# Kmni: Ethereum-Native Interoperability

**Abstract**

Ethereum's adoption of the rollup-centric roadmap has created isolated execution environments, dividing capital, users, and developers across rollup domains. Addressing these issues, Kmniwas developed to provide Ethereum-native interoperability in a secure, performant, and globally compatible manner. Kmniestablishes a new precedent for secure interoperability by deriving its cryptoeconomic security from restaked $ETH. Building on its novel security framework, Kmnifeatures a custom network architecture that is capable of verifying cross-rollup messages with sub-second finality. Kmni's design facilitates ease of use and development across rollups with its emphasis on minimal integration requirements and the introduction of a universal gas marketplace alongside the KmniEVM. By establishing Ethereum-native rollup interoperability with minimal latency, Kmnialleviates the negative externalities associated with rollup fragmentation, enabling Ethereum to return to its role as a unified network for decentralized applications.

## 1 Fractured Blockchain Scaling

More than three years have elapsed since the Ethereum community endorsed the rollup-centric roadmap, marking the end of a comprehensive search for a scaling solution that would enhance Ethereum's throughput without sacrificing security, performance, or compatibility with the broader ecosystem. This search began during Ethereum's early development stages, which were inherently motivated by a desire to overcome the scalability and efficiency limitations observed in previous blockchain designs. While sharding was part of a long-term vision for scalability, the initial architecture focused on foundational elements that could later accommodate such advanced solutions. Building on this foundation, Ethereum's journey toward enhanced scalability saw the exploration of technologies including payment channels, state channels, and Plasma. This iterative process eventually culminated in the development of rollups, marking a significant advancement in Ethereum's scaling efforts.

By executing transactions off-chain and only storing data and/or proofs on-chain, rollups scale Ethereum in a secure, performant manner while ensuring compatibility with smart contracts and applications. Rollups provide secure off-chain execution environments by relying on Ethereum Layer 1 (L1) for consensus on state updates. Importantly, rollups also provide users with an uncensorable exit mechanism for transferring their capital back to Ethereum L1 in case of emergency. Leveraging Ethereum L1's robust security, rollups are free to optimize their execution environments for performance and compatibility. From a performance perspective, rollups are designed to reduce network congestion on Ethereum and lower transaction costs for users. In terms of compatibility, rollups' Turing-complete nature and support for multiple programming languages empower them to be adaptable to existing developer tooling and infrastructure services. This lowers the barrier to adoption for developers and enables them to create execution environments tailored for different use cases.

Despite these benefits, the rollup-centric approach forces Ethereum to scale via isolated execution environments. This creates a series of negative externalities that degrade Ethereum's network effects. The division of liquidity across multiple rollups diminishes the capital pool accessible to applications, leading to inferior economic properties compared to a unified

system. Consequently, distinct rollup environments force users to interact with multiple application instances across these isolated rollup economies.

For developers, this situation necessitates a shift in focus towards managing distributed state between environments rather than focusing on application functionality and utility. This requirement stands in stark contrast to the original design intentions of the Ethereum Virtual Machine (EVM). The EVM is engineered to abstract away the complexities of distributed state, even though it is operated by hundreds of thousands of nodes on Ethereum L1 [1]. This abstraction empowers developers to program as if they were dealing with a singular CPU and state machine, thereby simplifying application development and minimizing the risk of potential software vulnerabilities. However, the dispersion of users across rollups now requires developers to manage several subcomponents of their applications across independent rollups to access all of Ethereum's users and capital.

On our current trajectory, the expanding variety of rollup designs and their growing adoption will only exacerbate these issues. Consequently, Ethereum is faced with an existential threat that requires a native interoperability protocol purpose-built to realign Ethereum with its original vision of being a single, unified operating system for decentralized applications.

## 2 Interoperability for Ethereum Rollups

The purpose of any interoperability protocol is to verify the state of a source network and relay state updates to destination networks. To reflect the fundamental properties of its rollup ecosystem, the ideal Ethereum interoperability solution must provide this service in a way that is also secure, performant, and globally compatible with the Ethereum ecosystem. To meet our security standards, the solution should derive its security from the same source as Ethereum rollups: Ethereum L1. To be considered performant, the interoperability protocol must verify and process cross-rollup messages with minimal latency. Finally, for the protocol to be globally compatible, it must enable applications to be Turing-complete across all rollup environments, ensuring that applications are not limited by the resource constraints of any single rollup.

