Polycash: An Increased-Decentralization P2P Blockchain System

Asher Wrobel (asherwrobel@proton.me)

Abstract

The most widely used blockchains to date utilize the PoW (Proof of Work) or PoS (Proof of Stake) consensus algorithms. Unfortunately, these algorithms face a variety of issues. PoW decreases decentralization with mining pools and uses large amounts of energy, while PoS systems face lower resilience against so-called 51% attacks. In response to these issues, APoW(Adjusted Proof of Work) is introduced, using difficulty adjustment algorithms that increase decentralization and decrease power consumption in the blockchain. Combining this consensus algorithm with a new system of fork prevention, time verification, creates Polycash, a blockchain designed to allow secure transactions and the operation of peer-to-peer applications to be conducted with increased decentralization and security as opposed to traditional solutions.

1 Introduction

In the current day, financial institutions have largely become adopted as an intermediary for digital transactions. Using financial institutions relies on trust, an issue attempted to be resolved by cryptocurrencies. After these currency systems were established, blockchains began adopting a system of smart contracts to allow complex programs to be run using their infrastructure. However, blockchains' underlying mechanism, the consensus algorithm, is causing a increase in centralization, energy usage, and reliance on trust when using the existing PoW or PoS protocols. In PoW, large mining companies and pools have massive influence over the state of the blockchain, and cooperation between multiple pools could lead to a 51% attack on the network. In PoS, individuals with large amounts of stake influence the blockchain and the network in a similar manner. It also presents a feedback loop as entities holding large amounts of stake can restake their profits, netting them more gains. PoW also presents a large barrier to entry. Expensive mining hardware must be purchased to become a miner, which decreases decentralization due to the lower number of nodes. PoS suffers from the weak subjectivity problem, where the state of the blockchain up to n blocks in the past must be retrieved from a trusted source, where n represents the minimum amount of time funds must be staked before withdraws are permitted. The combinations of these issues in both PoW and PoS makes the need for a new consensus mechanism apparent.

To resolve these issues, a new consensus algorithm is needed: the APoW (Adjusted Proof of Work) algorithm. Instead of giving blockchain power directly proportional to mining power, APoW calculates block speed on a per-miner basis. It decreases target time (base 1 minute) by up to 50% for a miner with infinite mining rate, and increases target time by up to 50% for a miner with zero mining rate. This leads to an increase of decentralization in the blockchain and

the network, and allows for a lower energy requirement than with a standard PoW blockchain.

Another issue facing modern blockchains is the threat of quantum computing, specifically Shor's algorithm, which could, in theory, break the elliptic curve cryptography blockchains typically use for digital signatures. However, new alternatives have become available, such as the CRYSTALS-Dilithium family of public-key cryptography algorithms. Post-quantum public key cryptography algorithms must soon take on an important role in blockchain systems in order to future-proof their security guarantees.

A new blockchain, Polycash, is introduced to implement solutions to these issues. It utilizes the APoW consensus algorithm, Dilithium3 quantum-resistant signatures, and a new incentives system to increase decentralization, security, and efficiency.

2 Adjusted Proof of Work

To increase decentralization in the blockchain, a new consensus algorithm, Adjusted Proof of Work (APoW), is employed. The algorithm's aim is to increase the probability of a miner with a lower hashrate having the ability to create a block, and decrease the probability of a miner with a higher hashrate from doing the same. While the system is designed to maintain the higher chance of a higher hashrate miner winning a block, the gap between higher hashrate miners and lower hashrate miners narrows when using APoW.

The mechanism utilized by APoW to increase decentralization in this manner is block difficulty. As opposed to the traditional PoW algorithm, APoW calculates difficulty on a per-miner basis, attempting to adjust the difficulty for that miner every block so the time it takes them to mine a block matches a target time. To encourage miners to contribute their computing power to the network while still giving lower hashrate miners a chance of successfully mining a block, a modified version of the sigmoid function is utilized to alter the target time for a given miner based on the difficulty of the previous block they mined:

$$\frac{1}{1 + e^{\frac{-(x - mdpm)}{mdpm}}}$$

n DPM (Difficulty Points per Minute) represents the speed of a miner that can mine a block with difficulty n in 1 minute.

x represents the difficulty of the previous block the miner has created divided by the time it took to mine it, measured in DPM.

$mdpm$ is a constant representing 1 million DPM.

Dividing the difficulty by the result of the above formula allows faster miners a small advantage to encourage contributing compute to the network, but not to the degree of standard PoW systems.

