BINANCE RESEARCH

zkEVM and the Future of Ethereum Scaling

Stefan Piech

Table of Contents

Key Takeaways

- Zero-knowledge proofs have brought benefits to user privacy for multiple years already, and further evolution of the technology can help to foster a more secure operating environment in the future

- At the end of the day, zkEVMs are only one way to scale Ethereum - they have tough competition with optimistic rollups since they had time to establish themselves already. That said, zk-rollups (especially combined with EVM compatibility) offer numerous upsides to the Ethereum mainnet

- STARKs and SNARKs both have their benefits, and while we see a higher adoption rate of SNARKs in the foreseeable future, this might change, considering some of the technical benefits that STARKs bring

- Entering the ongoing debate if it is the holy grail to be fully Ethereum equivalent, compatible, or highly specialized, we reject the narrative of some zk-rollups and key opinion leaders that the more compatible to Ethereum zk-rollups are, the better they are. Instead, we see a need and use cases for all ends of the spectrum

- In the long term, we can envision highly EVM-compatible zk-rollups building the future of blockchains. At the same time, we expect to see more specialized zkVMs help develop the Web3 space as a whole, tackling unique problems with custom solutions

Introduction

Ethereum is scaling, and it is scaling fast. The Ethereum community is busy building, and multiple approaches to scaling Ethereum have found traction over the past few months. While there are multiple approaches, Layer 2 solutions have been at the forefront of the scaling narrative for some time now. As of now, the narrative has been dominated by projects focusing on optimistic rollups such as Arbitrum and Optimism. However, zero-knowledge rollups might have their time to shine now. "Why" you ask? Because **zero-knowledge Ethereum Virtual Machines (or simply zkEVM) might introduce substantial benefits to zk-rollups that will make them more competitive in the long term** - allowing them to (potentially) dethrone the current optimistic solutions.

Both zero-knowledge proofs and Ethereum Virtual Machines are concepts that have been around for some time now but bringing them together in a fully operational way, this soon after the Ethereum merge, might heat up the debate around Layer 2 scaling solutions again.

While zkVMs differ in their ambitions, zkEVMs all seem to aspire to a similar goal - creating a zk-rollup experience that feels almost exactly like using Ethereum's layer 1 blockchain. In this report, we'll be taking a step back, looking at zero-knowledge proofs and EVMs first, to build a foundational understanding of these two concepts. Not leaving the task incomplete, we will then look at zkEVMs in more detail and look at the different types of zkEVMs. We will also look at some of the leading projects in the space to see who is building what. And most importantly, we will think about how this will shape the future of crypto and Ethereum scaling going forward.

What is zkEVM?

Zero-knowledge Ethereum Virtual Machines ("zkEVMs") are pretty cool - to understand why, we should take a step back and look at what they're all about. zkEVM itself is a virtual machine (on Ethereum) that generates **zero-knowledge proofs** to verify the correctness of a statement. But there is more to it, and in an old-fashioned way, it will help if we break down the different components of zkEVM to understand them better.

Zero-knowledge ("zk") proofs are a way of proving the validity of a statement without revealing the statement itself. This is quite important when thinking about privacy, but zero-knowledge proofs are more than that. While zk proofs (also known as validity proofs) were originally mainly used for privacy-focused blockchains, they have since expanded their use case and reached Ethereum as a scaling solution. Let's explore them a bit more...

SNARKs vs. STARKs

A zero-knowledge proof, simply put, is a method by which one party can prove to another party that something is true. And they do so without revealing any information apart from the fact that this specific statement is true.

Every zk-proof needs to fulfill the following three criteria

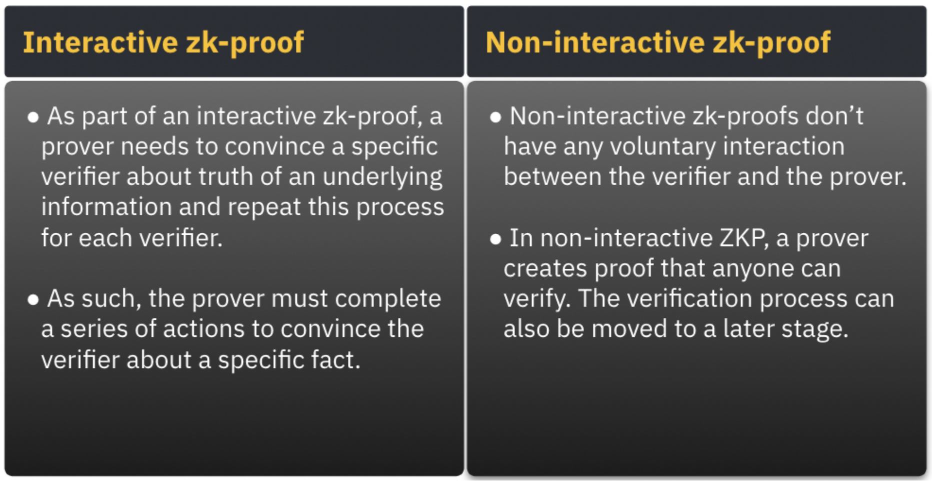
- **Completeness -** If the input is valid, we want the zero-knowledge protocol to return the value "true". This allows us to make the assumption that the proof can be accepted (assuming the prover and verifier act honestly)

- **Soundness -** Assuming that a prover would act dishonestly, we would want the verifier to be able to reject an invalid statement rather than accept it as true (with high probability)

- **Zero-knowledge -** The underlying assumption of zk-proofs is "zero knowledge". This means that the verifier learns nothing about a statement beyond its validity or falsity. This requirement also prevents the verifier from deriving the original input from the proof

We can differentiate between two fundamental types of zk-proofs - interactive and non-interactive. While the former was more popular in the early days, we have a trend that tends towards non-interactive zk-proofs. The reason for this is straightforward. Unlike interactive proofs, non-interactive proofs only require one round of communication between participants. In its simplest form, the prover would be in control of secret information that would be passed on to an algorithm that computes a zero-knowledge proof. The verifier would check that the prover knows the secret information with the help of another algorithm.

Figure 1: Differences between interactive and non-interactive proofs



Interactive zk-proof	Non-interactive zk-proof
● As part of an interactive zk-proof, a prover needs to convince a specific verifier about truth of an underlying information and repeat this process for each verifier. ● As such, the prover must complete a series of actions to convince the verifier about a specific fact.	● Non-interactive zk-proofs don't have any voluntary interaction between the verifier and the prover. ● In non-interactive ZKP, a prover creates proof that anyone can verify. The verification process can also be moved to a later stage.