Interoperability protocols offer different security guarantees depending on how they verify the state changes of their supported networks. Existing protocol designs can be broadly classified according to the four categories of verification as defined in Table 1 [2].

Table 1: Interoperability Verification Types

| *Verification Type* | |
|---|---|
| **Native** | Security providers of the underlying networks validate the data communicated between networks. |
| **Local** | Only the parties involved in a given cross-network interaction verify the interaction. |
| **Optimistic** | Every message is assumed to be valid and full verification is only performed if a dispute is raised. |
| **External** | A set of third-party security providers is used to validate the data communicated between networks. |

Natively verified systems offer the only fully trustless approach to interoperability. However, these systems are the most difficult to implement since they require the security providers of the underlying networks to validate state changes. Given that rollups obtain their security from Ethereum L1, the sole method to establish a natively verified interoperability solution for rollups is to use Ethereum L1 for validating rollup state changes. This can be done with smart contracts that use optimistic or zero-knowledge verification mechanisms. However, the run times of these processes on Ethereum L1 carry high costs and latency with optimistic verification requiring a seven-day challenge period and zero-knowledge verification requiring several hours. As a result, Ethereum L1 does not meet our desired performance criteria and compels us to explore alternative verification mechanisms.

Similar to a strictly native approach, other interoperability solutions also fail to meet our three key criteria. Solutions that rely on local verification provide strong security guarantees and can be applied across various rollups, yet their inability to handle arbitrary messages limits their application compatibility. Optimistic approaches, which incorporate a latency period for potential challenges, inherently do not satisfy our performance requirement.

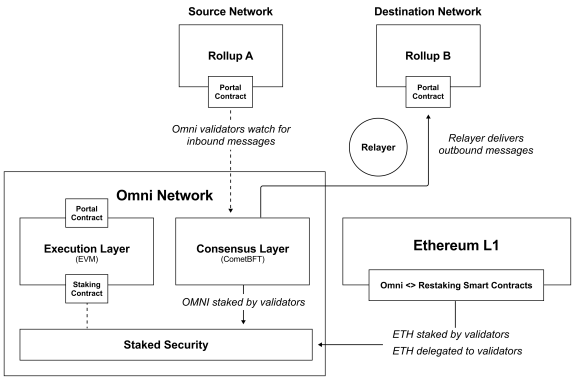
Externally verified systems are the only solution that can support our desired performance and global compatibility requirements. However, this approach introduces trust assumptions tied to an external verifier set, thereby compromising on our security requirement. To mitigate these trust assumptions, externally verified systems can introduce cryptoeconomic security measures that require the verification set to commit capital that is at risk of being slashed in the event of misconduct. Interoperability protocols that choose to implement this model can be considered game-theoretically secure as long as the staked value of their verifier set is greater than the value that the verifier set can transfer in a single state transition period. However, existing solutions that use this approach derive their cryptoeconomic security from a

native asset rather than Ethereum L1, making them unfit for our security requirements. By devising a method to extend Ethereum L1's cryptoeconomic security to an external set of verifiers, we can fulfill our security criteria and create a novel interoperability solution that offers all three of our desired properties.

## 3 Kmni

The guiding principle of modular design theory is the separation of a system into distinct, isolated components, each optimized for a specific function. Applying this approach to existing interoperability solutions, we created Kmni, a natively secured, externally verified interoperability network that establishes a new precedent in security, performance, and global compatibility for the future of Ethereum's modular ecosystem.

Figure 1: KmniProtocol Overview




Kmnirealizes this vision by introducing a novel network architecture tailored for low latency cross-rollup communications and global compatibility with Ethereum's entire rollup ecosystem, underpinned by the cryptoeconomic security of restaked $ETH. This design achieves sub-second cross-rollup message verification while harnessing Ethereum's industry-leading cryptoeconomic security budget. Furthermore, Kmniis intentionally designed to be easily in-t egrated with any rollup architecture and local rollup application, while also providing a programmable state layer for managing application deployments across rollups. The following sections of this whitepaper will explore the architectural evolution of Kmni, detailing how it fulfills our design requirements of security, performance, and global compatibility.

## 4 Security

Externally verified interoperability systems optimize for performance and compatibility but compromise on security guarantees. Their ability to easily connect with various networks and offer rapid verification allows them to provide a desirable user experience, contributing to their widespread adoption throughout the industry. However, securing these systems has been a persistent challenge, with no clear solution that offers robust and stable security at scale.

