Adamnite: A scalable and secure blockchain development platform

Archie Chaudhury
archchaudhury02@gmail.com

Introduction

Since the advent of Bitcoin in 2009, cryptocurrencies and other digital assets have reshaped finance. Bitcoin was the first successful implementation of a decentralized currency supported by a peer to peer network, relying on no centralized authority or power to instill validity for the underlying asset. Recently, digital assets have gained immense attention due to their use-cases beyond just currency transfers, with Non-Fungible-Tokens (NFTs) and other use-cases becoming strong markets in their own right. Yet, despite the importance of digital assets, the invention of Bitcoin also led to the creation of blockchain technology as a tool for data-storage, validation, and consensus. Blockchain technology, along with its close counterpart Distributed Ledger Technology (DLT), has reshaped the Internet; Decentralized Autonomous Organizations (DAOs), InterPlanetary File Storage (IPFS), and Decentralized Applications (DApps) are all part of the new Internet commonly referred to as “Web3”. Accordingly, Adamnite seeks to produce a fully permissionless blockchain platform that will allow developers across the world to confidently create Web3 applications. Coming with a compiler that checks for logical validity and running on a DPOS consensus mechanism, Adamnite will be faster and more efficient than current blockchain solutions, thus accelerating the adoption of blockchain technology across both public and private sectors.

Background

Distributed Ledger Technology (DLT), the larger group of software that encompasses blockchain, has its roots in the early Roman Empire, which allowed its citizens to barter across the entirety of the empire using a record-keeping system. Any asset database that is shared across multiple nodes and does not rely on a central administrator can be defined as a DLT. DLTs, despite their potential, were never adopted *enmasse* in modern software due to the fear of malicious actors overtaking the system. This problem was summarized as “The Byzantine Generals Problem” by Leslie Lamport, Robert Shostak, and Marshall Pease in their 1982 paper of the same name. Simply put, the Byzantine Generals Problem uses the example of multiple generals in the Byzantine Army being unable to reach consensus on a plan, or worse, picking a bad plan, because of the presence of traitors within their decentralized communication platform. All honest generals must agree upon the correct plan, regardless of the presence of malicious actors in the system. Finding a solution to the Byzantine Generals Problem, or achieving

Byzantine Fault Tolerance, was a key hurdle in both the development and adoption of DLTs. Satoshi Nakamoto solved the Byzantine Generals Problem[1] through Bitcoin, which used a new algorithm that depended on participants solving cryptographic problems in order to reach consensus on a distributed ledger.

Bitcoin's initial success was a direct result of its unique integration of Proof-of-Work Consensus with the "Longest Chain Rule", which refers to the philosophy of assuming that the longest valid ledger is the most legitimate one. This implementation has come to be known as Nakamoto Consensus. In Bitcoin, Nakamoto Consensus specifically solved the Byzantine Generals Problem by incentivizing block validators, also known as miners, to validate the longest and most accurate chain. This ensured that all participants can have confidence in the validity of the current chain of transactions without having to look to a centralized authority. Bitcoin also solved the double-spending problem that had plagued previous digital currencies such as Digicash by introducing a timestamp record for each transaction; this ensured that no singular Bitcoin was "spent" multiple times by the same account. The idea of grouping these transactions into blocks, which were then validated through Nakamoto Consensus and stored on a public ledger, was revolutionary, and represented the first real use of what is now known as blockchain technology. The consensus mechanism also protects the network against malicious actors: an attacker will need to control over 50% of the computing power in the network in order to gain control of the blockchain (commonly known as a 51% attack). Bitcoin's success paved the way for blockchain technology and DLT in general. In the years following Bitcoin's release, multiple blockchains and their corresponding digital assets have been released, with each new chain focusing on a different method to improve upon Bitcoin's original model. For example, Ethereum proposed a blockchain with a built-in Turing Complete programming language to further the development of both smart contracts and DApps, while platforms such as Algorand and Cardano have leveraged different consensus mechanisms to create blockchains that are more scalable for enterprise use.