To determine if a cryptographic proof of work is valid, the SHA3-512 hash of the block, represented in this case as an unsigned 64-bit integer, must be less than the maximum value of an unsigned 64-bit integer (18446744073709551615) divided by the block's difficulty. This means if block A has difficulty Da and block B has difficulty Db, and if $Db/\ Da = n$, then block B is expected to take n times longer to mine than block A. In other words, difficulty is proportional to mining time.

2.1 Block Reward Adjustment

In order to create a net loss of tokens for miners forking their hashrate, a block reward adjustment system is used. For each new miner that joins the network, the block reward decreases by 1%. This change makes miners lose profit when forking their hashrate, instead of gaining tokens. Although it is impossible to seperate the true identities of miners apart from their 'mining identities', public keys that are used for mining blocks, the block reward adjustment mechanism makes it unprofitable to hold over 100 actively mining identities.

About once per year, defined in the blockchain as once per 31536000 blocks, the block reward multiplies by 5 times. This keeps an active supply of PCSH present and prevents deflation. The increase doesn't nullify the 1% decrease mechanism, as this is a *proportional* increase of the block reward, and does *not* reset the 1% decreases in any way, offsetting them by a *constant* amount instead.

Formally, each 31536000 blocks is known as an 'epoch' and governs the way tokens are distributed according to the formula:

$$r = 0.99^m * 5^n$$

Where r is the block reward, m is the number of miners, and n is the current epoch, with the first epoch at 0.

We can prove that an increase in m carries over between epochs. Initially, two states, one with one more miner than the other, show the expected 1% differential:

$$r1 = 0.99^m * 5^n$$

$$r2 = 0.99^{m+1} * 5^n$$

$$r2/r1 = 0.99$$

$$\frac{0.99^m * 5^n}{0.99^m + 1 * 5^n} = r1/r2$$

$$\frac{0.99^m}{0.99^{m+1}} = r1/r2$$

$$\frac{0.99^m}{0.99^m * 0.99} = r1/r2$$

$$\frac{1}{0.99} = r1/r2$$
$$r2/r1 = 0.99$$

As expected, before any epoch transitions, the *r2* is 1% less than *r1*. After an epoch transition, the ratio remains the same:

$$r1_{new} = 0.99^{m} * 5^{n+1}$$
$$r2_{new} = 0.99^{m-1} * 5^{n+1}$$

Using the same method as before:

$$\frac{0.99^{m} * 5^{n+1}}{0.99^{m+1} * 5^{n+1}} = r1/r2$$
$$\frac{0.99^{m}}{0.99^{m+1}} = r1/r2$$
$$\frac{0.99^{m}}{0.99^{m} * 0.99} = r1/r2$$
$$\frac{1}{0.99} = r1/r2$$
$$r2/r1 = 0.99$$

And so the assertion holds. The 1% decreases in block reward each time a new miner joins the network span across the per-epoch increases in block reward and do not interfere with any existing mechanisms in the blockchain.

2.1 Cryptoeconomic modeling

Let a function, $m(b, s)$, be equal to the block reward at block number b with s mining identities. Let d equal the fraction of the block reward that is sold at market price by miners. Let h equal 1 - d. Let $a(b, s)$ equal:

$$a(b, s) = \sum_{n=0}^{b} m(b, s)$$

Thus, $a(b)$ represents the total amount of PCSH that has been minted by miners.

To model the reward per mining identity (assuming each identity is mining with optimal compute), we can use a function $r(b, s, s_a)$, where b is the block number, s is the number of mining identities, as with before, and s_a is the number of mining identities that are *active* and are being used to produce blocks.

$$r_{PCSH}(b, s, s_a) = 0.99^{s} * 5^{floor(b/31536000)}/s_a$$

This represents the average reward per block in PCSH. To get the average reward taking into account the current value of PCSH, the model must introduce a *base*

currency, which is assumed to have a fixed value. The value of PCSH relative to the base currency is v.

$$r_{BASE}(b, s, s_a) = v * r_{PCSH}(b, s, s_a)$$

From here, we can estimate the number of active mining identities. For this cryptoeconomic model, we assume that people will create new mining identities based on the square root of r_{BASE}, so we create a function, $s_a(r)$, which determines the number of active mining identities based on the result from $r_{BASE}(b, s, s_a)$.

$$s_a(r) = \sqrt{a}$$

To calculate the most likely configuration for the model's state, we need to define a function, $L(b, s, s_a)$, that calculates the total energy of the system. In the terminology of quantum mechanics, which often uses this method of calculating the most likely state for systems, this function would be known as the *Lagrangian*.

$$L(b, s, s_a) = |s_a(r_{BASE}(b, s, s_a)) - s_a|$$

This function calculates the energy of the system (the Lagrangian) based on the difference between the current s_a and the next calculated s_a. If the two are exactly equal, the system is in its *vacuum state* and in complete balance. If the two are farther apart, the system will soon experience a sharp correction back to a lower-energy state represented by a decrease in the Lagrangian.

