other overhead*.

How blocks get delivered varies across blockchains. Many, e.g. Ethereum, use a gossip protocol to deliver both transactions and blocks. Solana uses Reed-Soloman codes to split blocks among downstream validators, thus saving bandwidth at the leader. All of these approaches improve bandwidth efficiency at the cost of increasing block delivery latency on the network.

Silvermint takes an alternate approach, requiring all nodes have sufficient bandwidth to deliver their blocks reliably to all other nodes in the network. A bandwidth requirement of 1 Gigabit/second for each node in a 2,000 node shard implies that a node can produce a block stream consuming about 1.25 Megabytes/minute. Such a shard can produce a total transaction stream consuiming around 2.5 Gigabytes/minute of bandwidth.

3 Economic Incentives

Silvermint offers its validators two primary forms of economic incentive. In the long run, the primary source of incentive is end user transaction fees. The cost to process and secure a single payment transaction is expected to be a fraction of a US dollar (on the order of a few cents), so user transaction fees can be affordable for end users while producing substantial income for the validator pool. However, this relies on the assumption that user traffic will be sufficient to cover the operating expenses of a shard's validators.

As a result, validators are also offered an incentive in the form of currency creation. This is analogous to Bitcoin's *coinbase* transactions, which are commonly referred to as "block rewards." Whenever a Silvermint validator produces a block, it claims its share of the overall block rewards as of that block. Validators are rewarded when other validators build on their blocks, so it makes sense for them to produce blocks on schedule and get them out to all other validators as quickly as possible.

Unlike on proof-of-work blockchains, all Silvermint fees are split between validators, and validators claim their portion using a share claim transaction when they reference a block either directly or indirectly for the first time.

Silvermint validators are free to follow any policy for prioritizing transactions. The Silvermint implementation will prioritize transactions based on the fees they provide, since this is in the best interest of the validator. However it is expected to be unusual, or at least transient, for a validator have a transaction queue that is non-empty.

3.1 Stake

Silvermint is a proof-of-stake protocol. When validators come online and wish to join the network, they do so by a process called "bonding." When a Silvermint validator bonds to the network, one or more UTXOs are consumed as the validator's "stake" in the network. The cost to stake a shard is a per-shard parameter. A validator's stake is held as security against the validator's good behavior. So long as the validator honestly participates in the protocol and meets the baseline performance requirements for the shard, the validator's stake will remain.

When a validator behaves incorrectly, they are said to be "equivocating." Any account holder may present evidence of the equivocation to the network, which will then punish the validator. This is known as "slashing," and typically results in forfeiture of the validator's stake and ejection from the pool of active validators. Usually, the account holder first reporting evidence of equivocation is provided a significant reward, which serves as incentive to monitor others for correct behavior.

3.2 Attestation vs. Voting

Casanova implements an attestation network where nodes publish information about what other blocks they have observed. Most other proof-of-stake blockchains use "voting" to accumulate stake behind a particular fork. Silvermint validators do not "choose between" two forks, but instead merely attest to the portion of the DAG they have observed so far. This attestation data, referred to as "DAG structure," is included directly within the block itself. Finality occurs when the DAG below a block has a certain shape.

3.3 Equal Stake

Silvermint is an equal stake network where the per-node stake is a shard parameter. Initially, it is expected that the root shard will be quite affordable to encourage widespread adoption, but over time as validator slots become more desirable, we expect the required stake to increase substantially. Silvermint uses equal stake for each node because nodes contribute capacity in the form of bandwidth and computation and because it maximizes decentralization.

Silvermint requires each node provide a minimum amount of bandwidth that is then allocated for various forms of messaging and recovery traffic. As a result, each node contributes throughput to the network. Most consensus networks work the other way: fewer nodes is prefered, because a larger population of nodes requires more messaging. In Silvermint, when a node operator has access to large amounts of capital, it is prefered that they run multiple nodes. In this way, they contribute not only stake, but also transaction capacity.

Fixed stake also promotes decentralization: because each node's stake is identical, they contribute the same amount of "weight" to finalization. There is no small group of nodes that can finalize transactions without the remainder of the network.

3.4 Stake as a Leaky Bucket

Because *Casanova* is a bandwidth constrained protocol, validators must be able to keep up with network traffic. To ensure validators will allocate enough bandwidth, stake drains away at a slow but constant rate. A validator's stake is restored at a faster rate by the block production rewards. Any block rewards and transaction fees earned by the validator first refill the stake, then go into an overflow account. If a validator cannot keep up, its stake drains away. Eventually, the validator is ejected rather than allowing its stake to drain to zero. Operationally, validators are given time to resolve whatever performance issue is preventing them from keeping up, so that the real risk to operators is minimized. But the network can and will eject nodes that fail to live up to performance and bandwidth requirements.