Applications beyond Decentralized Finance

Bitcoin's rise propelled the creation of new subsets of software and even the development of entirely unique sectors; for example, Decentralized Finance, or DeFi, rose entirely out of Bitcoin's successful implementation of a decentralized currency. Most importantly, the ability to make monetary transactions without a centralized authority or trusted third party was, and still is, revolutionary for the finance industry. However, blockchain technology has applications beyond just financial transactions and asset management. For example, a decentralized governance application for voting could easily be built using blockchain technology, with votes being recorded directly on the public ledger. Additionally, a storage service can also be built on the blockchain, enabling files and other forms of data to be encoded

[1] Although the Byzantine Generals Problem had been solved before, Bitcoin was one of the first scalable solutions that was also widely adopted.

within the public ledger. A more recent innovation are Non-Fungible-Tokens, or NFTs. NFTs are singular tokens representing a proof of ownership, and have seen numerous use cases in digital art, real-estate, and content management. For most developers, there are two main ways to create new software using blockchain technology. They can either create a new blockchain entirely, or leverage an existing one that fits their specific needs. Most DApps and token systems are built on top of existing chains, thus signifying the need for current blockchain solutions to be as scalable as possible.

Unfortunately, most alternative use cases are currently limited, at least in Bitcoin. This is largely due to Bitcoin's consensus model, which is perfect for a decentralized currency supported by a peer-to-peer network, but fails when used for higher-level applications. In particular, Bitcoin's utilization of POW means that individual transactions are costly, inefficient (on average, a transaction takes around 10 minutes), and environmentally unfriendly. This not only makes Bitcoin unsuitable for enterprise use, but also places restrictions on the use-cases for the underlying blockchain technology. The amount of computational power needed to process blocks of transactions makes Bitcoin somewhat restricted; governance and gaming applications may need to process 1000s of transactions per minute in a cheap and efficient manner, which is something Bitcoin cannot handle.

Bitcoin Script

Development on Bitcoin is primarily done through two means: direct additions to the Bitcoin Core software, and implementation of the current Bitcoin Protocol with the Bitcoin Script programming language. Here, we will focus on the Bitcoin Script programming language as it is a method for integrating the Bitcoin blockchain into a decentralized application or smart contract. Script is a stack-based programming language that allows for a transaction to have direct specifications with regards to how the receiver will be able to unlock the coins to be spent or transferred elsewhere. This is implemented through operations on Unspent Transaction Outputs (UTXOs) that essentially govern when and how a certain portion of the currency is made available. This allows for the creation of basic smart contracts that can be used to create applications that offer payouts based on certain requirements being met. These requirements can range from the completion of tasks that can be verified by the program to a simple exchange payout which dictates that you must send "x" amount of another asset in order to receive the corresponding amount of Bitcoin. Below, a code snippet of a Bitcoin Script program designed for a simple transaction to a public key address is defined.

```
scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
scriptSig: <sig> <pubKey>
```

Source: [Bitcoin Wiki]

The opcodes OP_DUP, OP_EQUALVERIFY, and OP_CHECKSIG define functions for duplicating the top stack item, checking that all the inputs are equal, and checking that the top item of the stack is valid, respectively. The Script programming language comes with multiple opcodes that can be used to create unique contracts depending on the preferred use-case.

While Script can support a plethora of applications that connect directly to the Bitcoin blockchain, it lacks several key features that prevent it from being a long-term scalable solution in blockchain development. Script only focuses on UTXOs, thus limiting applications to only transactions on the blockchain. This prevents developers from creating smart contracts that can take into account or mutate on-chain data. Script thus does not allow applications to interact with the overall state of the blockchain. Script is also decidedly complex; its syntax and structure make it difficult to implement for more difficult use cases. For example, an average script for a MultiSig Transaction will require all users to send custom scripts in order to have the transaction function as intended. This leads to inefficiency when creating scripts that handle more complex operations and transactions.

Ethereum Programming

The Ethereum Blockchain, described as “A Next-Generation Smart Contract and Decentralized Application Platform” in its initial white-paper, was meant to be a direct improvement over Bitcoin in terms of its scalability. Coming with a built-in Turing Complete programming language, the Ethereum Blockchain has become the main platform for DApp Development, with numerous developers and organizations using the platform to power their own blockchain base solutions. Programs written in Ethereum are significantly more scalable than their Bitcoin counterparts due to both their Turing Completeness and their capacity to process state; loops and on-chain data are frequently used to create programs that are both efficient and responsive.