We can solve for a zero Lagrangian:

$$\begin{gathered}
|s_a(r_{BASE}(b, s, s_a)) - s_a| = 0 \\
s_a(r_{BASE}(b, s, s_a)) = s_a \\
\sqrt{v * r_{PCSH}(b, s, s_a)} = s_a \\
\sqrt{v * 0.99^s * 5^{floor(b/31536000)}/s_a} = s_a \\
v * 0.99^s * 5^{floor(b/31536000)}/s_a = s_a^2 \\
v * 0.99^s * 5^{floor(b/31536000)} = s_a^3
\end{gathered}$$

The last formula is the formal definition of the PCSH Fundamental Economics Model (PFEM). Using it, it is possible to construct several proofs of economic phenomena that will occur in Polycash's economy.

2.1.2 Proof of Lagrangian Descent

For the formal definition of the PFEM to be useful, the Lagrangian must be equal to zero. Otherwise, the equation is invalid. However, due to market inconsistency and latency and illogical trading strategies, $L(b, s, s_a)$ will never be exactly zero. Instead, we can choose a value L_{err} and assume the PFEM model is valid when the Lagrangian is less than or equal to that value.

$$L(b, s, s_a) - L_{err} \geq 0$$

PFEMs that satisfy this equation are known as vPFEMs, the v indicating *valid*.

However, this method has a glaring issue; the PFEM is simply ignored outside of this range, and can only be used when the Lagrangian is very small. However, the economy will always fall into a state that can be described by a vPFEM (a *v-state*). We can prove this formally, but first more definitions are needed to describe how the PFEM evolves over time.

The PFEM can be described as a state machine. Every simulation step, it updates its internal state to represent the current economic conditions. This is accomplished by calculating the next s_a:

$$s_a(r_{BASE}(b, s, s_a))$$

This formula recalculates s_a using the $s_a(r)$ function, using the state machine's current state. Looking back at the Lagrangian, $L(b, s, s_a)$, it is apparent that its main function is to calculate the difference between the next s_a and the current s_a. If we define:

$$\Delta s_a = s_a(r_{BASE}(b, s, s_a)) - s_a$$

Then the Lagrangian can be redefined as $|\Delta s_a|$. Now, the set of all possible states of the PFEM must be redefined as a 4-dimensional scalar field, L_v (L because it represents an altered Lagrangian, and v because it is a **v**ector space). For the sake of standardization, we can define the x-axis to represent s_a, the y-axis to represent s, the z-axis to represent p, and finally the w-axis to represent b. Each point p in L_v represents a unique Δs_a, the Δs_{a_p} for that point. We can introduce another nearly identical field L_{vn} in which every point represents the Lagrangian, $|\Delta s_{a_p}|$. The only difference between the two fields is that at each point L_{vn} holds the absolute value of the corresponding point in L_v.

For the Lagrangian to decrease or stay the same on a given simulation step, an equation (the Lagrangian Descent Viability Equation, or LDVE) must be satisfied:

$$L_{vn}(L_v(s_a, s, p, b) + s_a, max(s, L_v(s_a, s, p, b)), p, b) \leq L_{vn}(s_a, s, p, b)$$

Looking back at the formal PFEM definition:

$$v * 0.99^s * 5^{floor(b/31536000)} = s_a^3$$

One viable proof utilizes s_a and works as follows.

In solving this proof, there are two ways of imagining its goal. The first is to make the Lagrangian decrease. The second is to make the two sides of the PFEM formula closer to equal. The two perspectives are in fact identical, as an increasing Lagrangian will cause the two sides of the PFEM formula to diverge. In order to make solving the proof easier, we will use the latter definition.

There are two possible cases when attempting to correct the PFEM equation. Either the left side of the equation is greater than the right or the left side of the equation is lesser than the right.

