Immersera - Layer 2 Network Based on BSC

ImmerseLabs

Brief description

Immersera is a Layer2 blockchain protocol based on Binance Smart Chain.Immersera offers higher throughput, faster finality and more efficient dispute resolution than previous Rollups.Immersera implements these features through several design principles: separating the order of transactions from deterministic execution; separating the order of transactions from deterministic execution; Combining the existing BSC Layer1 network with extensions to enable cross-chain functionality; compiling separately for execution and proofs, so execution is fast and proofs are structured and machine-independent; and using the Supertrust protocol to settle transaction results to the underlying BSC chain.

Immersera is focused on building web3, Metaverse and GameFi infrastructure, providing developers and users with unlimited possibilities of block GameFi experience. ImmerseEra has optimized the EVM protocol layer for the product features of Web3.0, Metaverse and GameFi industry, BRC-1233, BRC-1244, BRC-1255 protocols have revolutionized the traditional game developers to GameFi, BRC-1233, BRC-1244, and BRC-1255 protocols solve the biggest problem for traditional game developers to switch to GameFi, bringing trillions of traditional games to the blockchain and revolutionizing the Web3.Metaverse industry.

1.Immersera introduction

In previous work, we described Immersera, a system and protocol that extends the performance and scalability of smart contracts. This paper describes Immersera, a significantly improved design that provides advantages over the original, including higher efficiency, reduced latency, stronger activity guarantees, and better incentive compatibility.

1.1. The nature of Immersera

Immersera technology supports the execution of smart contracts. The system is implemented as "Layer2" on top of the BSC, although in principle it can be implemented on any blockchain system that supports at least basic smart contract functionality.

Immersera offers a BSC-compatible chain of:
Smart contract applications deployed in BSC Virtual Machine (EVM) code, and Immersera nodes support the same API as public BSC nodes.

The Immersera protocol guarantees the security and progress of the Layer2 chain, assuming that the following BSC chain is secure and that at least one of the participants in the Immersera protocol behaves honestly. The protocol is called "SuperTrust" because the execution is more efficient when the parties act in accordance with their incentives. Provides a lower cost in exchange for an additional assumption of trust

1.2 Design Method

Immersera's design has four unique features that we will use to organize the presentation.
Deterministic execution after sequencing: Immersera processes committed transactions in two stages. First, it places the transactions into the sequence in which they will be processed and commits them to that sequence. Second, it applies a deterministic state transition function to each transaction in order.

Geth: Immersera's core execution and state maintenance functions are handled by code from the open source BSC ("geth") package, the most popular BSC execution layer node software. By compiling in this geth code as a library, Immersera ensures that its execution and state are highly compatible with BSC.

Separate execution and proof: Immersera compiles the code for its state transition functions for both targets. The code is compiled for local execution when used in common operations in the Immersera node. The same code is compiled into a portable web assembly ("wasm") for the fraud proof code, if a

protocol is required. This dual target approach ensures fast execution while the proof is based on structured, machine-independent code.

SuperTrust with Interactive Proof of Fraud: Built on the original design, Immersera uses an improved SuperTrust protocol based on an optimized anatomy-based interactive proof-of-fraud protocol

3. Immersera Key Benefits

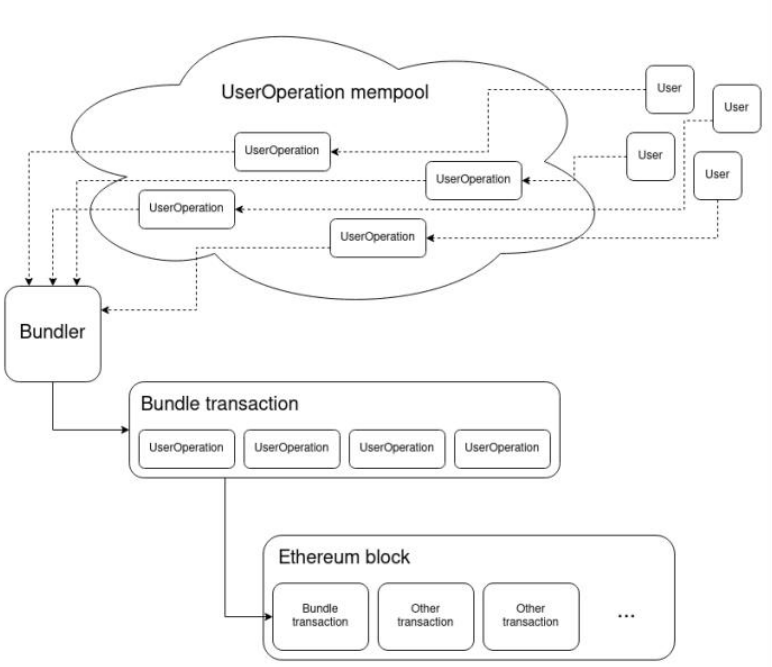
1 Smart contract, integration of BRC-4337 protocol, development of BRC-1233, BRC-1244, BRC-1255 protocol

1.1 Integration of BRC-4337 protocol, abstract accounts

1.1.1 Social account login, players do not need to remember the helper word or private key, through the social relationship to restore the account, the game integrated BRC-4337 protocol, players can use common email, cell phone login

Social account login solves 3 problems of traditional private key wallets.

1.1.1.1 The threshold is too high, users need to have some basic knowledge about public key and private address before they can use the wallet; 1.1.1.2 Assets are easily lost, users need to save their own private key, once the private key is leaked, the assets will never be recovered; 2.1.1.3 The efficiency is low, every transaction on the chain requires signature authorization and other steps to complete the transaction confirmation. For example, if you play web3 game, if all the games are on the chain, then you need to sign and confirm every action in the game, which will affect the game experience.



Social account login process

1.1.2 Gas fee substitution, players do not need to buy BNB for transferring Gas fee when operating on the chain, Gas fee can be paid by the game publisher or use BRC20 tokens as Gas fee, reducing the learning cost of players

The introduction of this feature reduces the cost of use for players, making it more convenient and user-friendly, especially for players new to GameFi or those who do not hold a BNB. Players can focus on the game itself without worrying about the purchase and management of Gas fees.