Ethereum’s most popular programming language is Solidity, a high-level programming language based heavily on Javascript. Solidity is Turing-Complete, and any code written in Solidity is meant to be transitioned to lower level byte code and run on the Ethereum Virtual Machine (EVM). A typical Solidity program is built around contracts, which are essentially classes that define various structures within a larger program. A contract can be used to create and define operations for a new asset, governance mechanism, or identity verification tool. A simple smart contract designed to create a new asset, taken from Solidity’s Docs, is shown below:

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract Test_Coin{

   address public minter;
   mapping (address => uint) public balances;

   event Sent(address from, address to, uint amount);


   constructor() {
       minter = msg.sender;
   }


   function mint(address receiver, uint amount) public {
       require(msg.sender == minter);
       balances[receiver] += amount;
   }

   error InsufficientBalance(uint requested, uint available);


   function send(address receiver, uint amount) public {
       if (amount > balances[msg.sender])
           revert InsufficientBalance({
               requested: amount,
               available: balances[msg.sender]
           });

       balances[msg.sender] -= amount;
       balances[receiver] += amount;
       emit Sent(msg.sender, receiver, amount);
   }
}
```

Contracts support the use of functions, events, errors, etc. This allows developers to create complex smart contracts that respond directly to input. The contract above, for example, defines the event *Sent*, which signifies the transfer of the asset from one party to another.

Despite Ethereum's advantages, it still possesses several problems that act as a barrier to widespread institutional adoption. Namely, the high transaction fees in Ethereum, commonly referred to as Gas, mean that every smart contract must be carefully optimized to reduce deployment costs. Based on current network congestion, the deployment of a simple smart contract such as the one described above can cost upwards of 400 USD. Additionally, Ethereum programs, while significantly more high-level than their Bitcoin counterparts, come with their own nuances. Most notably, Ethereum code lacks a high degree of composability and reusability,

thus creating an ecosystem where the same code is often deployed over and over again. While libraries in Solidity have provided a temporary fix, they still remain to be widely used. Internal functions can be called recursively; however, there are clear limits based on both the size of the stack and the underlying memory. Ultimately, programming on the Ethereum blockchain on an enterprise-level scale is extremely difficult for the average programmer due to its underlying intricacies, with gas optimization and a lack of significant modularity being the key reasons. This causes developers to make rudimentary mistakes, leading to applications that have fatal security vulnerabilities as result of simple errors. These vulnerabilities result in significant financial loss every year as malicious actors take advantage of logical loopholes within the smart contracts driving these applications.

Alternative methods and chains

Currently, alternative blockchain platforms such as Algorand, Cardano, and Polkadot seek to make blockchain technology more scalable for both institutional adoption and DApp development. These platforms are often an improvement on existing solutions: Algorand and Cardano both reach on-chain consensus through a proof-of-stake implementation that drastically reduces transaction costs, while Polkadot offers interoperability and cross-chain transfers. However, these chains have their own unique problems, with centralization, a lack of scalability, and a lack of security being several of the most commonly cited issues. Current blockchain solutions often also focus on developing infrastructure for a specific area or use-case, with most chains placing a large emphasis on DeFi. This leads to platforms that are well-suited to dictate asset transfers, but less apt for other use-cases such as data storage.

Furthermore, as with Ethereum and Bitcoin, the learning curve for most alternative chains are extremely steep. While a skilled developer should be able to execute automatic transactions with relative ease, creating complicated smart contracts or applications that manipulate on-chain data is much more difficult. In order to create a full-fledged DApp, developers often have to contend with low-level languages with little documentation, ensure that their application is secure, and find structural support for extraneous use cases. While community-based development (the creation of multiple SDKs on Algorand is a good example) has made blockchain development more accessible, they are still not integrated into the blockchain's core smart contract architecture. Although current blockchain solutions succeed in facilitating decentralized currencies, they remain difficult to use for standard application development. This significantly limits the adoption of blockchain beyond finance, as most developers will prefer the more efficient and easier legacy solutions.