In the first case, where the left side, $PFEM_l$, is greater than the right, an increase in s_a would cause $PFEM_r$ to increase, as p is equal to its cube root. This would bring the $PFEM_l$ and the right side of the equation, $PFEM_r$, closer together, decreasing the Lagrangian. A increase in s_a has a clear incentive for all parties that choose to create more mining identities or activate existing ones: because $PFEM_l$ is proportional to r_{BASE}, the average reward per block per mining identity, $PFEM_l$ being greater than $PFEM_r$ signifies greater rewards for anyone who chooses to do so. In addition, a decrease in $PFEM_r$ also creates a stronger incentive, as r_{BASE} divides its reward calculation by s_a, the cube root of $PFEM_r$.

In the second case, where the left side, $PFEM_l$, is less than the right, a decrease in s_a is desired as it would decrease the Lagrangian in the same manner as the first case. Also, with a decrease in $PFEM_l$ and/or an increase in $PFEM_r$, rewards are decreased for all mining identities. This would lead many parties ceasing to mine blocks, causing a decrease in $PFEM_r$. Also, decreased supply would increase v due to scarcity, which would increase $PFEM_l$.

As both cases have incentives that align with the correct action in those scenarios, the proof holds; any PCSH economy will always eventually reach a v-state, as long as L_{ERR} is large enough to ensure the market has a buffering period to respond to any sudden price movements.

2.1.3 Proof of Stable Rewards

In a system such as Polycash, it is necessary to ensure that miners will always receive stable rewards for their contributions.

Again, keeping in mind the now-proven PFEM equation:

$$v * 0.99^s * 5^{floor(b/31536000)} = s_a^3$$

Now, defining:

$$r_t = 0.99^s * 5^{floor(b/31536000)}$$

The equation can be simplified for this calculation:

$$vr_t = s_a^3$$

Our goal is to calculate the change in rewards for a miner based on a change in vr_t. If vr_t was to decrease by another value, $vr\_$, than $PFEM_l$ and $PFEM_r$ would both decrease by $vr\_/v$. Note that $vr\_$ is a single variable and does not represent multiple variables multiplied together. Because s_a is equal to the cube root of $vr\_$:

$$\Delta s_a = \sqrt[3]{PFEM_r - vr\_/v} - \sqrt[3]{PFEM_r}$$

Because the difference between positive numbers is always greater than the difference between their cube roots, Δs_a is always smaller than $vr\_$, indicating that s_a is affected less than $vr\_$ when market fluctuations, epoch transitions, or changes to s occur. In the Polycash ecosystem, it is said that s_a is *cube-damped* from vr_t.

Because of this cube-damping effect, $r_{BASE}(b, s, s_a)$ (the average reward a mining identity will receive per block, in the base currency) is also effected:

$$r_{BASE}(b, s, s_a) = \frac{vr_t}{s_a}$$

Because, as shown by the equation above, s_a linearly affects r_{BASE}, r_{BASE} also becomes cube-damped from vr_t, meaning that changes in market conditions, block rewards, or block difficulty will not affect the profit of mining identities linearly; miner's flow of income is protected via cube-damping from the economy's state.

3 Time Verifiers

When difficulty is adjusted based on the time it takes for a miner to create blocks, it becomes necessary to ensure miners are honest about the time they start and finish mining. If these two timestamps can be verified, it would also prevent all manipulation of past blocks in the blockchain, even if 51% of the mining power in the network is controlled. To implement this security feature, time verifiers are used in the network. Any node that has mined a block may become a time verifier. When a node starts or finishes mining a block, they must request and collect the signatures of time verifiers. Time verifiers will provide their signature if the current time is within a 10-second range of the time they need to verify. There may be no less than 75% of the time verifiers in the previous block in the current block, and each additional signature beyond the

number of signatures in the previous block earns the miner a reward. Because of the reward gained from adding additional time verifiers, miners will attempt to gather signatures from as many nodes as possible. Therefore, the number of signatures will roughly match the number of verifiers in the network. As a given miner will never have control of over 50% of the verifiers in the network, they will fall short of the required number of signatures if they attempt to lie about the times they begin and end mining.

In addition to the number of signatures a miner needs to create a valid block, the majority of the verifiers the miner gathers signatures from must also be verifiers in the previous block. This almost completely prevents a miner from gaining malicious verification. Even if a miner controls 51% percent of time verifiers, these verifiers must not sign blocks- if they did, they would contribute to the number of verifiers in a block, and the number of signatures the miner could receive in their malicious block would still drop by about 50%, below the required 25%. This means a miner must control over 75% of miners in the network *and* 50% of the computing power. Without the second rule, miners controlling malicious verifiers could have those verifiers not sign blocks, so they would not contribute to the verifier count. Then, they could be used to verify the malicious block. However, none or very few of those verifiers will have verified the previous blocks, so with the second rule the malicious block will be invalid. This security measure, along with the next, creates an extremely secure blockchain with significant advantages over one using standard PoW.