1.1.3 Batch packaged transactions, support multiple packaged one send, similar to the transactions on Dex, new tokens need to approve authorization before swap (swap), while account abstraction can be done in one step.

1.1.4 Automatic transaction (similar to Alipay's secret-free payment), when the user sets the transaction conditions, the transaction will run automatically when the relevant conditions are met. In the game, after the requirements are met, the signature is automatically authorized to improve the game experience, and the high frequency transaction on the chain can improve the usage experience.

1.1.5 User Authentication and Authorization: ImmerseEra uses blockchain user authentication and authorization mechanisms to enable players to be authenticated by wallet address and to control

permissions and access in a smart contract. This protects players' personal data and game assets, and ensures a fair and secure gaming environment.

1.2 BRC-1233

1.2.1 Support NFT game props to be chained with one click, using BRC-1233 protocol standard, hundreds of game NFT props in traditional games can be issued on the chain in a very simple way, traditional game developers do not even need to build smart contract development environment, no need to learn solidity development language, which greatly reduces the migration cost of traditional games to GameFi

1.2.2 Support one key to convert traditional game currency such as game gold and points into blockchain tokens. After conversion into blockchain tokens, the game can develop many advanced gameplay methods according to the technical characteristics of blockchain, and transform players into shareholders and promoters, so that the income of the game is associated with individuals, which will greatly reduce the promotion cost of the game and is a big trend in the future.

In the future, the innovation of GameFi industry will surely make people feel the new production relationship of blockchain distribution for the first time, and the traditional corporate system of distribution will become history.

1.3 BRC-1244

It supports one-key listing transaction of game props, using BRC-1244 protocol, game players can conveniently put the game tools and characters on the NFT trading market with one key, so that game developers do not need to develop their own game stores, reducing the development cost of the game and players gain more autonomy, and the model of game trading platforms such as 5173 will be advanced in the blockchain field.

1.4 BRC-1255

The BRC-1255 protocol provides protocol-level support for prop synthesis, upgrades, and inscriptions in games, eliminating the need for developers to build complex contracts at the contract development level, and requiring only a few hours to build a BRC-1255-compliant synthesizable, upgradable NFT. Without the BRC-1255 protocol, it would take weeks to complete the same work

1.5 Using these BRC protocols provides strong standards-based support for traditional games to move to GameFi

1.5.1 Virtual Character Creation:ImmerseEra allows players to create virtual characters and assign each character a unique

Abstract accounts. These abstract accounts can be associated with a real user's account, but can also exist independently of the real user.

1.5.2 Character Attribute Management: ImmerseEra provides the function of character attribute management, which allows developers to define and manage various attributes of game characters, such as level, skills, and equipment. These attributes can be stored on the blockchain and bound to the character's abstract account to achieve security and tamper-evident attributes.

1.5.3 Character trading and auctioning: ImmerseEra supports character trading and auctioning through abstract accounts. Players can freely trade characters on the chain and use smart contracts to make transactions secure and trustworthy. ImmerseEra also provides an auction function that allows players to sell and buy characters through a bidding process.

1.5.4 Cross-Game Character Circulation:ImmerseEra enables cross-game character circulation through the BRC-4337 protocol. Players can transfer their characters from one game to another without being limited by the closed nature of a single game. This provides players with more choice and flexibility, and facilitates interaction and cooperation between games.

1.5.5 Highly customizable character features: ImmerseEra's abstract account feature allows developers to customize character features and behaviors based on game requirements. Developers can write custom logic via smart contracts and integrate it with the character's abstract account to achieve a rich variety of gameplay and functionality.

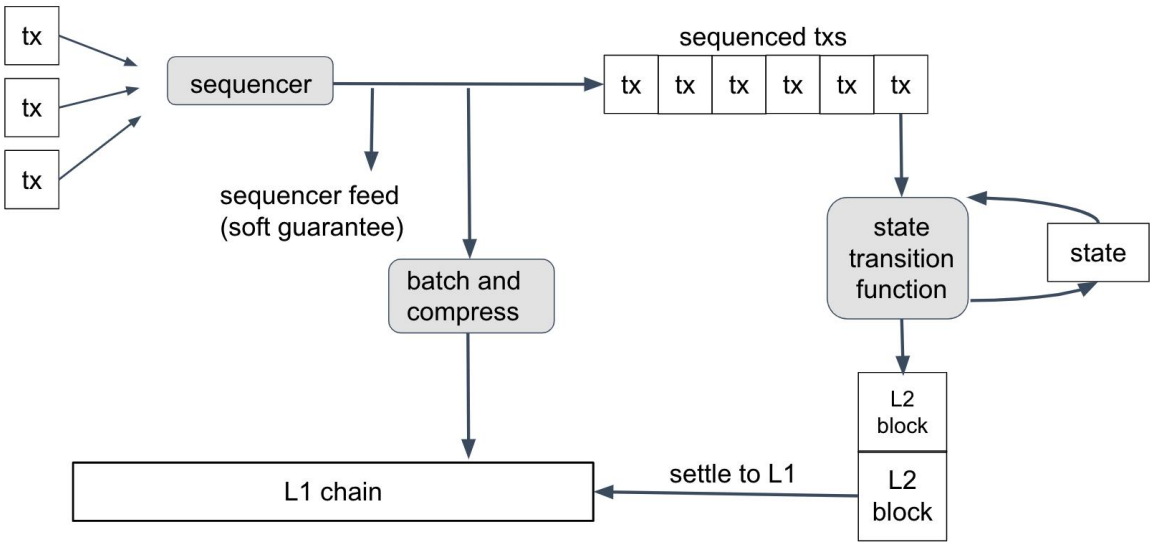
Using ImmerseEra's BRC-1233, BRC-1244, and BRC1255 protocols developed specifically for the GameFi industry, Game to GameFi conversion only takes 15~30 days and 2~3 developers to capture the future trillion dollar GameFi market.