Adamnite

Adamnite is meant to represent the future of blockchain development, allowing both professional developers and hobbyists to leverage blockchain technology without having to go through an arduous learning process or sacrificing a large amount of computational resources. Furthermore, a significant focus will be put on enterprise level use: the Adamnite blockchain should be faster, safer, more powerful, and cheaper to implement than existing blockchain solutions. Adamnite accomplishes this by introducing a blockchain that comes with advanced security protocols, lower transaction fees, and a simplified programming ecosystem that allows developers to easily develop on chain-applications. By providing a system that is more intuitive and easier to understand, Adamnite hopes to create a platform that encourages more developers and organizations to embrace blockchain development.

Key Features

Easy to use: Adamnite's key appeal is its simplicity: its programming language and associated concepts should be as easy to learn as Python, C++, etc. Abstraction and modularity will be key; the Adamnite blockchain should emphasize ease of development even if it comes at the cost of efficiency or decentralization.

Scalability: Organizations and businesses should easily implement Adamnite into their current framework without worrying about security, transaction costs, or a high learning curve. A crucial part of Adamnite's ecosystem will be SDKs that allow developers to directly interface with the blockchain using popular programming languages such as the ones mentioned above.

Security: A significant feature of Adamnite's compiler will be an automatic verification engine that checks for both safety and validity. This automatic verification tool should help ensure that smart contracts created on the platform are as safe as possible, and avoid simple mistakes that could result in a significant loss of profits or exploit attacks by malicious players.

Adamnite Transactions

Adamnite, like any cryptocurrency, has native wallets and accounts that run on its blockchain. Adamnite does not differ between external accounts and contract accounts. Any account on the Adamnite blockchain should be able to operate as either an autonomous entity, or be manually controlled by a user. Adamnite's ecosystem is geared toward developers of all skill levels: experienced smart contract developers can directly interact with the system and make autonomous accounts that have code stored in them and interact directly with the Adamnite Virtual Machine, while other developers can choose to import their accounts into external applications created by others.

Transactions on the Adamnite network contain the same standard fields as any cryptocurrency platform:

1. Sender's Public Adamnite Address
2. Amount of NITE being sent
3. Public Address of the recipient
4. Message, an optional field where additional data can be stored.
5. Message_Size, an arbitrary integer describing how large the message is in bytes.
6. ATE_Max, the maximum transaction fee of the transaction
7. Sender's signature

Adamnite specifically allows developers to send messages of arbitrary size to power larger applications. In most blockchains, messages, note fields and other forms of arbitrary data storage are often non-existent or limited. In Adamnite, the message is allowed to be as large as possible, given that the sender pays a higher transaction fee. This allows for a broader range of applications to be built directly on the blockchain rather than turning to centralized services.

The net fee ATE_MAX[2] is determined similarly to Ethereum's net gas price: each transaction, whether made by an external account or smart contract, is analyzed to determine the total amount of processing power or storage imposed on the blockchain. Like Ethereum's model, this is to prevent malicious players from taking advantage of the network by sending repetitive transactions that consume a lot of computing power.

Consensus Mechanism

Adamnite's blockchain and consensus protocol leverages a variation of Delegated Proof of Stake (DPOS). DPOS is a consensus mechanism in which participants vote on a group of validators, or witnesses, to represent them in the network. These validators are then given the ability to both create and approve new blocks. Rewards occur twice: validators who successfully propose a new block of transactions are rewarded a portion of the transaction fees associated with that block, while active participants who regularly stake their tokens for the purpose of voting are rewarded an amount proportional to what they are staking. An important note is that in order to participate, a node simply needs to send a specific participation transaction (with no actual assets). This participation transaction will essentially communicate the individual node's preferences for the witness (a node is allowed to select more than one witness) for the next 5 blocks of transactions. This also means that witnesses can lose their position every 5 blocks if they are suspected of malicious activity.