The simplest method to secure an externally verified system is to rely on a trusted verifier set. While straightforward to implement, this approach provides minimal security, especially when verifiers remain anonymous, thereby raising the risk of collusion. A marginally better strategy involves using public entities as verifiers, relying on their reputation as a deterrent against collusion. Still, this method remains vulnerable, as these entities can become targets for attacks, particularly if a single entity controls multiple verifiers. As a result, a number of protocols using these trusted verification methods have fallen victim to exploits, cumulatively costing the industry over $1 billion.

Improving upon the trusted verifier model, protocols can use cryptoeconomic security to introduce a new dimension of security to an external verifier set [3]. This approach has been attempted by several prior networks but encounters inherent limitations due to its reliance on the protocol's native asset for security. Under this approach, the scale of the protocol's security is directly tied to the demand for the protocol's asset, introducing reflexive dynamics that result in unstable security guarantees. To establish a security model that is both stable and secure at the scale of the modular ecosystem, it is essential to derive cryptoeconomic security from a broader network.

Prior to the advent of EigenLayer's restaking, a novel primitive that extends Ethereum L1's cryptoeconomic security to external networks by reusing staked $ETH from Ethereum's consensus layer [4], achieving this level of security was unattainable. Now, restaking enables Kmnito leverage the cryptoeconomic security of Ethereum L1 for its own validator set.

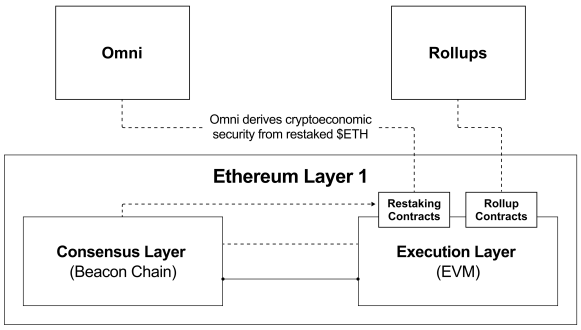
The cryptoeconomic security provided by restaked \$ETH, $C_e$, is given by the formula:

$$C_e = \frac{2}{3} \sum_{v=0}^{n} P(S_v)$$

- $S_v$ = the amount of \$ETH restaked by validator $v$
- $P$ = the function mapping the amount of restaked \$ETH to validator power
- $n$ = the total number of validators

With more than \$100 billion securing the network, Ethereum's current security budget is an order of magnitude larger than any other Proof of Stake (PoS) network [5]. By leveraging restaked \$ETH, a highly liquid, low volatility asset, Kmni's security achieves significantly greater stability than its predecessors. Moreover, by deriving security from Ethereum, Kmni aligns its security base with the rollups it connects, facilitating a security model that grows in tandem with Ethereum's modular ecosystem. By implementing a cryptoeconomic security model using restaked \$ETH, Kmnieestablishes a new paradigm for secure and reliable interoperability across the entire industry.