Metaverse Ecological Support

NFT game contracts developed using BRC-1233, BRC-1244, BRC-1255 protocols can be seamlessly shelved and traded in ImmerseEra's on-chain NFT trading platform, for some of the same properties, bulk issue of NFT, without pending orders, can be traded directly in the NFT pool, trading NFT as simple as trading tokens, compared to Sudoswap's NFT pool trading, the function is more wanting, the transaction is more silky. Sudoswap's NFT pool trading, the function is more want to play, the transaction is more silky smooth, players do not even need to leave the game, you can complete the transaction with one click senseless

2 Certainty after sequencing

The processing of a transaction committed in Immersera is divided into two stages. First, a component called the sequencer puts the transaction into the sort and commits it to the sort. Second, the transactions are consumed sequentially by the deterministic state transition function. This process is shown in



Submitted transactions may or may not be valid. For example, they may lack a valid signature or they may be garbage data. An honest sequencer will do its best to discard committed invalid transactions, but the protocol makes no assumptions about the validity of the transactions in the sequencer output. The state transition function on the invalid transaction processing being performed will simply discard the transaction.

2.1 Sequencer

The sequencer can only honestly order incoming transactions based on a first-come, first-served policy.1 Currently the sequencer is a centralized component operated by an off-chain lab, but in the future we intend to transition to a committee-based sequencer using a fair distributed sequencing protocol

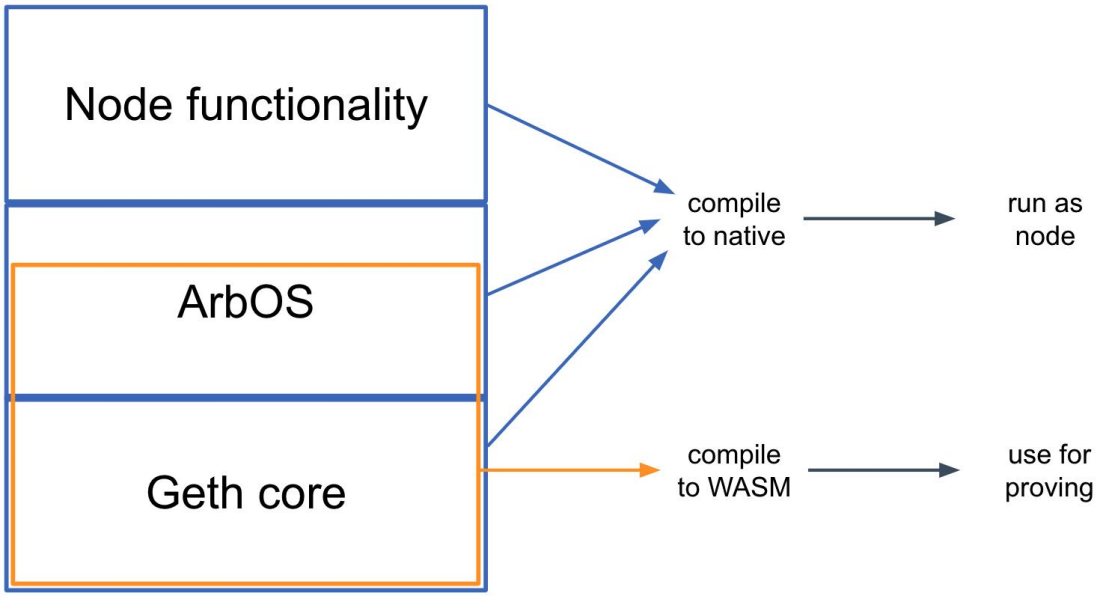
In principle, the sequencer can implement any transaction sorting strategy. A first-come, first-served strategy is easy to implement and minimizes latency.

The sequencer does not have the power to prevent the chain from progressing, nor does it have the power to prevent the inclusion of any particular transaction.

The sequencer publishes the order of its transactions in two ways. First, it publishes a live synopsis of the sequenced transactions, to which any party can subscribe. This synopsis represents the sequencer's commitment to eventually record the transactions in a particular order. The sequencer has the power to fulfill its commitments, so any deviation from the committed sequence could be due to a malfunction or malicious intent of the sequencer, or due to a deep reorganization of the layer 1 chain.

Next, the sequencer publishes its sequence of transactions as BSC invocation data. The sequencer collects a batch of consecutive transactions, compresses it using a generic compression algorithm (currently brotli [1), and passes the result to the Immersera chain's inbox contract, which runs on the L1BSC. These batches represent the final and authoritative transaction ordering, so that the sequence of transactions of the Immersera chain is final once the transactions of the sequencer to the inbox have finality on the BSC.

Deferred Inbox While most user transactions will be submitted directly to the sequencer and included in a batch in the sequencer, there is another way to submit transactions via the Deferred Inbox. This serves two purposes. First, it allows transactions to be submitted via L1BSC contracts, which cannot generate the digital signatures required to submit transactions via the sequencer. Second, it provides a backup method for anyone to submit a transaction in case the sequencer starts reviewing valid transactions.



Transactions are added to the delayed inbox by calling a method on the Inbox contract of theImmersera chain. These contracts keep a queue of transactions with timestamps. The sequencer can include the first message in the delayed inbox queue in its order. An honest sequencer will do so after a short delay, which is enough to ensure that messages arriving in the delayed inbox do not disappear due to reorganization of the L1 chain - typically a 10-minute delay.

However, if a message is in the delayed inbox for at least a threshold time period (currently 24 hours), anyone can force the message to be included in the chain's inbox, thus ensuring its execution. This forced inclusion step prevents sequencer censorship, but is only needed because the sequencer is malicious or has a long downtime.

Processing of transactions in Immersera. The sequencer creates orders for transactions and publishes the orders as live feeds and batches of compressed data on the L1 chain. The ordered transactions

are processed once through a deterministic state transition function, which updates the chain state and produces L2 blocks. These blocks are later fixed on the L1 chain.