Source: Binance Research

In its most basic form, a zero-knowledge proof is made up of three elements. A witness, a challenge, and a response. Let's have a closer look at each of these three elements in more detail.

- **Witness -** Using zero-knowledge proof, the prover wants to prove knowledge of some hidden information. The hidden information is oftentimes referred to as "witness" to the proof. Using a set of questions that can only be answered by someone with knowledge of the information, the prover would randomly choose a question, calculate the answer, and send it to the verifier (for interactive zk-proofs)

- **Challenge -** Using interactive zk-proofs, the verifier would next pick another question randomly and ask the prover to answer it. Using non-interactive zk-proofs would remove interaction between the verifier and the prover by sharing a common, short, random string

- **Response -** During a response, a prover would accept the question and next calculate the answer that will be returned to the verifier. The prover's response allows the verifier to check if the former really has access to the witness (hidden information). To ensure the prover isn't guessing blindly and getting the correct answers by chance, the verifier and the prover repeat this interaction many times (for interactive zk-poofs) to lower the possibility of the prover faking knowledge of the witness

Now that we have a basic understanding of zero-knowledge proofs, let us look at the major classifications of zk-proofs used in blockchains. These include zk-SNARKs, zk-STARKs, as well as recursive zk-SNARKs.

zk-SNARK

zk-SNARK is an acronym and actually stands for "**Zero-Knowledge Succinct Non-Interactive Argument of Knowledge**". While the word is a lot more daunting than its acronym, it will be less scary once we split zk-SNARK into its components.

- **Zero-knowledge -** At this point, we should have a good understanding of what zero-knowledge is, making this the most straightforward component of zk-SNARKs. Simply put, a verifier can validate the integrity of a statement without knowing anything else about the statement. The only knowledge the verifier has of the statement is whether it is true or false

- **(S)uccinct -** The zk-proof is smaller than the witness and can be verified quickly

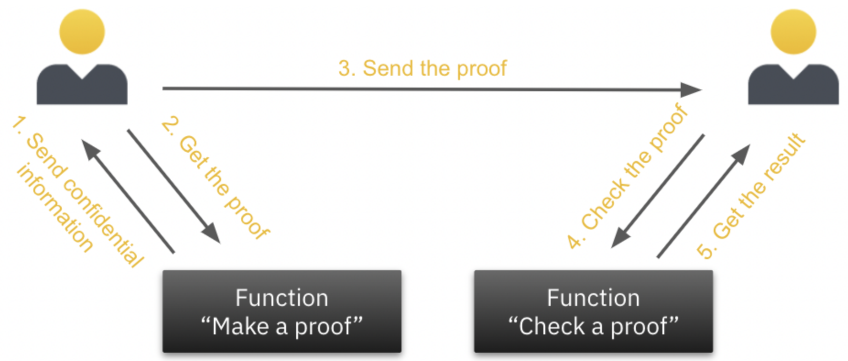
- **(N)on-interactive -** The proof itself is 'non-interactive' meaning the prover and verifier only interact once using algorithms

- **(AR)gument -** The proof satisfies the 'soundness' requirement, so cheating is unlikely

- **(K)nowledge -** The zero-knowledge proof cannot be construcdted without access to the secret information (witness). The notion of knowledge soundness implies that there is an extractor that can compute a witness whenever the adversary produces a valid argument

Validity proofs generated using zk-SNARK can prove the validity of a transaction without revealing the inputs. zk-SNARKs can also confirm the correctness of a computation performed off-chain without nodes replaying every step of the calculation. This is where zk-SNARK becomes useful for scaling blockchains.

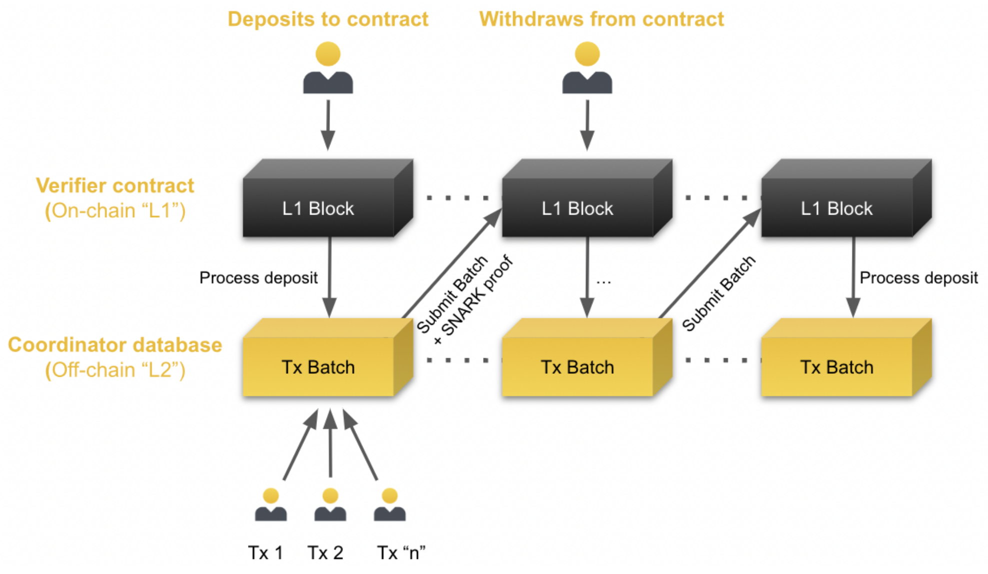
Figure 2: Design of zk-SNARKs



Source: Binance Research

Considering the use of zk-SNARKs within the rollups space, we can observe that the majority of projects follow the same structure. Users of the respective Layer 2 ("L2") sign transactions and subsequently submit them to validators. In the next step, validators would compress the transactions into a block and generate a corresponding validity proof (SNARK). Using zk-SNARKs, we can guarantee that the new state results from the addition of valid transactions only. In the final step, a smart contract on the underlying Layer 1 ("L1") blockchain determines if batched transactions are validated and posted on the main blockchain.

Figure 3: Execution of contracts using zk-SNARKs



Source: Simon Brown

Before moving on, let us have a closer look at the benefits and drawbacks of using zk-SNARKs.