4 Miner Count Limit

The miner count limit was removed in the Jinan upgrade.

In order to decrease the energy usage of the network and to slow the growth in the number of time verifiers, the number of miners in the network is limited to a maximum. If a miner with a public key not present as the miner in any previous block creates and broadcasts a new block and the number of unique public keys present as miners in the blockchain equals or exceeds the maximum, the block is rejected. After a miner has created one block, it can continue to create blocks even if the maximum number of miners have been reached; they are already registered. After they are registered, they do not have a strong motivation to use more energy to mine faster past a certain point, as the sigmoid difficulty function does not significantly reward faster miners after a point. This motivates miners to use a low to medium mining rate to optimize their profits; past a certain mining rate, energy costs outweigh mining reward benefits. In this way, the miner count limit decreases the energy usage of the network.

To calculate the maximum number of miners at the current time, divide the number of blocks in the blockchain by 20, rounding up.

5 Smart Contracts

To enable complex and customized functionality on the blockchain, Polycash implements a smart contract system that uses a blockchain-specific assembly

language. When a miner is sent a transaction, they evaluate all smart contracts that would be triggered in that transaction, and, if they initiate any new transactions, those transactions are added to the mempool if they are valid. The block is considered invalid if the contract-created transactions are missing or do not match the correct ones.

A smart contract may automatically make transactions on the behalf of another party, provided that party signs the smart contract when it is created, granting their approval of the instructions contained within it. This functionality is required to allow a contract to be deterministically executed without the need for trust of a party to fulfill their part of the agreement.

Each transaction may contain one or more smart contracts. Each smart contract may be either populated or empty. Populated smart contracts contain the entire contents of the smart contract, alongside its signatures and other metadata. It does not contain the location of the smart contract, which serves as an ID. When a populated contract is sent to the network, it instantiates an entirely new contract on the blockchain and runs it. On the other hand, empty smart contracts contain nothing but the location of an existing contract. When one is sent to the network, an existing contract on the blockchain with the matching location is executed.

5.1 Virtual Machine

Smart contracts running on the blockchain use the PVM (Polycash Virtual Machine) with a custom instruction set to deterministically initiate transactions. It uses a Polycash-specific instruction set and assembly language to evaluate these contracts.

5.1.1 Memory Management

The memory of the Polycash virtual machine is contained within a set of virtual buffers, expandable sections of data with no fixed length. Different data types may be represented by different lengths and organizations of buffers. There are an unlimited number of buffers, and each is an unlimited size, so the virtual machine may hold as much data as it needs in memory, so long as it doesn't run out of gas (see section 5.1.7)

The buffer at hex address 0x00000000, used for global errors is pre-initialized during the VM boot process.

5.1.2 Polycash VM Instruction Set *[extended]*

This section is located in a seperate document, instructions.md, located in the reference implementation repository.

5.1.3 Errors

Errors are split into two types: local errors and global errors. Local errors write an error value (hex code 0x01) into a pre-initialized error buffer if an error occurs in an instruction. Global errors write an error value (also hex code 0x01)

into the pre-initialized error buffer at 0x00000000. If the error buffer needed to throw a local error is not found, a global error is thrown.

5.1.4 Inter-Contract Communication

To enable communication between contracts, there is a constant, known as the External State Writeable Value (ESWV), which can be stored at a location in a contract's state. Then, any other contract may choose to write a different value to that location, despite their lack of write access to it in normal circumstances. Because the location will now contain a value other than the ESWV, it isn't writeable and hence "locked". In a similar way that a mutex prevents simultaneous writes to a location in memory, external writeable state will lock when it is written to and will unlock when the smart contract controlling the writeable state sets it back to the ESWV.

5.1.5 State cache

To enable changes to state within a single transaction to take effect before the block containing it is finalized, each smart contract will read/write from/to a cache instead of from/to the finalized state, which is updated every time a new block is added or removed from the blockchain. When a contract attempts to read from state, it will first check if the requested data is in the state cache. If it is, it will use the state cache to resolve its query. If not, it will query the finalized state, storing the result in the cache. In a similar fashion, when a contract attempts to write to state, it will write directly to the state cache. At the end of execution, the contents of the cache are written into finalized state in a flush operation.