Figure 2: Security Overview




## 5 Performance

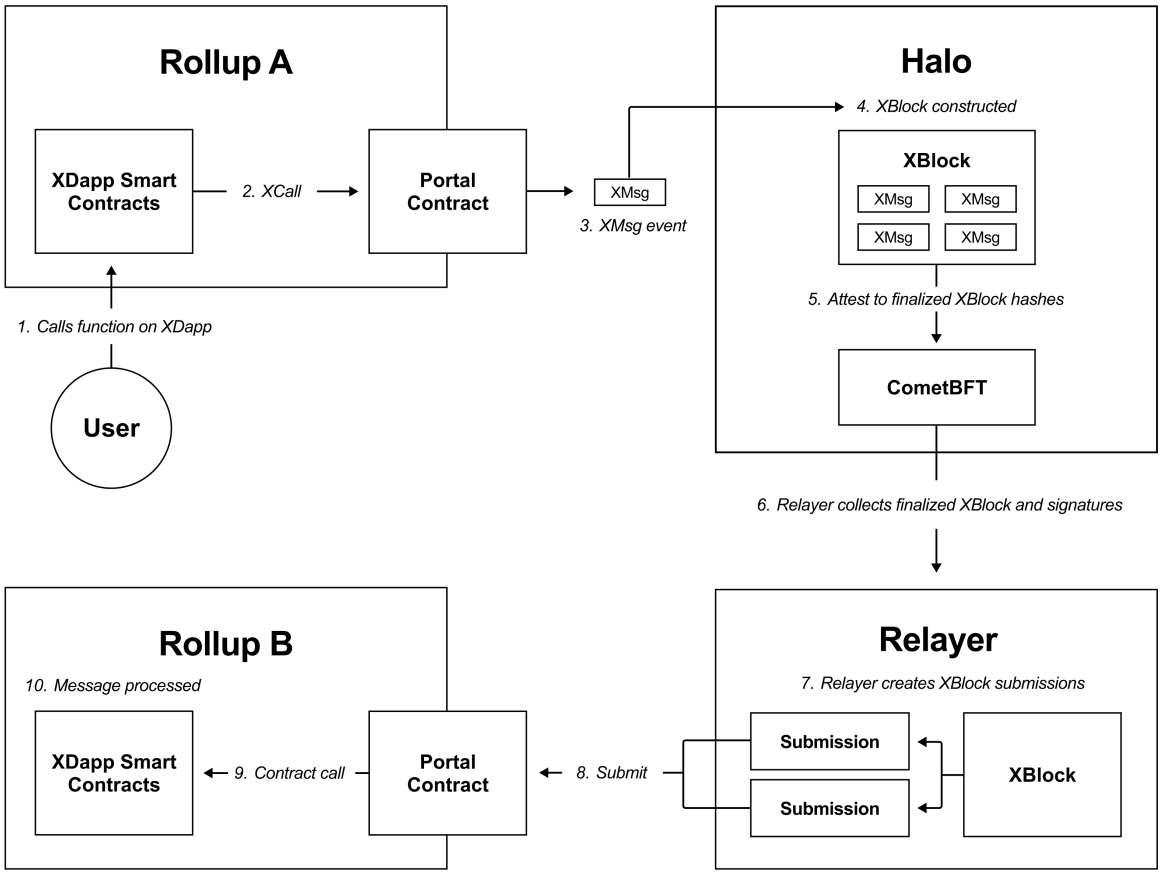
Having identified restaking as our desired security mechanism, our focus shifts to building a performant protocol architecture. Specifically, our goal is to create an externally verified protocol that minimizes the verification time for cross-rollup messages (XMsgs) and is secured by restaked \$ETH. The CometBFT consensus engine (i.e. Tendermint) offers properties that fulfill both of these requirements [6].

1. CometBFT delivers instant transaction finality, thereby removing the need for additional confirmations or concerns for block reorganizations within Kmni. This makes Kmni's finality time solely determined by the time it takes for Kmnivalidators to reach consensus.
2. CometBFT also provides support for Delegated Proof of Stake (DPoS) consensus mechanisms, a natural fit for Kmnivalidators to accept restaked \$ETH delegations.
3. Finally, CometBFT is one of the most robust and widely tested PoS consensus models that is used in several production blockchain networks that each secure billions of dollars.

Kmniuses CometBFT to process XMsgs according to the sequence visualized in Figure 3. First, Kmnivalidators operate full nodes for each rollup virtual machine (VM) to check for XMsg requests. For every rollup VM block that contains XMsg requests, Kmnivalidators build an XBlock containing the corresponding XMsgs. During CometBFT consensus, Kmnivalidators use the vote extension feature from Cosmos' second generation Application Blockchain Interface (ABCI++) to attest to XBlock hashes [7]. Finally, external relayers re-package and deliver finalized XMsgs to their destination rollups. Through this process, Kmniachieves sub-second XMsg finality backed by previously unparalleled levels of cryptoeconomic security.

The XMsg verification process is designed with a foundational security standard. Kmnivalidators wait for XMsg requests to finalize on Ethereum L1 (2 epochs, ∼12 minutes) before attesting to their validity in consensus. However, this design is intentionally decoupled from consensus so that low-latency finality mechanisms can be used. Specifically, this can be achieved using transaction insurance mechanisms on a message's source rollup or pre-confirmations from technologies like shared sequencers [[8],[9]]. By supporting these alternative finality mechanisms, Kmni will deliver an end-user experience for cross-rollup messaging that mirrors modern cloud-based applications.