2.2 Deterministic execution

After sorting the incoming transactions, they are processed by using the chain's state transition function (STF). the STF takes a state (i.e., the root hash of the BSC state tree [14), and an incoming message, typically a transaction. the output of the STF is an updated state and a new BSC-compatible block header that will be appended to the Immersera chain.

STF is completely deterministic, so the result of executing STF on a transaction depends only on the data of the transaction and the state before the transaction. Because of this, the outcome of a transaction depends only on the occurrence state of the Immersera chain, the sequence of transactions before T, and T itself. Due to this determinism, an honest party can determine the complete state and history of the chain, given only the sequence of transactions, or given the confirmed state of the chain at some point in the past and the sequence of transactions thereafter. The nodes do not need to communicate, nor do they need to agree among themselves in order to agree on the correct state and history, since this depends only on the sequence of transactions that are visible to all.

Immersera does have a Rollups subprotocol (discussed in Section 5) that confirms that the L1BSC chain confirms the outcome of a transaction. This subprotocol does not determine the outcome of the transaction, but merely confirms and records the outcome that is already known to the honest part of the protocol -

3 Software Architecture: Geth at the Core

The second key design concept of Immersera technology is "geth at the core". The "geth" here refers to the BSC, the most common execution layer of the BSC node software for almost all of Immersera.

The software that makes up an Immersera node can be thought of as being built in three main layers, the bottom layer being the core of geth - the part of geth that emulates the EVM contract execution

and maintains the data structures that make up the BSC state. Immersera is compiled in this code as a library with only a few minor modifications to add the necessary hooks. The middle layer, which we call NovaBridge, is custom software that provides additional functionality related to layer 2 functionality, such as decompressing and parsing the sorters' data batches, calculating layer 1 gas costs and charging fees for reimbursement, and supporting cross-chain bridge functionality, such as from L1 and withdrawals to L1

2: High-level structure of the Immersera code, showing the main components. The boundaries of the state transfer functions are shown in orange.

The top layer consists of node software, mostly from geth. it handles connections and incoming RPC requests from clients and provides other top-level functionality needed to operate BSC-compatible blockchain nodes.

Because the top and bottom layers rely heavily on geth code, this structure is known as the "geth sandwich" state transition function consisting of a geth layer at the bottom and a part of the NovaBridge layer in the middle. In particular, the STF is a specified function in the source code and implicitly contains all the code for that function call. Executing STF may modify the state and at the end issue a new block header (in BSC's block header format) that will be appended to the Immersera chain.

3.1 NovaBridge

NovaBridge is a software layer that implements the functions necessary and convenient for managing Layer 2 chains. This includes bookkeeping functions, cross-chain communication, and L2-specific expense tracking and collection.

3.1.1 State representation

All states of the Layer 2 Immersera chain are stored in the Merck-Patricia state data structure of the BSC. This includes the state of NovaBridge, which is modified as part of the state transition function.

NovaBridge encodes its state in the storage slot of a special BSC account whose private key is unknown. Select the specific slot used to meet the following objectivesKeeping all NovaBridge state in the storage of a BSC account allows NovaBridge's subcomponents to manage their state separately without collisions, keeping reasonable locations within the same subcomponents, and avoiding restrictions on future additions to state.

The state is organized as a hierarchy of nested "spaces", each of which is a mapping from a 256-bit index to a 256-bit value, all of which are implicitly initialized to zero. This structure is mapped to a flat 256-bit to 256-bit key value store, which is the store of the special BSC contract. Each space is associated with a key.

The key of the root space is zero, and the key of a subspace named n in a space with key k is H(k||n), where H is the standard Keccak256 hash function of BSC. This scheme ensures that the keys of spaces do not collide. In the space with key k, items with index i are stored in the underlying planar store at position H \ (k||i), where H is a hash function that preserves locality.

The function H\(x) hashes all but the last 8 bits of x, truncates the result to 248 bits, and then appends the last 8 bits of x. This ensures that a contiguous group of 256 indices remains contiguous through the Ha function, while ensuring that the function is collision-resistant.

The use of such scatter functions that remain local will reduce the cost of state access when the BSC switches to a state representation that rewards continuity.

3.2 Cross-chain interactions

One of the roles of NovaBridge is to support secure cross-calls in both directions between Immersera and the Layer 1 BSC. An account on one layer can send a transaction to the other chain, which will be executed asynchronously. In this section, we describe the outbox, which supports calls from the Immersera chain to the BSC, and two mechanisms, the inbox and the retrievable ticket, which support calls from the BSC to the Immersera chain.

3.2.1 Address aliases

When a Layer 1 BSC contract submits a transaction to the Immersera chain, it comes up what sender address should be attached to the transaction when running on Immersera. It is easy to simply send the contract using the L1 address, but there may be a contract in the Immersera chain and if this is the case both contracts will not be able to call the recipient Immersera, which would allow either one to emulate the other Immersera chain. This could be dangerous.

To avoid this, the address of the L1 sender at address A on layer 1 is denoted on the Immersera chain as f (A) = (A + C) mod 2160, where C is a specified odd constant. Because all BSC addresses and all other Immersera addresses are generated by hashing some data (the exact data depends on the source of the address), it is not feasible to generate a collision between an address and another Immersera address between the aliases. Immersera software trans-formats when interacting with the BSC, translating addresses in either direction as needed.

3.2.2 Outgoing mailbox

Immersera's outbox system allows arbitrary L2-to-L1 contracts to invoke messages initiated on L2 for eventual execution on L1. Given the security properties inherent to SuperTrust, the L1 execution of an outgoing message can only take place after the dispute period of its message has passed and its Rollups block has been confirmed (as described in Section

5) Logically, a message from L2 to L1 is like a "ticket" created on L2 that can later be "redeemed" on L1 to cause a specified transaction call to occur at L1. The recipient of the transactional call can verify that it is an authorized L2 to L1 message call and can confirm the L2 sender and the call data. This feature is sufficient to support the secure transfer of ETH, tokens or other forms of values from L2 to L1. For security purposes, an asynchronous ticket model needs to be used. Messages must be asynchronous because they cannot be redeemed confirmed until the RBlock containing them; redemption is done per ticket, not in strict order, because redeeming a particular ticket may require the execution of arbitrary code, which can be very expensive or even impossible in L1 gas.