[2] ATE is the internal currency for fees, similar to gas on Ethereum.

The Adamnite Blockchain also plans to use governance, similar to that of a Decentralized Autonomous Organization, to make protocol and structural changes to the blockchain. This is similar to the governance mechanism created by Algorand, where users vote on various proposals and have the opportunity to earn rewards in exchange. In Adamnite, governance will be used to select both delegators (individuals who oversee the blockchain's structure and govern the network) and vote on legitimate proposals coming from the delegators.

DPOS was specifically chosen as a consensus mechanism for its speed and decentralization; it allows the blockchain to process transactions quickly by delegating block proposal and validation to a small number of individuals, while still allowing for every user in the network to have an active say. For example, if there were *i* participants in the Adamnite Ecosystem, then there will only be *i/n* validators, where *n* is dependent on the congestion and complexity of the network at a particular point in time. Adamnite restricts the amount of nodes needed to propose and validate new blocks while incentivizing said nodes to act honestly, thus allowing it to be faster and more efficient than first generation Proof-of-Work or Proof-of-Stake chains.

Blockchain

The Adamnite Blockchain is similar to other proof-of-stake blockchains: the computational resources required for both block proposal and block validation are minimal, blocks are validated through shared consensus rather than the solving a computational problem, and all blocks are recorded on the blockchain upon validation. However, there are key differences. Adamnite blocks contain a list of all the individual storage messages for each transaction and the net storage size in bytes. Furthermore, a validation list (a copy of the public addresses of both the block proposer and validators, along with a cryptographic proof that they all were indeed in the set of validators chosen for that particular block) and the block number are also recorded. In order to validate a block, a witness *W* needs to first ensure that the previous block is valid, ensure that the current block has a reasonable timestamp t_n ($t_{n-1} < t_n < t_{n-1} + 10$, where *t* is recorded in minutes), ensure that each transaction is successful based on transaction parameters, ensure that the block number is valid, and finally ensure that the block proposer is a part of the pool of selected witnesses W_{total}. While there may be concerns over a malicious party specifically targeting the witnesses for a specific block based on both the public record of chosen witnesses and the inclusion of the validator list in each individual block, Adamnite's unique DPOS scheme should prevent this. Specifically, a cryptographic sortition scheme, similar to ones employed by Algorand and Witnet, is leveraged to randomly select the pool of witnesses W_p from the top *n* addresses as determined by vote, where *n* is inherently dependent on the total number of participants in the ecosystem. This process is repeated for every block in a round, which consists of 5 blocks. The cryptographic sortion scheme ensures that randomness is a key element of choosing the validators for every block, thus making it extremely difficult for attackers to target specific validators.

$$h(sig(t, rand(i), M)) < K$$

In the above equation, *h* represents a Verifiable Random Function (VRF), the signature function takes in time, a random value, and the key, and *K* represents the reputation of the individual address, which simply is the total number of votes the particular address received during the previous “election”. For each block, the probability that an address within *n* will be chosen to be within W_p is inherently dependent on the amount of votes that were allocated to it.

Programming on Adamnite

All transactions and autonomous contracts on Adamnite are compiled using ADM code, a low-level stack bytecode language that directly interfaces with the Adamnite Virtual Machine (AVDM). AVDM is a computational engine that is shared among all Adamnite nodes, and allows developers to write code off-chain. ADM code is Turing-Complete: it is meant to allow developers to have as much flexibility as possible when creating applications. Developers will have access to opcodes that allow them to perform loops or create sub-programs. This is extremely similar to the Ethereum Virtual Machine (EVM), although there are several key differences. The main difference is the way in which smart contracts are processed. In Ethereum, smart contracts are limited due to the presence of gas fees, which increase in proportion to the computational complexity of the code. In Adamnite, deploying a smart contract essentially costs the same as sending a regular transaction, with the only additional variable being the total amount of