Figure 3: XMsg Lifecycle




## 6 Global Compatibility

To this point, we have established a framework for creating a highly performant cross-network messaging service with unparalleled security guarantees. However, our ultimate goal for Kmniis to enable all applications to become Turing complete across all rollup environments. To accomplish this, Kmni must be designed to support any rollup architecture. It must also be backward compatible with existing smart contract application instances on rollups, simplifying the complexities associated with contract integrations and gas management. Finally, Kmnimust be expanded to support a global orchestration layer that alleviates the growing complexities of managing application deployments between rollup environments.

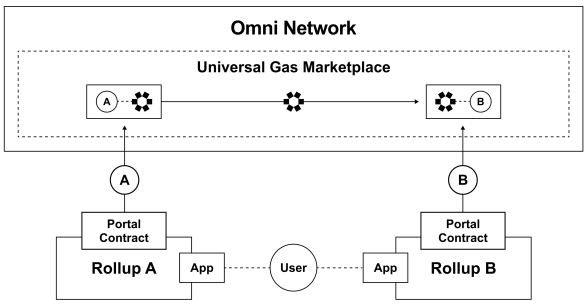
### 6.1 Rollup Diversity

Ethereum is still in the early phases of rollup design development. In the coming years, we anticipate the rollup ecosystem will proliferate in diversity. Projects will develop more customized rollup solutions, each tailored for specific functionalities and performance needs, incorporating unique virtual machines, programming languages, and data availability architectures. Understanding this progression, Kmni is intentionally designed with minimal integration requirements to ensure compatibility with any rollup architecture, reflecting the distinct layer separations within the Internet Protocol stack. Thus, Kmnipromotes innovation at the rollup level while functioning as a global hub that connects the entire rollup ecosystem.

## 6.2 Universal Gas Marketplace

To support a diverse range of rollup architectures, Kmnimust be capable of handling gas payments in diverse assets, ensuring that users can interact with any application regardless of where it is deployed. To achieve this, Kmniintroduces a universal gas marketplace that simplifies the process of gas payments across rollups. A "pay at source" fee model is used to support payments in the native asset of the source network. Internally, these payments are converted into $Kmni, serving as the currency for compensat-i ng relayers that facilitate transaction processing on the target rollup on behalf of users. Alternatively, the protocol can accept $Kmnias a gas payment m edium across all rollups. This design establishes $ Kmnias a gas abstraction primitive, allowing users to obtain $Kmnion any rollup and use it to pay for gas fees anywhere they transact. Using these mech-anis ms, Kmni's universal gas marketplace not only sim plifies transactions across rollups but also drives de mand for $Kmni, which correlates directly with th e volume of cross-network communications facili-ta ted by the protocol.

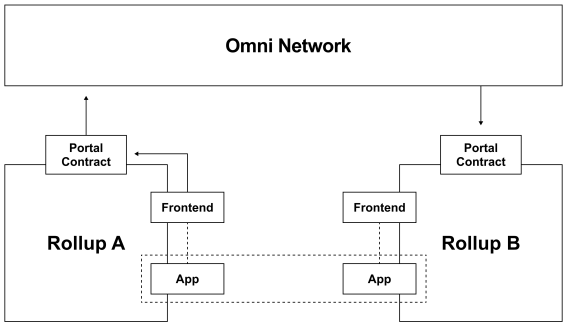
Figure 4: Universal Gas Marketplace




## 6.3 Backward Compatibility

Building on the foundation of Kmni's universal gas marketplace, the protocol is designed to offer backward compatibility with existing rollup applications. Specifically, applications can integrate Kmniwithout modifying their existing contracts. Instead, applications can employ modified frontend instructions to assemble an XMsg that is directed to the KmniPortal c ontract on the originating network. Subsequently,t he Kmniprotocol takes over, processing XMsg re-qu ests and executing their instructions on the tar-ge t network on behalf of the originating application.A s a result, Kmnieffectively functions as a wrapper around existing applications, allowing them to aggregate users and liquidity across all deployments.

Figure 5: Kmni's Backward Compatibility




For developers seeking a higher degree of customization and control, Kmnican be integrated directly into applications at the smart contract level using bespoke cross-network function calls. This empowers developers to transmit any type of generic message between rollups, thereby enhancing existing rollup-based applications or enabling entirely new applications with native interoperability.