L2 to L1 messages are initiated via an L2 transaction that invokes a special ArbSys precompilation that is part of NovaBridge. NovaBridge serializes the L2 sender's address, the ETH amount provided

at the time of the invocation, the L1 destination address, and the invocation data, which results in an L2 to L1 message.

The partial state asserted by the RBlock is the root hash of the Merkle tree of all L2 to L1 messages in the history of that chain. When an RBlock is asserted, this root hash is updated in the outbox contract for the chain on L1; at this point, the user can redeem it using the Merkle proof to include a message. the L1 outbox contract keeps track of messages that have been successfully redeemed, so each message can be redeemed at most once.

NovaBridge uses an efficient representation to support incremental computation of Merkle tree roots while requiring only logarithmic storage. Any kind of Merkle tree can be decomposed into a minimal set of complete binary subtrees of decreasing size. NovaBridge only remembers the size of the entire Merkle tree, and the root hash of these complete subtrees. Adding a new leaf to the tree will result in the state of a complete subtree that does not exist at all.NovaBridge issues an L2 EVM event containing the hash of the newly created subtree.Each contained proof in the Merkle tree version consists of a set of these subtree hashes, and a client that wants to create a proof can use the standard event search to find the hash containing the hash required for its proof L2 events. (The BNBN OVA node API includes support for automatic construction of these proofs.)

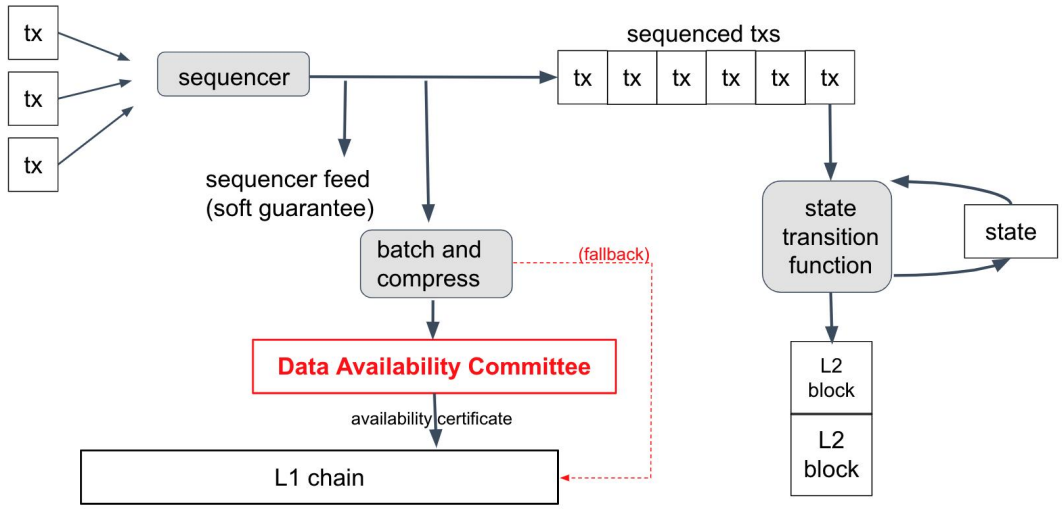
3.2.3 Inbox

Inboxes are managed by a set of Layer 1 BSC contracts that are responsible for recording messages (usually transactions) sent to the Immersera chain. Delayed inboxes receive messages committed on layer 1, while the main inbox receives messages sent by them

sequencer and merging messages from delayed inboxes.

The Delayed Inbox is a set of Layer 1 BSC contracts that receive messages to be delivered to the Immersera chain. This is an alternative to submitting via the sequencer. A deferred inbox is the only way for a Layer 1 contract to submit messages, since a Layer 1 contract cannot sign messages or submit them to a sequencer. It also provides a way for any user to submit messages without relying on the sequencer in case the sequencer is unavailable or misbehaves.

The Delayed Inbox is logically a queue. It keeps track of the number of messages submitted to it and the hash chain commitments for the contents of those messages. These messages are eventually copied to the main inbox, as described below.



The sequencer submits its data batches directly to the inbox contract. Each batch contains a compressed sequence of transactions and an instruction containing a specified number of messages from the Delayed Inbox header. NovaBridge's main input loop will use these sequential sequences of sequencer batches.

A well-behaved sequencer will include delayed inbox messages after a short delay, which is long enough to reduce the risk that reorganization of the layer 1 chain will cause the included messages to disappear or change. The current sequencer implementation includes delayed inbox messages after a ten minute delay. If the sequencer fails to include delayed Inbox messages within a fixed interval4 , either party can invoke Inbox to force the inclusion of messages, which is done by forcing the inclusion of a sequencer batch containing (multiple Inbox) messages into the Inbox. The ability to submit delayed inboxes and force inclusion of messages without relying on a sequencer supports Immersera's activity guarantees.

3.2.4 Transferable Tickets

Tier 1 contracts can submit transactions to the Immersera chain, but these transactions must run asynchronously on the Immersera chain, so committed Tier 1 transactions cannot see if they are

successful. This poses a problem for applications such as token bridging

4 Setting this parameter reflects a trade-off between the desire for cue inclusion and the desire to avoid unexpected behavior when the sequencer experiences downtime. It is currently set to 24 hours, but we anticipate that this value will decrease as the perceived risk of sequencer downtime decreases.