storage a particular contract may take up. In fact, smart contracts themselves are defined through a specific type of transaction that encodes a new set of instructions to a particular account, similar to how Bitcoin Script encodes a set of instructions of a particular wallet. These fees are consistent across all smart contacts, only changing based on the amount of data the smart contract needs to store. This allows developers to write complex applications that leverage blockchain technology without worrying about progressively higher fees. Additionally, most of the smart contract’s parameters (excluding storage and state) can be manipulated after deployment through a separate transaction; this enables developers to easily update their smart contracts to both correct security vulnerabilities and implement new software.[3]

On-chain code will also be easy to both write and deploy: each primary client will have an extensive collection of functional libraries, thus allowing developers who may not be familiar with smart contract development to leverage modularity in order to create DApps. For example, an app developer who wants to interact with the Adamnite Blockchain will be able to do so without possessing an extensive low-level knowledge of the underlying protocol by simply importing existing libraries that handle various functions. This is not just limited to fetching data or sending transactions; a reasonably skilled developer should be able to manipulate the state and create automated transactions/smart contracts in as little as 5 lines of additional code.

[3] This model is also currently implemented in Algorand, which uses the transactions model for its smart contracts (defined as application transactions).

This will be an iterative process; as more applications, libraries, and packages are built on Adamnite, the more easier it will be for new developers to get started. In that sense, Adamnite is a programming software not unlike Python or C++; its goal is to enable a shared community of developers to create functional programs that use blockchain technology.

Compiler

Compilation is where a specific part of Adamnite's proposed ecosystem comes into play: every one of Adamnite's programming languages will have a compiler with a verification engine that checks for logical invalidity and specific knowledge assertions defined by the developer. An example follows. A developer wants to ensure that a smart contract will not allow a participant Alice to arbitrarily withdraw funds through a loop. They will simply need to declare this through an assertion in a separate script used specifically for compilation. This assertion can simply be a check of a single variable, or something more complex such as checking the intermitim value of a function's output. Whatever the case, the Adamnite compiler will be able to check the assertion against the body of code for which it is defined and see if it holds. The compiler will also check for loop invariants, which are essentially conditions that are true at the start of every *i*th iteration of a loop. The compiler's security will be mostly centered around property-based testing, thus allowing for developers to check individual parts of their code to ensure that it functions as intended before deployment. This is similar to the verification engine provided by Reach, a blockchain development platform that allows developers to write smart contracts for multiple platforms at once. However, for Adamnite, the verification engine is built into compilers for every high-level language into the ecosystem, and does not require developers to learn a new language that they may be unfamiliar with. Furthermore, the compiler also supports the use of direct tests; developers should be able to write exploit scripts to test their own programs. Much like how QuickCheck enabled programs written in Haskell to be a lot more secure, Adamnite's verification software will enable programs written for the Adamnite to be a lot more secure. By making verification a key part of every program, Adamnite's compiler will allow for a streamlined and secure development process which will hopefully help create DApps that are less prone to exploits.

Potential Use-Cases

Asset Creation

Assets built on top of existing blockchains have grown rapidly in popularity, and now make up a significant portion of the cryptocurrency market. These assets can represent equity stake in a physical company, have some form of unique utility, or be used for governance in a DAO. Sub-Tokens, as they are often called, often serve to support or tokenize a different use case. On Adamnite, a new asset can easily be built directly onto the blockchain. The creation of an Adamnite-Sub-Token simply involves encoding standard token parameters such as the name, amount, and the address of the creator. Transaction functions are built into every high-level language on Adamnite, allowing developers to create assets with pre-defined parameters. An example of what Adamnite hopes to achieve with asset creation is given below in a high-level pythonic based language:

```
def test_token(minter):
      Self.amount = 100000
      Self.name = "Test Token"
      Self.decimal_values = 0
      Self.symbol = "TT"
      Self.creator = minter
      Self.Inital_sender = True
```

Additional functions that define minting more of the asset, specific rewards, transactions or fee calculations can be added by the developer. All high-level languages on Adamnite will support a plethora of built-in functions that will allow developers to both create assets easily and introduce a certain degree of uniqueness to their smart contracts. The flexibility of the Adamnite Programming Stack will enable developers to take either a hands-off approach or delve deeper into the source to modify the built-in contracts at a deeper level.

Decentralized Finance

Adamnite's accounts could easily be leveraged to create smart contracts that allow for autonomous transactions to be processed directly on-chain. These smart contracts can be used to create DeFi applications that only send transactions based on certain requirements being met. For example, this could be used to create an exchange, where the transfer of an on-chain asset to an autonomous account results in that account paying out the equivalent value in a separate asset. Due to Adamnite's storage capabilities, a wide variety of data can be used for analysis in smart contracts, thus allowing for the creation of more diverse DeFi Applications. An example will be a banking application that lends assets as collateral, and establishes a credit history for a particular account based on its past transactions.

Streaming and Video Encoding

Decentralized Streaming has a variety of applications, from setting up communication hubs directly on the blockchain to providing centralized services in a decentralized fashion. There have been implementations of decentralized streaming, such as LivePeer, that have seen success in recent years. On Adamnite, decentralized streaming applications can be easily built due to the blockchain's storage capabilities. Footage will be able to be stored in the message field of an Adamnite transaction, and in turn an external smart contract will be able to send these transactions between different parties. Adamnite's speed will also help these applications; streaming DApps that leverage Adamnite should have the same efficiency as their Web2 counterparts. Developers can create a similar model to existing solutions, with miners being rewarded to help transcode videos to put on the blockchain. Because of the ability to directly encode large amounts of data within transactions sent on the blockchain, users will be able to directly put video data on the blockchain, with transaction fees being the only source of concern.

Decentralized Autonomous Organizations

A DAO, in its most simplest form, is a group of people who come together to make decisions regarding the rules or structure of an organization. Decisions are often made through votes through a standard governance process, with a computer program automatically making changes based on the outcome. Rules and decisions could extend beyond code; many DAOs make decisions on the allocation of internal capital, investments, and more. The key to implementing a basic DAO structure is to have mutable code that depends on the consensus reached by the members of an organization, and a basic governance model for determining said consensus. Governance models have also gained popularity in the blockchain community, with decentralized organizations and some blockchain networks using governance to enable the

wider community to reach consensus on a particular proposal. A simple DAO or governance model can be implemented on Adamnite as follows:

- Create separate blocks of code that only activate based on certain logical parameters. For example, there could be a block of code that is meant to transfer *X* amount of an asset from one account to another, and only executes when a satisfactory number of voters choose to approve the transaction.
- Create smart contracts that define a new proposal and allow constituents to vote on whether or not it should be implemented. The actual creation of the smart contract is flexible, and can be done in multiple ways. A simple method is to simply create autonomous Adamnite Addresses representing each decision, and have these addresses register votes either through the transfer of an asset or the manipulation of some other on-chain data.
- Implement the correct block of code based on the decision reached by the constituents.

Adamnite can also be used for governance methods by any community, thus allowing any group of people to make decisions directly on the blockchain.

Miscellaneous

Why DPOS?

DPOS as a consensus mechanism is perceived to have two common flaws: security and centralization. These issues are hand-in-hand; critics of DPOS point that there must be active engagement in the voting process in order to prevent the same accounts from being selected repeatedly as witnesses, which therefore leads to centralization and security concerns, as a malicious attacker will be able to pinpoint the witnesses. Adamnite solves this problem in two ways. First, by introducing incentives for voting, Adamnite ensures that participants will be more likely to participate in the voting process. Second, by leveraging cryptographic sorition, Adamnite ensures that the pool of witnesses for a particular block is semi-random, with the initial pool of the highest vote earners being the only public information.