5.1.6 Smart Contract Packing

Without smart contract packing, smart contracts must wait 1 or more blocks for messages to be sent to and from other contracts, as the VM's state updates once per block. In addition, smart contracts have to wait for each other if they wish to send data to the same smart contract at the same time, as state can only hold one piece of data at a time which can only be reset once per block for the same reason that smart contracts have to wait for 1 block until another contract can read data the first contract has sent. To resolve these issues, the Polycash blockchain uses a method called *Smart Contract Packing* which can 'pack' multiple smart contracts into one transaction.

Any smart contract may, using the `Invoke` instruction, run another smart contract (a 'child contract') that runs in a newly created stack frame and shares the parent contract's state cache. Any contract which uses the `Invoke` instruction in this way is known as a 'parent contract', and any smart contract called directly from a transaction, rather than from another smart contract using `Invoke`, is called a 'root contract', as it serves as the root node of a tree (the 'invocation tree') of smart contracts, all of which can be packed into a single transaction.

Because a child smart contract will share the state cache of their parent, and thus the state cache of the invocation tree's root contract, they can communicate

via the cache with any other contract in the tree without having to wait for read and write operations to the state to go through the finalized state, which, as mentioned previously, can only be updated once per block. This also allows for contracts that are frequently used by multiple other contracts (i.e. Uniswap) to handle many requests simultaneously, rather than having to wait for the finalized state's mutex mechanism to unlock, finalize, and lock each time a request is sent to it.

5.1.6 Gas Fees

The PVM's fee calculation uses a model similar to that of Ethereum, where each instruction costs a specific amount of 'gas'. When a smart contract begins execution, the PVM sets a gas limit for its execution equal to the total balance, measured in units of gas, of the contract's sender. During execution, the PVM keeps track of the amount of gas that has been spent so far. If at any point the amount of gas used exceeds the gas limit, the transaction that executed the contract is rejected as invalid.

6 Transaction Body

To allow data to be sent through the blockchain, a body may be attached to each transaction containing data of any form. A data fee is paid to the miner of that block of a size proportional to the amount of data transferred, in bytes.

This transaction body feature allows for the system to be used to create new tokens, implement two-factor authentication, or to permanently and reliably store data, while utilizing the security and reliability of the blockchain.

7 Hashrate Forking Prevention

It is crucial to prevent miners from forking their hashrate through multiple wallet addresses, circumventing the difficulty adjustment algorithm. Two major strategies are applied to this issue.

1. *Block Reward Locking*

   Block reward locking prevents miners gaining block rewards until they have mined at least 3 blocks, lowering the efficiency of mining rate forking.

2. *Block Reward Adjustment*

   Block Reward Adjustment decreases the block reward for each new miner that mines a block by 1% in order to impose a net loss of profits on miners that fork their hashrate. This is the most effective approach.

8 Fees

There are three scenarios in which taking actions on the blockchain requires the payment of a fixed or variable fee. Fees are deducted from the sender's balance in the same transaction as the action that required the fee.

8.1 Initiating Transactions

Once a transaction that has been created, signed, and sent to the network moves from the mempool to the blockchain, the sender of that transaction (the key that signed it) is incurred a constant fee of 0.0001 PCSH. This constant rate keeps fees low and predictable, as opposed to other blockchains that use an auction-based or traffic-based fee calculation methods, where increased transaction load increases fees dramatically.

8.2 Gas usage

Whenever smart contracts contained in a transaction consume gas, the sender of the transaction must pay 0.000001 for each unit of gas that was used during the contract's execution as part of the fee model for PVM resource usage (see section 5.1.6).

8.3 Transaction Body

When a transaction is appended in a new block to the end of the blockchain that contains data in its body (see section 6), the sender of the transaction is charged a fee equal to 0.000001 PCSH per byte of data present in the transaction's body.

Conclusion

We have seen that using an alternative consensus algorithm, APoW, with the help of the time verification protocol increases decentralization and decreases energy usage when compared to traditional PoW blockchains. It also avoids the security issues present in PoS or PoH blockchains, where controlling 1/3 of stake in the blockchain can cause the network to fail. We have analyzed how combining this new consensus mechanism with the Dilithium3 signature algorithm ensures its tolerance against quantum-based attacks. We have outlined a new blockchain, Polycash, that implements these features alongside a smart contract system that together secure and decentralize digital transactions and decentralized applications.