The ERC20 contract is supported with additional functionality to meet the mint/burn from the bridge contract, as well as callback hooks for revelation. Alternatively, to use a different contract as its L2 counterpart, an L1 token contract can register itself to any other "custom" gateway. The gateway router contract is responsible for keeping track of the L1 tokens mapped to the gateway (which in turn maps them to the L2 counterpart token)

Many additional token bridge features are theoretically possible, including irreplaceable token bridging, fast L2-to-L1 atomic switching, and locally deployed L2-to-L1 token bridging. Some standalone services provide enhanced bridging capabilities, often built on canonical bridging.

3.3 Gas and costs

Like many blockchains, the Immersera organization collects a fee from each transaction to cover the operating costs of the blockchain, adjust incentives, and ration resources when demand is high. Fees are charged by specific chain gases. For clarity, we will use the term Immersera gas to denote Layer 2 gas on the Immersera chain and L1 gas to denote Layer 1 gas on the BSC. Each EVM instruction has the same number of gas units on both chains; for exam- additionally, the MULMOD instruction requires Immersera8Immersera gas and BSC8L1 gas.

Each transaction requires a certain amount of Immersera gas, which depends on the resources used by the firm. The price of nitrous gas is equal to the current base fee and the algorithm changes as described below. Both nitro gas price and nitro gas payment are denominated in ETH.

The Immersera transaction specifies a gas limit, which is the maximum amount of Immersera gas it will be allowed to use sume. If the transaction tries to ✷ consume more Immersera gas than its limit,

the transaction fails, but it must pay for the Immersera gas it used. The transaction also specifies the maximum base fee it is willing to pay. If the current base fee is higher than the transaction's maximum, the transaction will not run (and therefore will not consume nitrocellulose gas). Together, these rules ensure that a transaction's nitrous gas

Expenses cannot exceed the product of their natural gas limit and the maximum base fee. By signing a transaction, the user authorizes the deduction of the cost of gas up to this amount from their ETH account, and Immersera respects this limit.
This approach preserves the BSC user experience and allows developers and users to use standard tools and wallets.

4 Compile for execution and proof

One of the challenges in designing a practical Rollups system is the tension between wanting the system to perform well in ordinary execution and being able to reliably prove the results of the execution.Immersera solves this problem by using the same source code for both execution and proof, but compiling it to different targets for the two cases.

When compiling the Immersera node software for execution, a normal Go compiler is used to generate native code for the target architecture, although this will vary for different node deployments. (The node software is distributed as source code and as a Docker image containing the compiled binaries.

In addition, the code part of the state transition function is compiled by the Go compiler to WebAusshgnt (wasm), which is a typed, portable machine code format. The wasm code then undergoes a simple conversion to a format we call WAVM, as described in detail below. If there is a dispute about the correct result of computing the STF, it is resolved by an interactive fraud proof protocol

4.1 WAVM

The wasm format has many features that make it a good tool for proving fraud – it is portable,

structured, well specified, designed to control the execution of untrusted code, and has pretty good tools and support - but it needs some modification to do the job completely. We have defined a slightly modified version of wasm that we call WAVM. a simple conversion phase will convert the wasm code from the compiler into proof-suitable WAVM code.

WAVM differs from wasm in three main ways. First, WAVM removes some features of wasm that are not generated by the Go compiler; the conversion phase verifies that these features are not present.

Second, WAVM restricts some of the features of wasm. For example, WAVM does not contain floating-point instructions, so the transformer replaces floating-point instructions with calls to the Berkeley soft floating-point library .8WAVM does not contain nested control flow, so the transformer adjusts the control flow structure to convert control flow instructions into jumps. Some wasm instructions require variable time to execute, which we avoid in WAVM by converting them to constructs using fixed-cost instructions. These transformations simplify the proof.

1).WAVM adds several opcodes to enable interaction with the blockchain environment. For example, the new instructions allow WAVM code to read and write the global state of the chain, fetch the next message from the chain's inbox, or issue a signal state transition function that signals the end of a successful execution.

2).We use software floating-point to reduce the risk of floating-point incompatibilities between architectures. The core Immersera functions do not use floating point, but the Go runtime uses some floating point.

4.1.1 Read image command

The most interesting new instruction is ReadPreImage as input hash H and offset, and return data of data in offset H (and return the number of bytes, which is zero if I am at or ✷ like the end of the image). Of course, it is generally not feasible to generate a preimage from an arbitrary hash. For safety reasons, the read-read ✷ image command can only be used in publicly known environments9

where the size of the known preimage is less than a fixed upper limit of about 110k bytes.

For example, the state of an Immersera chain is maintained in BSC's state tree format, which is organized as a Merkle tree. The nodes of the tree are stored in a database and indexed by the Merkle hash of the nodes. In Immerseraide, the state tree is kept outside the storage space of the STF, which only knows the root hash of the tree. Given the hash value of a tree node, STF can recover the contents of the tree node by using reads, depending on the fact that the complete contents of the tree are publicly known and the nodes in the BSC state tree are always smaller than the upper limit of the like size. In this way, STF is able to read and write the state tree arbitrarily, although only its root hash is stored.

The only other use of ReadPreImage is to get a hash of the contents of the most recent L2 block header. This is safe because the block headers are publicly known and have a finite size.

This "hash prediction technique" for storing data structures in Merkle hashes and relying on protocol participants to store the entire structure, thus supporting hash fetching of content, originated with the original Immersera design

4.2 WAVM modules

WAVM also allows virtual machines to compose multiple wasm binaries, called modules. Each module maintains its own code, global code, and memory. Modules can call other modules by cross-module calling WAVM instructions, and the called can read and write to the memory of the caller in order to pass data between them. This allows bootloaders to be written, functions written in Go to be stateful, and various libraries written in C, all running on the same WAVM machine. Without the module system, Go's memory management would interfere with C, but the module system allows them to maintain their own separate memory.

In this case, "publicly known" information is information that can be effectively obtained or recovered by any honest party, assuming that a complete history of the L1BSC chain is available. For convenience, hash preimages can also be provided by third parties, such as public servers, and the

correctness of the values provided can be easily verified.