# 7 Powering Natively Global Apps with the KmniEVM

While Kmniis designed with local rollup applications in mind, the growing diversity within the rollup ecosystem is increasing the complexity associated with managing these application deployments across rollups. Much like how an operating system's kernel abstracts away the intricacies of a CPU's subresources, rollup application developers require a tool responsible for orchestrating application state and configuration across rollups in an efficient manner. This removes the need for developers to think about distributed state management and empowers them to program cross-rollup applications as if they existed within a single state machine.

To satisfy this requirement, Kmnimust transition from simply facilitating rollup interoperability to providing an abstraction layer that allows developers to program across all rollups from a single environment. From a design perspective, this requires us to expand Kmni's current verification layer to offer its own execution environment, the KmniEVM.

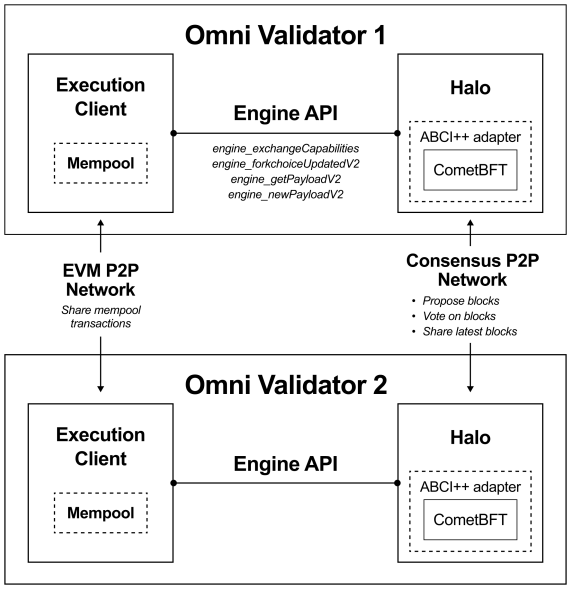
## 7.1 Network Architecture

To maintain compatibility with Ethereum's established ecosystem and developer tooling, it is essen-

tial for Kmnito adopt the EVM for its execution environment. Additionally, given Kmni's reliance on CometBFT consensus for its cross-rollup messaging service, we require a network architecture that combines the EVM with CometBFT consensus without compromising on the protocol's performance or security. However, previous frameworks for integrating the EVM with CometBFT make performance tradeoffs in several areas, with key challenges like mempool congestion and state translations between the EVM and CometBFT impacting consensus efficiency and limiting block times to around multiple seconds, even after optimizations.

We have developed a novel architecture that uses Ethereum's Engine API and ABCI++ to achieve our goal of combining the EVM with CometBFT consensus in a truly scalable manner. Drawing inspiration from Ethereum's PoS architecture, Kmninodes are configured with a new modular framework based on the Engine API. Kmni's framework creates a clear separation between its execution and consensus layers, thereby isolating the components that have bottlenecked performance in previous designs.

Figure 6: KmniNode Architecture




Existing approaches use the CometBFT mempool for handling transaction requests. As activity onthe network increases, this overloads the CometBFTmempool and compromises the network's consensusspeed. Kmniuses the Engine API to solve this problem, by moving the transaction mempool to the execution layer, thereby keeping the CometBFT consensus process extremely lightweight. The other major issue with previous design frameworks was the process of translating EVM state to a format that was compatible with CometBFT. We solved this problem by adding an ABCI++ wrapper around the CometBFT engine to handle state translation. Thus Kmni's Halo consensus client converts KmniEVM blocks into single CometBFT transactions inserted into consensus blocks. Since KmniEVM blocks are represented as a single consensus layer transaction, the only limiting factor for Kmni's throughput is performance of the EVM client itself.

The flexibility provided by the Engine API means execution clients can be substituted or upgraded without breaking the system. As a result, Kmnisupports any existing Ethererum execution client such as Geth, Besu, Erigon, and others, without specialized modifications. This approach eliminates the risk of introducing new bugs that are common with modified execution clients. Furthermore, Kmninodes can seamlessly adopt upgrades from any EVM client, ensuring ongoing compatibility with the Kmniconsensus layer. Kmninatively supports recent upgrades like dynamic transaction fees and partial fee burning from EIP-1559, as well as future upgrades such as Ethereum's introduction of proposer-builder separation and transient storage. Kmniensures the highest level of EVM equivalence by exactly mirroring the code run by validators on Ethereum L1.