Figure 4: Benefits and drawbacks of zk-SNARKs

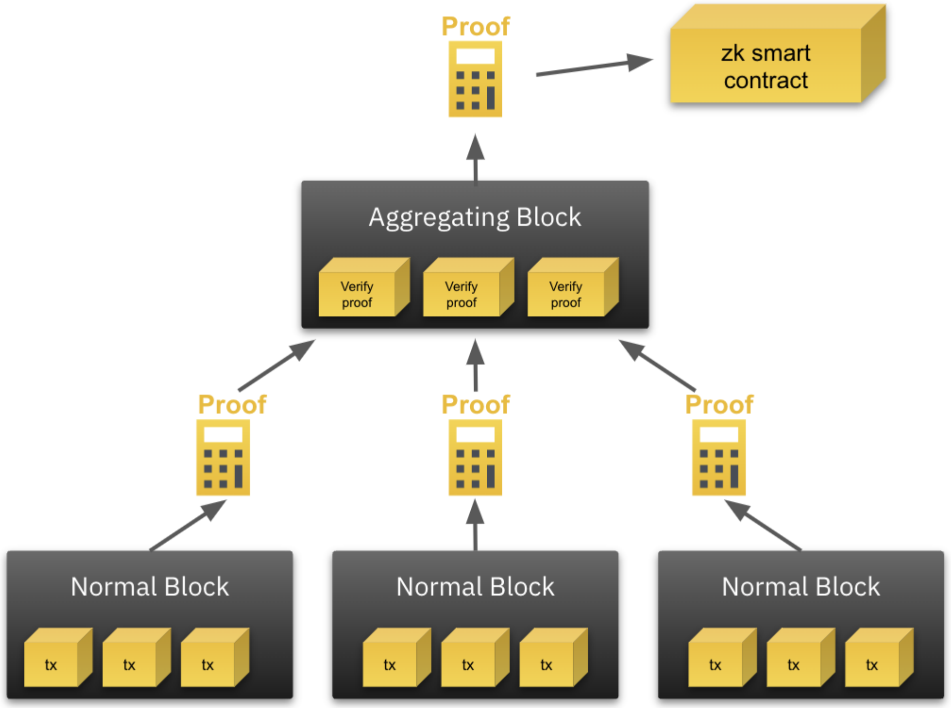
Benefits	**Drawbacks**
High throughput	**Proof verification costs**
zk-SNARKs scale throughput by shrinking computation on Ethereum's base layer. Using zk-SNARKs, rollups can increase TPS rates into the thousands. zk-SNARKs are many times smaller than the transaction data it verifies. This reduces congestion on the base blockchains and enables cheaper gas fees.	Verifying proofs requires complex computation and increases the costs of implementing zk technology in applications. This cost is particularly relevant in the context of proving computation. For example, zk-rollups can pay around 500,000 gas to verify a single zk-SNARK proof on Ethereum.
Small proof sizes	**Trust assumptions**
The small size of SNARK proofs makes them easy to verify on the main chain. On Ethereum, this means lower gas fees for verifying off-chain transactions, reducing the overall roll-up costs for end-users.	Public parameters are created via a trusted setup ceremony, where participants are assumed to be honest. However, there's no way for users to assess the honesty of participants. zk-STARKs - in contrast - are free from trust assumptions since the randomness used in generating the string is publicly verifiable.
Security	**Quantum computing threats**
zk-rollups are considered more secure than other scaling projects due to the cryptographic security mechanisms used in zk-SNARKs. The zk-SNARK proof is computationally sound, making it difficult to deceive verifiers and act maliciously. The non-interactiveness of ZK-SNARKs also means proofs can be trustlessly verified by anyone.	zk-SNARK uses elliptic curve cryptography (ECDSA) for encryption. While the ECDSA algorithm is secure for now, the development of quantum computers could break its security model in the future. zk-STARK, on the other hand, is considered immune to the threat of quantum computing, as it uses collision-resistant hashes for encryption. Unlike public-private key pairings used in elliptic curve cryptography, collision-resistant hashing is more difficult for quantum computing algorithms to break.

Source: Binance Research, Ethereum Foundation, Alchemy

Recursive zk-SNARKs

One sub-category of zk-SNARKs is that of so-called recursive zk-SNARKs. A recursive SNARK system generates proofs in parallel for different transaction blocks and aggregates them into one single block proof that gets submitted to the main blockchain, meaning one SNARK can verify other SNARKs. The L2 rollup still submits one validity proof on Ethereum. However, this "recursive proof" verifies transactions in multiple L2 blocks, all of which become valid once the on-chain contract accepts the submitted proof.

Figure 5: Illustration of a recursive zk-SNARK



Source: Binance Research, Matter Labs

Recursive zk-SNARKs dramatically increase the number of transactions that can be finalized using zk-proof by including multiple L2 proofs in a single proof submitted to the L1 chain. This is important since rollups can only submit one on-chain transaction (and the corresponding validity proof) per block, limiting the number of transactions that can be processed.

zk-STARKs

zk-STARK is an acronym for "**Zero-Knowledge Scalable Transparent Argument of Knowledge**". They are overall similar to zk-SNARKs, with the exception of two key differences.

- **Scalability -** zk-STARK is faster than zk-SNARK at generating and verifying proofs when the size of the witness is larger. While for zk-SNARK, the prover and verifier times increase linearly with the size of the hidden statements, STARK generally scales better with only slight increases in verification time as the size of the witness grows.
- **Transparency -** zk-STARK is more transparent compared to zk-SNARKs due to the usage of publicly verifiable randomness to generate public parameters for proving and verification instead of a trusted setup.

There are other (smaller) differences between STARKs and SNARKs. One is that zk-STARKs produce larger proofs than zk-SNARKs due to higher verification overheads. As mentioned above, this changes, however, once a dataset becomes large enough.

Considering that zk-STARKs are more scalable and transparent leaves the question of why a majority of projects (with the exception of StarkWare, Polygon Miden, and a few others) are using zk-SNARKs instead of zk-STARKs. If we compare both approaches, we can note that while zk-STARK is superior in terms of scalability (for larger datasets) and transparency, verification takes longer using zk-STARK due to the larger proof size. In addition to that, zk-SNARKs use lower amounts of gas, which suggests that using zk-SNARKs is substantially less expensive for end-users.

Figure 6: Differences between SNARKs and STARKs

	SNARKs	STARKs
Algorithmic complexity - prover	O(N * log(n))	O(N * poly - log(n))
Algorithmic complexity - verifier	O(1)	O(poly - log(n))
Proof size	O(1)	O(poly - log(n))
Size estimate for 1 tx	Tx: 200 bytes, Key: 50 MB	45 kb
Size estimate for 10.000 tx	Tx: 200 bytes, Key: 500 GB	135 kb
EVM verification gas cost	~ 600 k	~ 2.5 M
Trusted setup required	Yes	No
Post-quantum secure	No	Yes
Crypto assumption	Strong	Collision resistant hashes

Source: Binance Research, Matter Labs, Jens Groth

EVM (Ethereum Virtual Machine)

At the heart of the Ethereum protocol and operation is the so-called Ethereum Virtual Machine ("EVM"). It is the part of Ethereum that handles smart contract deployment and execution. But there is more to EVMs than the eye can see. Virtual machines ("VMs") are great ways to scale a decentralized network since they can be run on different operating systems from almost any geographical location. Ethereum's virtual machine, for example, uses a decentralized network of nodes to execute smart contracts.