4.3 One-step proof

The WAVM instruction set is designed so that a "one-step proof" covering the execution of a single WAVM instruction can be verified. Given a hash of the pre-state, a hash of the post-state, and a finite size witness, the BSC contract can verify that the execution of a single instruction using the pre-hashed state will produce a state with a post-hash.

For proof purposes, the hash of the WAVM state is computed as a certain Merkle hash on the WAVM/wasm VM state

5 Optimization Rollups protocol

Rollups protocol is Immersera's method of confirming L2 chain status and L1BSC related data

chain This is useful for contracts on the L1 chain, and for those parties who do not want to bother interacting with the L2 chain. But L2 users will usually not wait for L1 acknowledgement, rather they will rely on the decision-allowing transaction processing tic state transition function

The result from the sequence of recorded transactions.

The Rollups protocol produces a chain of Rollups blocks ("RBlocks"), which are not the same as L2 blocks. In short, an RBlock typically encapsulates a sequence of L2 blocks, so the number of RBlocks is much smaller than the number of L2 blocks. rBlock boundaries are not required and usually do not need to be aligned with the boundaries of sequencer batches.

RBlock consists of: an L2 block number, the header hash of the L2 block with that number, the number of incoming messages 10 consumed by the chain of L2 blocks

A summary of the outbox message output by the chain in the L2 block and earlier, a pointer to the predecessor RBlock, and

Additional bookkeeping information for tracking RBlock status in the protocol described below.

Initially, an RBlock simply indicates that a party claims that the data in the RBlock is correct. Eventually, each such statement is either confirmed or rejected by the protocol and then removed from the RBlock chain. This set of confirmed RBlocks will form a single chain start

10 The "message" here refers to a single transaction, or similar communication with the L2 chain, rather than a sequencer batch.

With the origin RBlock and grows over time. In general, the RBlock chain will consist of a chain of confirmed blocks, possibly followed by an unknown tree of solid RBlocks.

Each RBlock is valid if (a) the block is confirmed, or (b) the following is true:

The L2 block number of the RBlock, the header hash, the number of incoming messages and the summary of the message output all indicate the correct execution of the chain, as well as that any sibling of the RBlock (i. e.,) is invalid and that the predecessor RBlock is already valid.

By definition, the set of valid RBlocks will form a single chain with a set of confirmed RBlocks as prefixes.

A party can have shares in a particular RBlock that represent that party's assertion of the RBlock's validity. Because validity implies the validity of the predecessor, the party also asserts that the predecessor of that RBlock, as well as the predecessor chain back to the originating RBlock, are valid.

5.1 Common situations

The parties will be aware of the potential loss of shares on invalid RBlocks due to the following reasons, so if all parties follow their incentives, only valid RBlocks will be created. Those valid RBlocks will form a single chain that extends out of the confirmed chain RBlocks.

If an RBlock B is confirmed and B has a child, and the child is valid, and the time since the child was released is greater than a defined "challenge period" C, then the child can be confirmed. It follows that in the common case where both parties follow the above incentives, if an RBlock is posted at time T, it will be confirmed at T +C.

5.2 Challenges

If both parties do hold independent successors to the same RBlock, then both parties may face challenges. The party supporting the old successor will defend the viability of the L2 block number in the old successor and the correctness of the header hash, incoming message count, and output digest associated with that block number in the old successor. The other side will try to determine that one of these items is incorrect.

The challenge sub-protocol is described in detail below. In the overall Rollups protocol, its job is to identify one of the two competing parties

An incorrect statement (either hidden on one of the two rblocks or hidden at some point in the challenge subprotocol). The losing party has removed its shares from all rbl blocks. Half of the loser's shares will be donated to the winner and the other half will be added to the public goods fund.

The challenge subagreement ensures that the party whose initial claim is valid can always win the challenge by making a valid claim at each stage of the challenge. 12 Thus, an honest party (i. e., one that always makes a valid claim) will win every challenge. Because the honest party will eventually challenge every party that disagrees with it, the honest party will eventually eliminate all parties that disagree, and then the entire agreement can progress.

6. Supertrust: Immersera substance with external data

This section describes SuperTrust, a variant of Immersera that reduces costs by accepting a mild assumption of trust. support is included in the SuperTrust codebase to enable or disable SuperTrust functionality via configuration switches

The correctness of the Immersera protocol requires that all Immersera nodes have access to the data

for each L2 transaction in the inbox of the Immersera chain. On top, standard Immersera provides data access by publishing the data (in bulk, compressed form) as call data on the L1BSC. The BSC gas paid for this is the largest component of Immersera's costs.

Supertrust instead relies on an external data availability committee (the "Committee") to store data and make it available on demand. The Committee has N members, at least two of whom any Trust considers to be honest. This means that if N-1 Committee members commit to providing access to certain data, at least one hopeful party must be honest and ensure that the data is available so that the entire Immersera protocol can function properly.
In SuperTrust, the sequencer avoids publishing data from its batches on the L1 chain. Instead, for each batch, it
Issue a data availability certificate containing a hash of the data with which to prove that the batch data assumes that at least two committee members are honest and available to the committee.

7.1 Keysets

The key set specifies the public keys of the committee members and the number of signatures required for the data availability certificate to be valid. The keyboard makes the com-
The licensee's membership is subject to change and provides committee members with the ability to change their keys.

The key set contains the number of committee members and, for each committee member, the public key of an LBS, and the number of committee signatures required. Key sets are identified by their hashes. the L1 Key Set Manager contract maintains a list of currently valid key sets. The owner of an L2 chain can add

or remove the keyset from this list. When a keyset is valid, the keyset manager contract issues an L1BSC event containing the keyset's hash and full contents. This allows anyone to restore the content later, given only the keyset hash. While the API does not limit the number of key sets that can be valid at the same time, usually only one key set is valid.

7.2 data availability certificates

A central concept in SuperTrust is the Data Availability Certificate ("DACert"), which contains a hash of the data block and an expiration time for transactions under SuperTrust. Instead of publishing batches of data to the L1 chain, the sequencer sends wholesale to the Data Availability Committee and publishes the resulting DACert to the L1 chain, rather than the complete data.