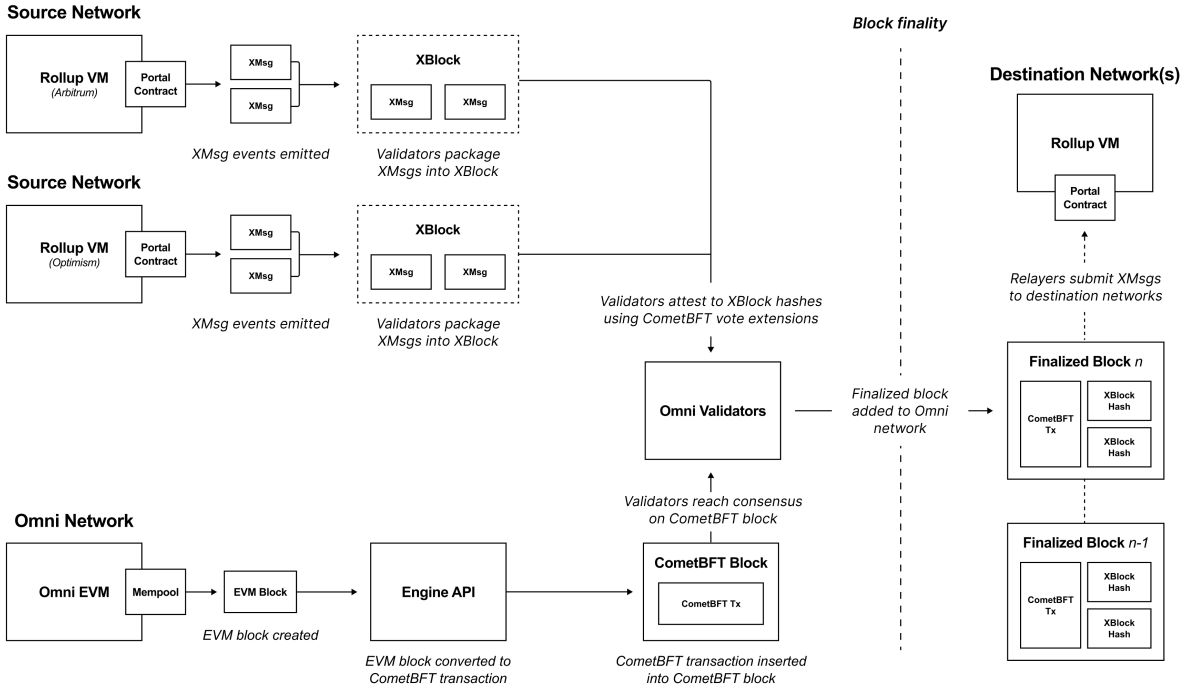
From our research, this novel framework for combining the EVM with CometBFT is capable of subsecond transaction finality at scale. To encourage wider innovation and enable other teams to leverage the best features of both Ethereum and Cosmos, we have open sourced this codebase as a public good for the industry [10].

## 7.2 Integrated Consensus

The addition of the KmniEVM requires Kmnivalidators to simultaneously manage consensus procedures for XBlocks as well as KmniEVM Blocks. To facilitate this, we have created a novel consensus process referred to as Integrated Consensus. This approach represents the culmination of Kmni's design objectives, enabling the network to achieve sub-second finality for both XBlock and KmniEVM Block verification in a single process, underpinned by the cryptoeconomic security of Ethereum L1.

Kmniimplements Integrated Consensus by intentionally decoupling the two consensus sub-processes prior to block finalization. For KmniEVM blocks,consensus is achieved through standard CometBFT procedures. During this process, Kmnivalidators use

Figure 7: Integrated Consensus Process



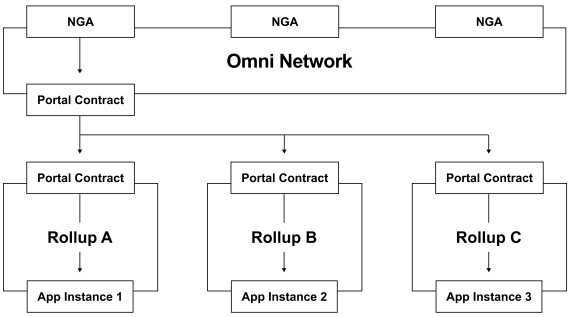

ABCI++ vote extensions to append XBlock attestations to the proposed CometBFT block. By separating these processes, Kmnireduces resource overlap for its validators and ensures that Integrated Consensus remains lightweight.