When using Ethereum, every execution of a smart contract will subsequently trigger a change in the state of the Ethereum virtual machine (referred to as state transition). As such, EVMs are oftentimes also described as "state machines" since they are responsible for computing the state changes.

To get a better understanding, let us have a high-level look at the most simple form of a smart contract transaction:

1. **Contract bytecode** is loaded from the EVM's storage and executed by peer-to-peer nodes on the EVM. Due to the fact that nodes use the same transaction inputs, we can guarantee that each node arrives at the same result

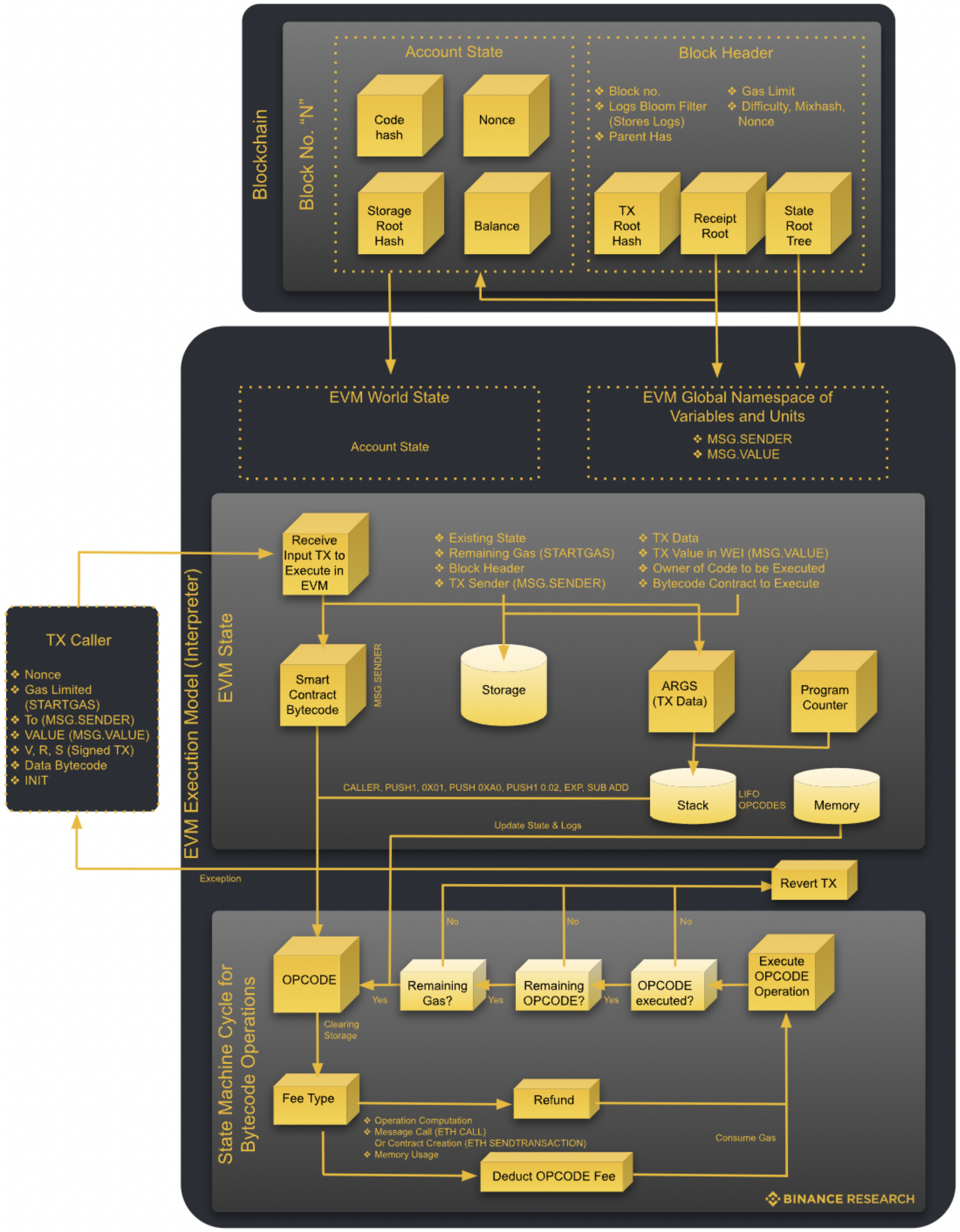
2. **EVM Opcodes** (contained in the bytecode) next interact with different parts of the EVM's state (memory, storage, and stack[1]). Opcodes perform read-write operations—reading values from state storage and write/send new values to the EVM's storage

3. In the final step, **EVM opcodes** perform computation on the values obtained from state storage before returning the new values. This update results in the EVM transitioning to a new state

The EVM is a quasi–Turing-complete state machine, because all execution processes are limited to a finite number of computational steps by the amount of gas available for any given smart contract execution. Both Polygon and BNB Chain, are direct forks of Ethereum, and therefore use the EVM as their run time.

[1] The EVM has a stack-based architecture, storing all in-memory values on a stack.

Figure 7: Ethereum Virtual Machine Architecture illustrated



Source: Binance Research, Ethereum Foundation

Today, the term EVM Chain oftentimes extends beyond just mirroring the above-mentioned architecture. There are several major specifications that began on Ethereum and have become de facto global standards.

- **Solidity -** A high-level language that compiles into EVM bytecode
- **JSON-RPC Client API -** A specification for interacting with Ethereum nodes
- **ERC20/ERC721 -** Ethereum token standards
- **Ethers.js -** A web library for interfacing with Ethereum
- **Cryptography -** E.g., keccak256 (hash function), secp256k1 (ECDSA signatures)

Compliance with these standards makes it substantially easier to use Ethereum tools. An excellent example is Polygon, which in addition to using all the above tools, is able to run a forked version of Etherscan (Polygonscan), use Ethereum developer tools like Hardhat, and be supported as a different Ethereum network in wallets like Metamask.

zkEVM - Merging zero-knowledge proofs and EVMs

Now that we have a common understanding of zero-knowledge proofs and EVMs, let us combine those two into zkEVM. At this point, you probably know without me telling you, but a zkEVM is an EVM-compatible virtual machine that supports zero-knowledge proof computation. However, combining those two elements is not as easy as it sounds. Zk-proofs require a very specific format (algebraic circuit) for all their computational statements, which can then be compiled into STARKs or SNARKs. Creating zkEVMs allows general-purpose rollup to run the EVM as its smart contract engine, and thus maintain compatibility with the common interfaces of the Ethereum ecosystem, making it easier to migrate existing contracts and tooling applications onto the rollup.