Proof that a sufficient number of committee members have signed (hash expiration time) pairs, including the hash of the key set used for the signature, and a bit that shows which committee member signed, and a BLS aggregated signature [4] (over the BLS12-381 curve proves that both parties signed the agreement.

Due to the 2-of-N trust assumption, DACert constitutes evidence that the block data (i.e., the image hashed in DACert) will be available from at least one honest committee member, at least until it expires.

In a normal (non-SuperTrust) Immersera, the Immersera ware sequencer publishes blocks of data on the L1 chain as call data. The hash of the data block is submitted by the L1 inbox contract, allowing reliable reading of the data through the L2 code.

SuperTrust provides two ways for the sequencer to publish blocks of data on L1: it can publish the complete data as above, or it can publish a DACert to prove the availability of the DACert. the L1 inbox contract will reject any DACert that uses an invalid key set; other aspects of the validity of the DACert will be checked by the L2 code.

The L2 code that reads data from the inbox will read the full data block the same as in normal Immersera. If it sees a DACert

Instead, it checks the validity of DACert by referencing the key set specified by DACert (valid at the time of publication because of L1 inbox validation). the L2 code can verify that: the number of signers is at least the number required for the key set, and that the aggregated signature is valid for

the declared signers and that the expiration time is at least two weeks after the current L2 timestamp.

If the DACert is invalid, the L2 code will discard the DACert and act as if an empty batch was received.
If DACert is valid, the L2 code will read the data block and ensure that the block is available because DACert is valid.

7.3 Data Availability Server

Committee members run the Data Availability Server (DAS) software. das
Two api's are exposed: the sequencer API, called only by sequencers in the Immersera chain, is a JSONRPC interface that allows sequencers to submit blocks of data to
DAS for storage. Deployments typically block access to this API by callers other than the sequencer.

Available to the world, the REST API is a RESTful HTTP (S)-based protocol that allows blocks of data to be fetched via a hash. The API is fully cacheable and deployments can use caching proxies or CDNs to increase scale and prevent DoS attacks.

Only committee members have a reason to support the sequencer API. we expect others to be able to run the REST API, usually by mirroring other REST API servers, which is helpful.

7.4 Interaction of the sequencer with the committee

When the Immersera sequencer generates a data batch that it wants to publish using a committee, it sends the batch's data, along with the expiration time (typically in the next three weeks), to all committee members in parallel via RPC. Each committee member stores the data in its support repository and indexes it according to the data's hash. The members then sign the data using their BLS key pair (hash, expiration time) pair and return the signature to the sequencer.

Once the sequencer has collected enough signatures, it can aggregate the signatures and create a valid DACert for the (hashed, expired time) pair. the sequencer then publishes the DACert to the L1 inbox contract, making it available to L2's SuperTrust chain software.

If the sequencer fails to collect enough signatures in a reasonable amount of time, it can abandon its attempt to use the committee and "go back to Rollups" by publishing the complete data directly to the L1 chain, just as it would in a non-SuperTrust chain. the L2 software understands both data publishing formats (via DACert or complete data) and will handle each format correctly.

7.5 Pricing of any trust and L1

By significantly reducing the amount of L1 data required for the same number of transactions, SuperTrust results in much lower prices for transaction data. The same L1 pricing algorithm (see Section 1.3.2 for details) is used for a normal Immersera chain, but under SuperTrust, the sequencer has a much lower data expense (paying to publish data availability certificates instead of full data), so the pricing algorithm will evaluate a much lower price per unit of data traded by the user. If a sequencer on the SuperTrust chain does fall back to releasing complete data at Layer 1, it will report a higher expense and the data price will rise to compensate. No change is required in the pricing algorithm; only the results will be different.

7 Conclusion

By using the design described above, Immersera achieves high throughput, with unreliable security and activity guarantees, in a system that can be implemented and deployed today.

References

[1] Alakuijala, J., Farruggia, A., Ferragina, P., Kliuch-
nikov, E., Obryk, R., Szabadka, Z., Vandevenne, L..
Brotli: A general-purpose data compressor. ACM Transactions on Information Systems (TOIS) 37(1), 1- 30 (2018)

[2] Allegro Tech. BigCache https://github.com/
allegro/bigcache

[3] Barreto, P.S., Kim, H.Y., Lynn, B., Scott, M.: Ef-
ficient algorithms for pairing-based cryptosystems.
In: Annual international cryptology conference. pp. 354-369. Springer (2002)

[4] Boneh, D., Lynn, B., Shacham, H.. Short signatures from the weil pairing. In: International con- ference on the theory and application of cryptology and information security. pp. 514- Springer (2001)

[5] Carlson, J.. Redis in action. Simon and Schuster (2013)

[6] Dgraph. BadgerDB https://github.com/ dgraph-io/badger

[7] Donovan, A.A.. Kernighan, B.W.. The Go programming language. addison-wesley professional (2015)

[8] Hauser, J.R.: Berkeley softfloat release 3e (2018)

[9] Kalodner, H., Goldfeder, S., Chen, X., Weinberg. S.M., Felten, E.W.. Arbitrum. Scalable, private smart contracts. in: 27th USENIX Security Sym- posium. pp. 1353- 1370 (2018)

[10] Kelkar, M., Deb, S., Long, S., Juels, A., Kannan, S.. Themis: Fast, strong order-fairness in byzantine consensus. Cryptology ePrint Archive (2021)

[11] Kelkar, M., Zhang, F.. Goldfeder. S., Juels, A.. Order-fairness for byzantine consensus. in: Annual International Cryptology Conference. pp. 451-480. springer (2020)

[12] OpenZeppelin project. OpenZeppelin Contracts. https://github.com/OpenZeppelin/ openzeppelin-contracts

[13] WebAssembly Working Group. WebAssembly https://webassembly.org/

[14] Wood, G.: Ethereum: A secure decentralised gen- Ethereum Project Yel- low Paper 151, 1-32 (2014)