## 7.3 Natively Global Applications

The KmniEVM reinstates Ethereum's original vision of a unified operating system for decentralized applications by providing a global orchestration layer powered by the full scalability of Ethereum's rollup ecosystem. This equips developers with a new form of infrastructure that simplifies application management across diverse rollup environments and empowers them to build Natively Global Applications (NGAs). This new category of applications functions by dynamically propagating contracts and interfaces to any rollup, allowing them to access all of Ethereum's liquidity and users by default.

This approach to building cross-rollup applications not only streamlines the development process but also minimizes the potential for smart contract vulnerabilities that often arise from the intricacies of working with distributed state. For users, NGAs provide the convenience of interacting with any rollup through a single access point, without the need to navigate between different networks. As the landscape of application management grows increasingly complex, NGAs are positioned to become the new standard application design for Ethereum's modular ecosystem.

Figure 8: Natively Global Applications



## 8 Reinforced Security

Through the process of making Kmniglobally compatible with the entire Ethereum ecosystem, we have established two sources of demand for $Kmni. First, $Kmniserves as the universal gas resource for facilitating messages between rollups. Second, $Kmnicapital, users, and developers across a growing numfunctions as the gas resource powering NGAs on the KmniEVM. These functionalities present an opportunity to reinforce Kmni's cryptoeconomic security model using staked $Kmniin addition to restaked $ETH. Effectively, the total cryptoeconomic security of Kmniis determined by the combined staked value of these two assets.

Using this dual staking model, the total cryptoeconomic security $C$ of the system is given by the formula:

$$C = \frac{2}{3} \sum_{a=0}^{m} \sum_{v=0}^{n} P_a(S_{a,v})$$

- $S_{a,v}$ = the amount staked by validator $v$ for asset $a$ $v$
- $P_a$ = the function mapping the amount of asset $a$ staked to validator power
- $n$ = the total number of validators
- $m$ = the total number of unique staked asset types

The protocol will implement separate functions for mapping existing stake to voting power and for mapping existing stake to staking rewards. The protocol dynamically incentivizes validators to contribute more security from either restaked $ETH or $Kmnidepending on market conditions.

By implementing this dual staking model, Kmni's security now scales across two dimensions. Restaked $ETH anchors Kmni's security to Ethereum L1, enabling it to grow in line with Ethereum's own security budget. The addition of staked $Kmnibuilds upon this base, expanding Kmni's security alongside its own network activity. Collectively, these two complementary mechanisms provide robust and dynamic security guarantees for Kmni, setting a new standard for secure interoperability for the Ethereum ecosystem.

## 9 Summary

The adoption of the rollup-centric roadmap has committed Ethereum to scaling across isolated execution environments. While this approach is actively solving the network's scalability challenges, it has fragmented ber of rollup domains. To solve these problems, we created Kmni, an Ethereum-native interoperability protocol that establishes low latency communications across Ethereum's rollup ecosystem.

In our quest to build the optimal interoperability solution for Ethereum, we found that existing protocol frameworks failed to meet our criteria for security, performance, and global compatibility. Thisprompted us to create a new design framework forKmni, prioritizing security as the protocol's founda-tion. Through restaking, Kmnideriveseconomic security from Ethereum L1 and uses it to secure its externally verified network architecture. The integration of a dual staking model further strengthens this security architecture and positions Kmnias a new benchmark for secure interoperability.

With a robust security framework in place, we shifted our attention to optimizing Kmni's performance. Our goal was to implement a consensus mechanism capable of handling cross-rollup communications with minimal latency. Through the development of a unique protocol architecture, incorporating technologies like CometBFT, ABCI++, and the Engine API, Kmnivalidators achieve this goal by providing sub-second verification for cross-rollup messages.

In terms of global compatibility, Kmniwas intentionally designed with minimal integration requirements to make it compatible with any rollup architecture and application design. The introduction of a universal gas marketplace simplifies the user experience for cross-network applications, while the addition of the KmniEVM provides developers with a global platform for deploying and managing cross-rollup applications.

Kmnirepresents a comprehensive interoperability solution for Ethereum rollups and is poised to reunify the Ethereum ecosystem. With Kmni, Ethereum can once again provide a single, unified operating system for decentralized applications, but this time, at a global scale.