DPOS was specifically chosen as a consensus mechanism because of its speed, security, and reliability. DPOS has been shown to be faster and more efficient than both POW and POS, primarily because it utilizes a small number of witnesses to validate new blocks, rather than requiring a proof of computational power or consensus among a large number of addresses. Furthermore, DPOS is decentralized, as every token holder has the opportunity to participate in voting. Finally, witnesses have a large incentive to act honestly, as they could lose both their position (and their potential for earning validation rewards) at any time. Ultimately, Adamnite's DPOS consensus mechanism will help make it more scalable, as developers will be able to create applications that are both speedy and secure.

Governance

Adamnite plans to use governance as a way for participants to both vote on delegates and on proposals. Initially, governance in Adamnite will follow a traditional coin-vesting scheme, similar to the governance mechanisms utilized by Cardano and Algorand. Participants will be asked to lock a certain amount of NITE in order to vote on a proposal or on delegates. They will then be able to vote for a certain option. The option with the highest amount of NITE dedicated to it will be the one that is implemented. In the case of delegates, the top *m* addresses will be chosen, with *m* again depending on the total number of participants. This is ultimately a coin-voting mechanism, as an individual participant will be able to lock a larger amount of NITE to have a larger say. Although coin-voting does cause a certain degree of centralization, it is currently the most thought-out and widely implemented solution. In the future, Adamnite will likely transition to a different governance mechanism such as proof-of-participation in order to ensure both decentralization and security.

Governance proposals could range from a simple adjustment in the rewards structure to funding for on-chain development. Delegates will ultimately be in charge of approving and selecting new proposals, although anyone may create a new proposal for review. Proposals will be approved in a similar fashion to Bitcoin Improvement Proposals (BIPs), with the delegates actively engaging with both the individual who created the proposal and the wider community. The creator will then have the opportunity to revise their proposal before it is reviewed for approval. This creates an open process, which helps make the Adamnite network more decentralized.

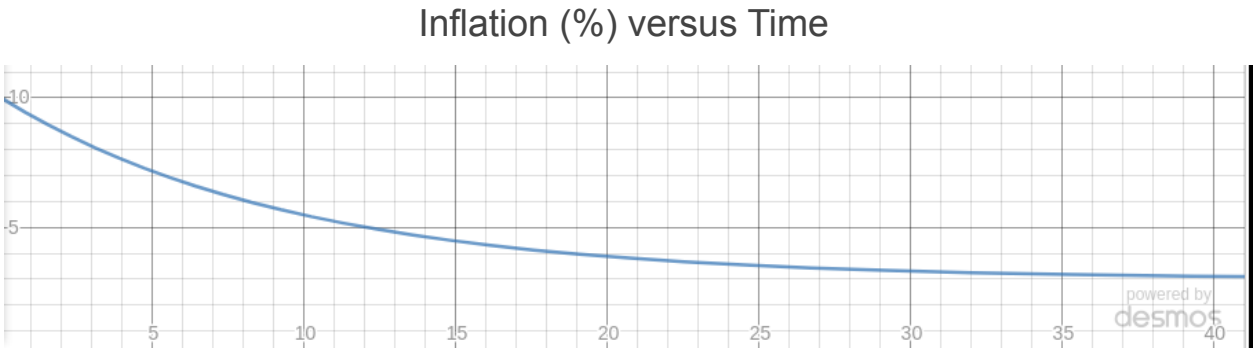
Currency and Tokenomics

The Adamnite blockchain will have its own built-in token, called nite. This will be used to process on-chain transaction fees, voting rewards, etc. There will also be smaller denominations of nite, analogous to cents in the USD Currency System and Satoshis in Bitcoin. As in Ethereum's initial white paper, these denominations are named after some of the most prominent contributors in cryptography and blockchain. They are defined as follows:

- 1: micali
- 10^{10}: sunny
- 10^{12}: vitalik
- 10^{14}: nite

Nite will have a permanently growing supply, with a mathematical function controlling the growth rate. The main argument for having a consistent growth rate, as opposed to fixed supply such as Bitcoin, is simply decentralization. Assets with fixed supplies are often concentrated in the hands of early adopters, thus preventing new individuals from being able to participate in the ecosystem. A growing supply ensures that the network is always ready to support new users. Token issuance will come from an account controlled by a delegated party, which could be a group of chosen delegates or an official organization with the sole purpose of growing the Adamnite network.

The growth function is also meant to control inflation. Currently, the proposed inflation rate is 10% at the minting of the genesis block, followed by a reduction to a long term inflation rate of 3%. These parameters, along with the annual reduction, can be altered by the broader community in governance proposals. Furthermore, the inflation rate is a maximum rather than an average or goal: the growth function does not account for tokens lost to burning, misplaced private keys, etc. Like the issuance models proposed by Etherem and Solana for their respective blockchains, the growth rate eventually reaches the constant proposed above:



Conclusion

Adamnite represents the future of blockchain development. By providing developers with the means to create fast, efficient, and secure applications, Adamnite sets the stage for a world in which blockchain technology is more widely used and preferred to legacy Web2 solutions. Adamnite also serves to push peer to peer computing forward; Adamnite's storage capabilities and consensus system mean that individual nodes/witnesses are essentially rewarded for hosting on-chain data encoded by other developers using the platform. However, the most

unique feature of Adamnite will be its development ecosystem. Developers, regardless of their prior experience with DLT or blockchain technologies, will be able to confidently build applications on Adamnite as a result of its modularity and security. This will significantly reduce the barriers needed to start leveraging blockchain technology in day to day development.

The idea of a next-generation blockchain platform is not entirely new: protocols such as Ethereum have already made a significant contribution to blockchain adoption and innovation. Adamnite aspires to accelerate this trend by providing a platform that is at once more efficient, safer and easier to use than current solutions. Adamnite will be at the forefront of a world in which blockchain technology is used to its full potential, allowing anyone, anywhere to build powerful applications with the power of decentralization and distributed computing.