3.5 Censorship Resistance

When a transaction is sent to a validator, the validator accepts that transaction by providing a "confirmation receipt" to the end user. This receipt consists of a signature of the transaction's hash. If the validator then refuses to include the transaction in some future block, the user may present the receipt as evidence of censorship. When needed, end users can deliver anonymized transactions to validators and recieve receipts before revealing the contents of the transaction. This is implemented as a multi-step commit-then-reveal protocol.

Because end users are allowed to send the same transaction to more than one validator at a time, it is possible to obtain receipts for very high levels of stake. As a result, even when the value of censorship is high, it is economically infeasible to censor the network.

3.6 Block Production Rate

Validators must produce blocks with timestamps that are at least as far apart as the block production interval. If a validator produces blocks too quickly, it is evidence of equivocation, and they can be slashed. In theory, however, a validator could produce blocks in advance with valid timestamps and spam the network with them.

Nodes use their local time to determine when to process blocks arriving from other nodes. Up to some small amount of time skew, nodes should generally agree on the current time. Blocks arriving from other nodes will be processed only when the node's local time exceeds the timestamp in the block. So, even if a node produces blocks in advance, they will not be processed by other nodes until the correct time.

Fortunately, it is in a validator's best interest to produce blocks as close to the block production interval as possible. A validator claims block rewards based on the portions of the DAG referenced in his block. Validators are rewarded only when other validators build on their blocks and can only claim those rewards by referencing a block that builds the DAG in this way. As a result, validators are incentivized to get blocks out at the maximum allowed rate, but no faster, and to transmit them as quickly as possible to all the other validators.

4 Symmetry Language & VM

Silvermint uses a specially designed general purpose object oriented programming language and associated virtual machine (VM) called **Symmetry**. The goal of both the language and the VM is to provide a familiar and easy-to-use programming environment that nevertheless extends the utility of formal verification and correctness. Innumerable smart contracts on the Ethereum blockchain have significant and dangerous security vulnerabilities. And since a smart contract exists entirely in public view, these vulnerabilities are easier to find and exploit than a "black box" Internet service would be. Before a smart contracting platform can be used for industrial scale global commerce, it must provide sufficient tools to ensure that developers can tell when their code works correctly and when it doesn't.

The Symmetry programming language has an onion-like architecture. The core language implements the simply-typed lambda calculus. This is a total functional language that is not Turing complete, but whose semantics are very easy to reason about. Smart contracts written in this language always terminate in finite time and have no mutable internal state. This core of the Symmetry language provides the greatest guarantees to the programmer about the behavior of their program.

Subsequent layers of the Symmetry language remove guarantees provided by the inner layers. The outer layers are therefore more expressive, but harder to reason about. Programmers are encouraged to use the innermost layer possible for the task at hand.

Outer layers of Symmetry allow for parallelism, frame-local mutable state, transactional failure handling, arbitrary recursion and nonterminating loops, static names, reducers, write-once variables, and fully mutable state.

Different portions of a single program can be written in different layers of the Symmetry language. This is intended to give developers the ability to move back and forth between layers of the language in a natural and free-flowing way. For example, developers might begin prototyping using the most expressive outer layers of the language. Once a smart contract's behavior is better understood, portions of it can be re-implemented in stricter and stricter layers of the language to help ensure that critical behavior is well understood. Alternately, developers may choose to implement critical sections of their codebase in the strictest possible language from the very beginning as a forcing mechanism for correctness elsewhere in their smart contract.

4.1 Communicating Event Loops

Symmetry's VM adopts the communicating event loop model of interaction, which should be familiar to anyone that has written `JavaScript` for a web browser. Contracts receives messages, synchronously process them, and then emit a set of notifications. Other contracts in the same shard may be called synchronously. Cross-shard communication is by way of validators in the other shard watching for notifications from the contract. Cross-shard communication is necessarily asynchronous.

4.1.1 Type System

Symmetry is a statically-typed programming language. The Symmetry VM implements the same type system provided in the language. Specifically, the native type system generated by the operational semantics. More formally, the type system is the full higher-order dependent type theory of presheaves on the lambda theory of the language.

4.2 Preventing Laziness

One form of cheating in a smart contracting blockchain occurs when a node operator declines to perform computations associated with transactions. This reduces the physical resources needed to process blocks, thus increasing a node's profits, but at the cost of reducing overall network security.