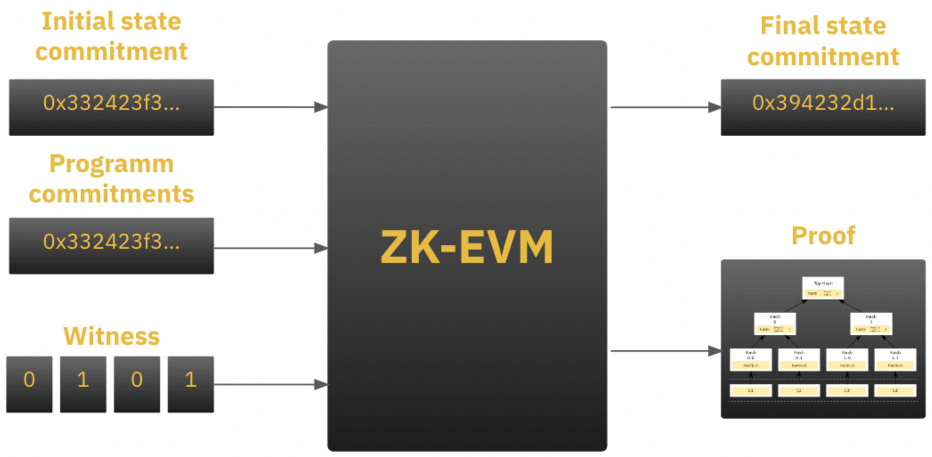
ZkEVMs can be divided into three parts. An execution environment, proving circuit, and verifier contract. Each component contributes to the zkEVM's program execution, proof generation, and proof verification.

- **Execution environment -** The execution environment is where smart contracts run in the zkEVM. The zkEVM's execution environment functions similarly to that of the EVM

- **Proving circuit -** The job of the proving circuit is to produce zero-knowledge proofs that verify the validity of transactions in the execution environment.

- **Verifier contract -** Zk-rollups submit validity proofs to a smart contract deployed on the L1 chain (Ethereum) for verification. The input and output are also submitted to the verifier contract. Subsequently, the verifier runs computation on the provided proof and confirms that the submitted outputs were correctly computed from the inputs.

Figure 8: Simple Design Visualization of zkEVMs



Source: Binance Research, BNB Chain

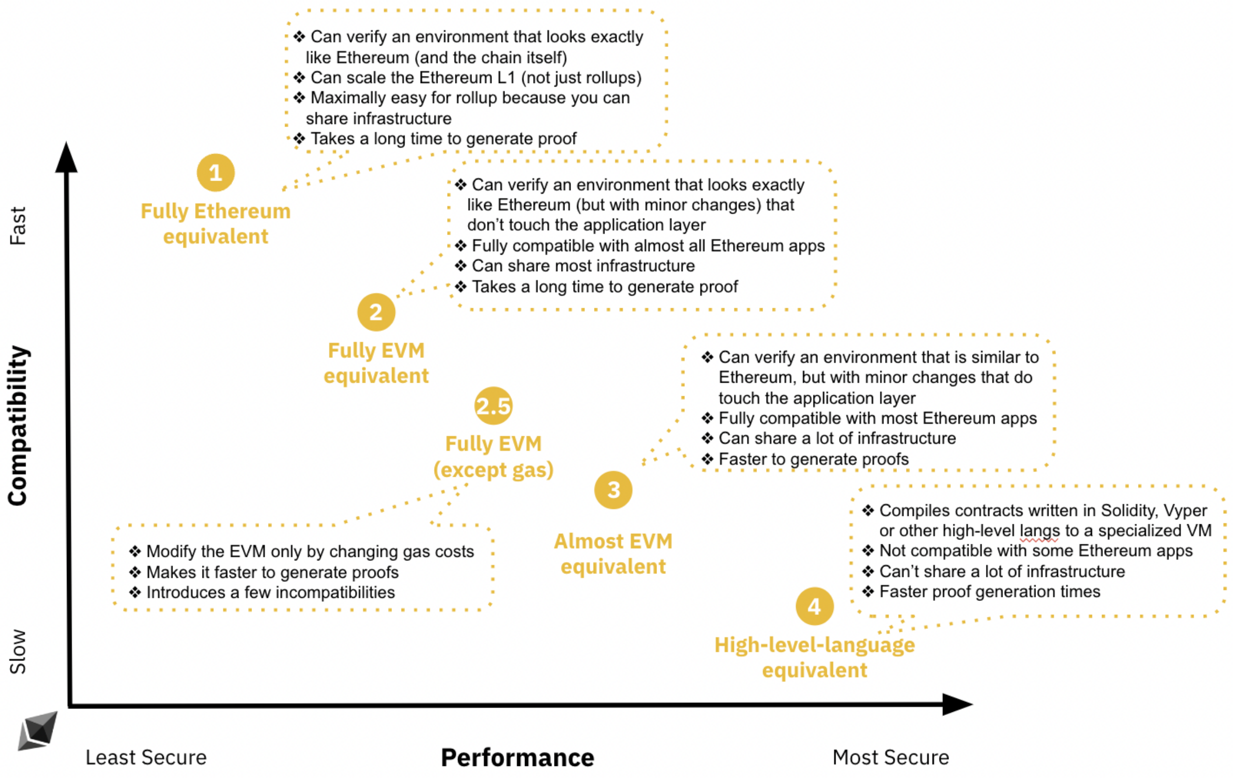
There are a few elements that make building zkEVMs difficult. Special opcodes, a stack-based architecture, high storage overhead, and high proving costs are just some of them. However, once implemented successfully, we will likely benefit from faster finality and capital efficiency as well as more secure scaling overall.

Since different zkEVM projects adopt different approaches to combining EVM execution with zero-knowledge proof computation, we will have a closer look at some of the projects currently building zkEVMs in a later section of this report. Before that, we should have a closer look at Vitalik's different types of zkEVMs and explore the differences between zkEVM compatibility vs. equivalence.

Different Types of zkEVM

All zkEVMs are equal, but some zkEVMs are more equal than others. In contrast to George Orwell's "Animal Farm", being different is nothing bad, and - as so often- having different approaches and solutions helps to cover many use cases. Before having a closer look at the different projects building zkEVMs, we should first establish a common understanding of the different "types", as well as the benefits and downsides of each. To differentiate between zkEVMs, we will follow Vitalik's classification.

Figure 9: Different types of zkEVMs and their tradeoffs



❖ Can verify an environment that looks exactly like Ethereum (and the chain itself)
❖ Can scale the Ethereum L1 (not just rollups)
❖ Maximally easy for rollup because you can share infrastructure
❖ Takes a long time to generate proof
1
Fully Ethereum equivalent
❖ Can verify an environment that looks exactly like Ethereum (but with minor changes) that don't touch the application layer
❖ Fully compatible with almost all Ethereum apps
❖ Can share most infrastructure
❖ Takes a long time to generate proof
2
Fully EVM equivalent
2.5
Fully EVM (except gas)
❖ Can verify an environment that is similar to Ethereum, but with minor changes that do touch the application layer
❖ Fully compatible with most Ethereum apps
❖ Can share a lot of infrastructure
❖ Faster to generate proofs
3
Almost EVM equivalent
❖ Modify the EVM only by changing gas costs
❖ Makes it faster to generate proofs
❖ Introduces a few incompatibilities
❖ Compiles contracts written in Solidity, Vyper or other high-level langs to a specialized VM
❖ Not compatible with some Ethereum apps
❖ Can't share a lot of infrastructure
❖ Faster proof generation times
4
High-level-language equivalent
Fast
Compatibility
Slow
Least Secure
Performance
Most Secure

Source: Vitalik Buterin, Binance Research

Type 1 - Fully Ethereum-equivalent

The first type of zkEVMs, according to Vitalik Buterin, is that of Fully Ethereum-equivalent zkEVMs. These zkEVMs aim to be **fully** and **uncompromisingly** Ethereum-equivalent. This means that they don't change any part of the Ethereum system itself. They don't replace hashes. They don't replace state trees, and they don't replace transaction trees or any other in-consensus logic.

The perfect compatibility of this type of zkEVM offers a few advantages that are worth pointing out. According to Vitalik, this type of zkEVM will allow for the most scalability of Ethereum's L1 and provide rollups with the necessary infrastructure. However, due to the high complexity of a fully Ethereum equivalent, zkEVM prover time is extremely high.

Type 2 - Fully EVM-equivalent

Type 2 zkEVMs aim to be EVM-equivalent, but are actually not. This means that while using EVM technology, they have some important differences in terms of data structures (block structure, state tree, etc.). This version of zkEVMs allows almost every Ethereum native application to also work on the zkEVM rollup (with a small number of exceptions). In addition to that, Type 2 zkEVMs are faster than Type 1 zkEVMs, though they come with their own downside. Incompatibilities could arise for applications that verify Merkle proofs and rely on complicated and zk-unfriendly cryptography. A Type 2.5 zkEVM would be fully EVM-equivalent except for gas cost. This might involve precompiles, the KECCAK opcode, and possibly specific patterns of calling contracts or accessing memory or storage, or reverting.

Type 3 - Almost EVM-equivalent

This type of zkEVM is sacrificing equivalence to improve prover times and make EVMs easier to develop. As such, this type of zkEVM might remove a few features that are exceptionally hard to implement in a zkEVM implementation. While the idea generally improves at prover times, it introduces new complexity due to the fact that some applications might need re-writing.

Type 4 - High-level-language equivalent

The last type of zkEVMs allows for smart contract code to be written in a high-level coding language allowing for extremely fast prover times, but also comes with high levels of incompatibility.

Figure 10: Equivalence vs. Compatibility vs. Vitalik's 4 Types

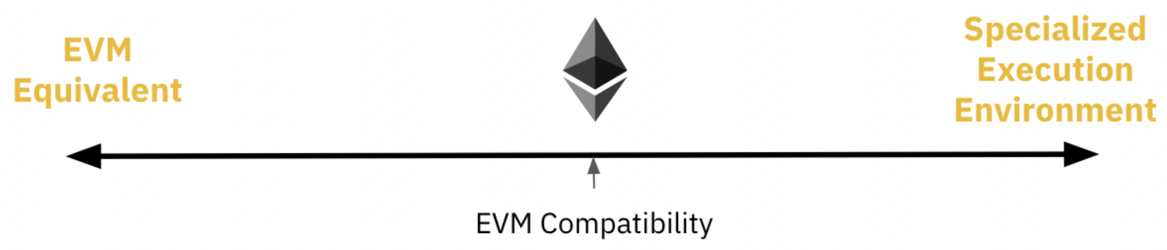
Concept	Proposed Definition	Vitalik's 4 Types
Ethereum Equivalent	100% compliance with the Ethereum specification	Type 1
EVM Equivalent	100% compliance with the EVM specification, proving EVM opcodes directly	Type 2
EVM Compatible	Compile EVM bytecode into the bytecode of your custom VM, then prove the computation in your custom VM	Type 3
Solidity Compatible	Compile Solidity into the bytecode of your custom VM, then prove the computation in your custom VM	Type 4

Source: Binance Research, Alex Connolly

Equivalence vs. Compatibility

Vitalik's 4 Types of zkEVMs differentiate between equivalence and compatibility amongst different zkEVM solutions. We can simplify this, however, and think just about those two elements (rather than 4 and a half). Looking at just equivalence vs. compatibility will allow us to understand the differences better and gain an understanding of why we have two different approaches.

Figure 11: Equivalence vs. Compatibility



Source: Binance Research

In the past most Layer 2 solutions have built their rollups with a focus on EVM compatibility (considering equivalence is actually hard to achieve) though we have seen a general trend towards increased compatibility, moving rollups from more specialized execution environments to more EVM compatible solutions. EVM Compatible solutions don't implement the EVM itself, but instead, implement a compatible subset that is small enough to be able to identify inconsistent state changes. As mentioned above, the advantages of EVM-compatible rollups are far-reaching. Especially in the early days, this approach allowed for a fast solution to scaling without having to implement a complex EVM structure in a smart contract. EVM compatibility reduces the amount of gas required for the fraud-proof verifier contract and execution.

However, smart contracts used by complex applications such as Uniswap were unable to use zkEVMs in the early days due to the complexity of their code. This is why recent changes to the space are bringing more advancements than we were expecting. For a project like Uniswap to settle with a “simple” form of compatibility would mean that they would be forced to modify their lower-level code - code that Ethereum’s supporting infrastructure also relies on.