To prevent validators from blindly accepting the claimed outcome of a contract without verifying it, validators are periodically issued a token to permit them to inject a fault into a block. If other validators detect the fault, the token can be presented to prevent the injector from being slashed. On the other hand, if other validators build on that invalid block without detecting the fault, the injector can provide the correct trace of the program to slash them.

4.3 Code Management

Every programming language must, eventually, address the questions of building, packaging, and managing the lifecycle of programs. Programming languages often take a relatively *laissez-faire* attitude that focuses on flexibility and usability. This is reasonable for normal operating systems, but not for blockchains.

Symmetry's dependency management features and tools are designed to help ensure that smart contracts are easily upgradable and flexible, while remaining secure and formally correct in the face of change.

Symmetry's include statements permit a rich collection of constraints such as version numbers and security guarantees to help ensure that newer versions of standard libraries do not degrade key features of the smart contract as a whole. As a result, Symmetry is able to permit both static and shared linkage, unlike other smart contracting platforms. For example, Symmetry can permit on-chain shared linkage when the target object is immutable. This can greatly reduce the on-chain storage needed for individual smart contracts.

4.4 Standard Library

At the heart of any successful programming language is a sophisticated and well-tested standard library of useful functions. Symmetry will include such a standard library and various versions of it will be available on the Silvermint blockchain for use by Silvermint's smart contracts.

5 Network Architecture

Silvermint's networking architecture builds on many years of experience in both distributed systems and blockchain architecture. The goal is to build a large-scale trustless transaction processing environment that nevertheless provides the most desirable security and performance features of a far more centralized platform.

Silvermint's underlying networking system is designed to be as efficient as possible on the wire in order to maximize the bytes per second available to the end users and minimize book-keeping overhead. Validators are organized into shards and communicate within and between those shards in a way that maintains the security and performance requirements needed by the network as a whole.

5.1 Validator Communications

Within a shard, validators must communicate blocks to one another in a timely manner. One of the strongest and most important commitments that a validator makes to the network is a commitment of dedicated bandwidth. Every shard has a parameter that specifies the minimum amount of dedicated bandwidth that must be available to validators within that shard in order to support reliable operations.

Most blockchains use a gossip protocol to promulgate blocks among network nodes. This is needed because not all nodes are aware of all other nodes. And, because most nodes have limited bandwidth. The result is that block propagation across the network is measured in minutes, rather than milliseconds. In Silvermint, nodes are fully aware of each other, because the list of all validators is always available when a validator produces a block. Blocks should be delivered to the entire network rapidly, so that most validators do not fall behind.

5.1.1 Non-validating Observers

Many functions in the Silvermint environment can only be implemented when there are services available allowing non-validator participants to query the state of the DAG. Any participant can provide slashing evidence if it observes equivocation, so account holders need to observe the state of the blockchain in order to finalize the results of hash time-lock contracts, and so on. This requires that participants be able to make inquiries and observations about the state of the network.

However, because validator bandwidth is heavily subscribed by block communications, it is likely that most validators will have little or no interest in providing real-time streams to more than a handful of users.

Network participants are encouraged to run observer nodes that can provide such streams to interested parties for a small subscription fee. We expect a part of that fee to go to one or more upstream validators and the remainder to cover the operational expenses of the observer node.

5.2 Sharding Architecture

Silvermint's network is organized as a root shard and a collection of subordinate shards organized into a tree. The root shard (or `/`) is the shard created at genesis and has some special properties. It is expected to be the shard with the most decentralization, the lowest bandwidth requirements, the highest total stake, and the greatest security. Conversely, finalization times are expected to be slower than subshards.

5.2.1 Staked Shards

A staked shard is a shard that shares its currency with its parent shard. Staked shards participate in the root currency. Unstaked shards with their own currency are also possible.

If a group of validators wish to run a shard with slightly different parameters from `/`, they can mount a staked shard somewhere in the shard tree. One example might be a group of US-based validators that want a shard whose governance will be entirely under US law. Such a shard might be mounted as `/us` and accounts and applications that wish to be governed under US law can be placed there.

This may make sense, for example, for media applications that must comply with the US DMCA. Any attempted DMCA-style censorship in the root shard would be seen as forking, resulting in slashing. But, nodes in the US may, strictly speaking, be required to adhere to DMCA takedown requests. The design and implementation of a DMCA-style takedown mechanism and its governance is beyond the scope of this paper, but one can envision a variety of approaches. As a general principle, any fault-tolerant majority can enforce governance and community standards on its shard, whatever those may be.