The general alternative to EVM compatibility that would solve these issues is EVM equivalence. EVM equivalence means that L2 solutions and all of their modules are completely compliant with the original Ethereum Architecture. With this, any bytecode that can be run on the L1 can also be run on the L2 counterpart. At this point, it becomes quite clear how equivalence and compatibility overlap with Vitalik’s classification of zkEVMs.

Ethereum equivalence offers numerous benefits. For example, EVM equivalence allows for the “copy/pasting” of DeFi codes across all EVM equivalent rollups, providing complete alignment with the Ethereum Virtual Machine specification. However, we disagree with David Hoffman that Ethereum compatibility is dead. **We believe that rather than seeing one approach as superior to another, we should instead see this differentiation between equivalence and compatibility as a categorization rather than a hierarchy, as each approach comes with its own benefits and drawbacks.**

Ethereum Compatibility is dead - Long live Ethereum Compatibility

While there is an argument for high levels of equivalence, we can see the same for a more hybrid (compatibility) approach that is used for more use-case-oriented rollups. After all, most applications have to decide between different tradeoffs. Do you want an easier build environment? Easily compatible with existing tools? Improved performance? Or do you want something completely different?

At the end of the day, we’re excited to see how different teams try out different approaches, and even the approach of building non-EVM blockchains that are more scalable due to design differences can be considered here.

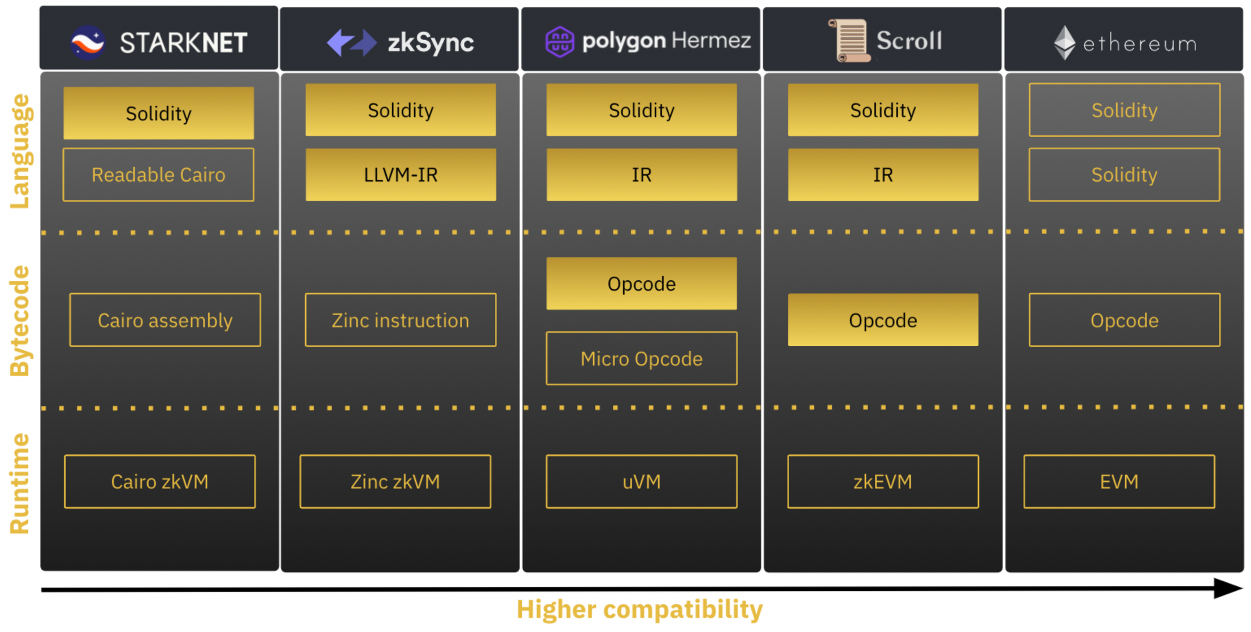
We agree with Vitalik Buterin, that different approaches are not per se "better" or "worse" than others. Rather, they are different points within the tradeoff space. More equivalent approaches are more compatible with existing infrastructure but slower, while more specialized execution environments are less compatible with existing infrastructure but faster. We disagree with him, however, on his view that every zkEVM should become a fully EVM equivalent.

zkEVM Project Overview

There are many projects building zkEVMs - each with its own unique approach. Considering the depth of the space, we will focus only on a few leading projects with promising technologies underlying their Layer 2 zkEVMs. To be more concrete, we will have a closer look at Polygon Hermez, Scroll, zkSync, and StarkNet, as well as zkBNB -a zero-knowledge protocol scaling BNB Chain using SNARKs.

While most solutions are similar at the developer and users level, the infrastructure of each can look quite different. Having a closer look at Figure 12, we can see that the further right an EVM is, the better the compatibility is - yet the slower the development.

Figure 12: Overview of different zkEVMs compared to Ethereum



Source: Binance Research, Immutable X

Scroll - From papyrus to zkEVM

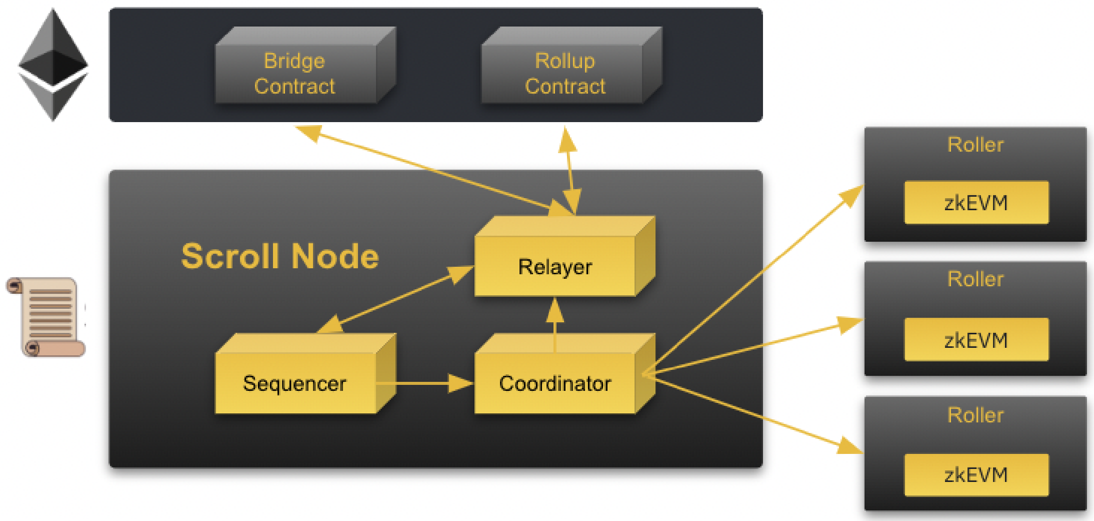
Scroll, considered highly compatible with Ethereum, is a zero-knowledge EVM implementation that is [currently being developed](). The approach that Scroll is taking is to design an "ASIC" circuit for each dApp and let them communicate through cryptographic commitments. By doing this, they allow developers to deploy Ethereum-native smart contracts EVM on Scroll without needing to modify the underlying EVM bytecode.