Each staked shard has a multi-signature account in its parent whose signatories are the staked shard's validators. This account is called the Vault. Such shards also track a multi-signature account with the same signatories that is called the Mint. Each Mint is allowed to issue UTXOs whose total value is less than or equal to the UTXOs in the Vault. If the Mint issues more UTXOs than currency deposited in the Vault, evidence can be produced of this and the validators can be slashed in the parent shard.

5.2.2 Transfers Between Shards

When Alice wants to transfer currency from a parent to a child shard, she makes a payment into the Vault. The Mint contract says, "Given a proof of payment from an account into the Vault in the root, issue that same amount minus fees into that same account here in the child." Suppose Alice pays, but the validators cheat so the Mint doesn't issue the money. Then she can present that same proof to the root shard. The validators need to provide proof of payment in a block. If the block is valid, then Alice has her money. If it's an equivocation, Alice can provide the conflicting block and slash. If they don't provide proof of payment after some time period, then they get slashed.

When Alice wants to move money from the child to the parent, it's similar. She pays the Mint. The Vault contract says, "Given proof of payment from an account into the Mint, release that same amount minus fees into that same account here in the parent." Suppose Alice pays the Mint, but the validators cheat in the parent so that the Vault doesn't release the money. (This would require the validators of the child to form a fault-tolerant majority in the parent). Alice can present the proof to the root shard, and it works as before.

Validators in a child shard may attempt to ignore Alice, never signing a block that has a payment to the Mint in it. Silvermint supports a commit/reveal protocol so that the recipient is unknown when the transaction is accepted and signed by the validators.

These processes are only necessary when changing the total amount of money in a shard. Any user may deploy a set of "money transfer agent" contracts. Money transfer agents provide inter-shard transfers that avoid parent shards, thus saving time. They will have accounts in all the shards, or at least in popular ones, and will engage in standard hash time-lock protocols allowing a user Alice1 to pay the service Agent1 in `/shard1` if and only if Agent2 pays Alice2 in `/shard2`.

5.2.3 Unstaked Shards

Many blockchain applications today seek to coordinate between two networks with different underlying currencies. Silvermint provides tools and APIs for mounting "unstaked shards" into the Silvermint hierarchy, so that other blockchains can communicate with the main Silvermint chain. In the simplest case, this might be another deployment of Silvermint itself that implements another currency, runs slightly different versions of the software, or has varying rules for validators. In more complex situations, it might be an entirely different blockchain such as Bitcoin itself. In any case, the objective is to allow the Silvermint blockchain to communicate with another protocol in a standardized and easy-to-understand way.

When an unstaked shard is mounted into the hierarchy, no stake is associated with the new shard. Therefore, the nodes participating in the unstaked shard cannot be slashed. In general, therefore, economic exchanges are made either through some form of currency exchange or through a hash time-locked contract that bridges the gap between the main Silvermint environment and the secondary blockchain. Silvermint provides transaction types and APIs to make this straightforward, but the details will vary depending on the properties of the unstaked shard.

6 Conclusion

We believe Silvermint to be a uniquely secure and scalable proof-of-stake blockchain platform that will provide both a global scale payments network and a useful application programming environment for the first time in the blockchain space. Our unique consensus protocol allows Silvermint to process orders of magnitude more transactions than other blockchains. Our uniquely designed programming language, VM, and associated tools will allow developers to author fast, secure smart contracts on which they can be build useful real-world applications. And because the hardware profile needed to implement nodes in our environment closely matches the usual IT load, Silvermint is both sustainable and cost efficient.

References

[1] "Algorand Consensus." https://developer.algorand.org/docs/algorand_consensus/#soft-vote

[2] "The Basics of TRON's DPoS Consensus Algorithm." https://coredevs.medium.com/the-basics-of-trons-dpos-consensus-algorithm-db12c52f1e03, Block generation and consensus.

[3] "Cardano." https://cardano.org/ouroboros#proof-of-stake.

[4] "Proof-of-stake (PoS)." https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/#how-does-validation-work.

[5] "The Tezos Consensus Algorithm." https://wiki.tezosagora.org/learn/baking/proofofstake/consensus#consensus

[6] "What is Tendermint." https://docs.tendermint.com/master/introduction/what-is-tendermint.html#consensus-overview.

[7] Kyle Butt, Derek Sorensen, and Michael Stay, "Casanova." https://arxiv.org/abs/1812.02232.

[8] Serguei Popov, "The Tangle." https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf

[9] Michael Stay and Nash Foster, "Silvermint Specification."

[10] Anatoly Yakovenko, "Solana: A new architecture for a high performance blockchain v0.8.13" https://solana.com/solana-whitepaper.pdf