Having a closer look at the Scroll architecture, we can see three building blocks that stand out.

- **Scroll Node -** Constructs L2 blocks from user transactions and commits them to the Ethereum base layer. It is the main way for applications and users to interact with Scroll and consists of three modules- the Sequencer, Coordinator, and Relayer
- **Roller Network -** Generates a zkEVM validity proof to prove that transactions are executed correctly. They are expected to utilize accelerators such as GPUs, FPGAs, and ASICs to reduce the proving time and proving cost
- **Rollup and Bridge Contracts -** Scroll connects to the base layer of Ethereum through the Rollup and Bridge smart contracts. This provides data availability for Scroll transactions, verifies zkEVM validity proofs, and allows users to move assets between Ethereum and Scroll

Putting these three elements together, we can simplify the Scroll zkEVM Architecture the following way:

Figure 13: Scroll zkEVM Architecture



Source: Binance Research, Scroll

Scroll is able to execute native EVM bytecode on L2 while inheriting strong security guarantees from base layer Ethereum. And it's straightforward, all they need to do is design a circuit for some cryptographic accumulator, link the bytecode with the real execution trace, and design a circuit for each opcode. Jokes aside - this is actually quite difficult, and not "simply" implementing every EVM opcode directly in a circuit makes it clear to us why we're still in the Pre-Alpha Testnet. In many cases - such as this - mirroring the EVM directly introduces massive overhead. As an example: Ethereum's storage layout relies heavily on keccak256, which is around 1000x larger in circuit form than a STARK-friendly hash function. However, replacing keccak will cause huge compatibility problems for existing Ethereum infrastructure.

Scroll's next step is the release of their Alpha Testnet, which will bring along a fully permissionless testnet open to all users without whitelisting. In addition to that, we will likely see throughput improvement with increased compute and batch block verification.

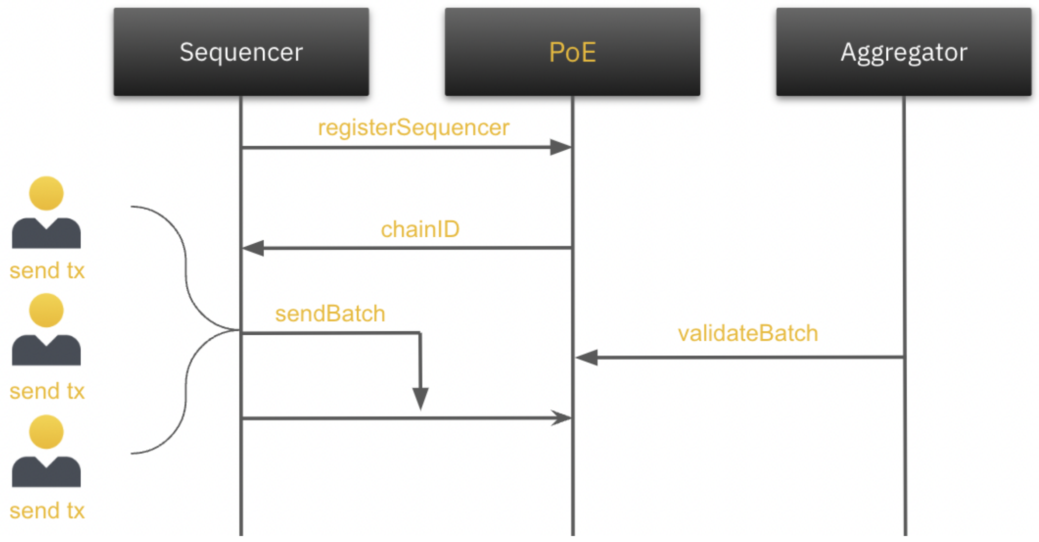
Polygon Hermez - Ready, Set, Go

Going from the right to the left on Figure 12, let us look at Polygon Hermez next. Polygon has many different scaling solutions under its umbrella, with Hermez currently building a zkEVM that recently launched its testnet. Hermez is a decentralized zk-rollup for the Ethereum mainnet working on a zero-knowledge Ethereum Virtual Machine (zkEVM) that executes Ethereum transactions in a transparent way, including smart contracts with zero-knowledge-proof validations.

An early version of Polygon Hermez (Hermez 1.0) was based on a so-called Proof of Donation ("PoD") consensus mechanism. This changes with Hermez 2.0, which is currently using Proof of Efficiency ("PoE"). The Proof of Efficiency model leverages the existing Proof of Donation mechanism but also supports a permissionless participation of multiple coordinators to produce batches in Layer 2. In the new model, the creation of batches consists of a two-step model that splits activities between different parties. The first party is the Sequencer and the second one is the Aggregator.

The PoE smart contract makes two basic calls: A call to receive batches from Sequencers, and another call to Aggregators, requesting batches to be validated.

Figure 14: Simplified version of Proof of Efficiency



Source: Binance Research, Polygon Hermez

The Polygon Hermez version of zkEVM uses a zero-knowledge prover (“zkProver”), which is intended to run on any server and is being engineered to be compatible with most hardware. Every Aggregator will use this zkProver to validate batches and provide validity proofs.

Hermez’s virtual machine uses both SNARK and STARK proofs to verify the correctness of program execution. Zero-knowledge tools such as STARKs for proving purposes are very fast though they are bigger in size. So, instead of publishing the sizeable STARK proofs as validity proofs, a zk-SNARK is used to attest to the correctness of the zk-STARK proofs. These zk-SNARKs are, in turn, published as the validity proofs to state changes. This helps in reducing gas costs from 5M to 350K, according to Polygon.

Polygon’s zk-rollups are designed to shrink the computational cost of creating proofs, which puts them in a competitive position. However, Polygon Hermez is still lacking full equivalence at this point (Figure 15) and is in competition with some other innovative companies.

Figure 15: Polygon compatibility test

Total	Generation errors	Ignored	Compatible	Not compatible	Coverage
13282	185	3173	9799	125	97%

Source: Binance Research, Polygon Hermez

zkSync - Evolving from 1.0 to 2.0

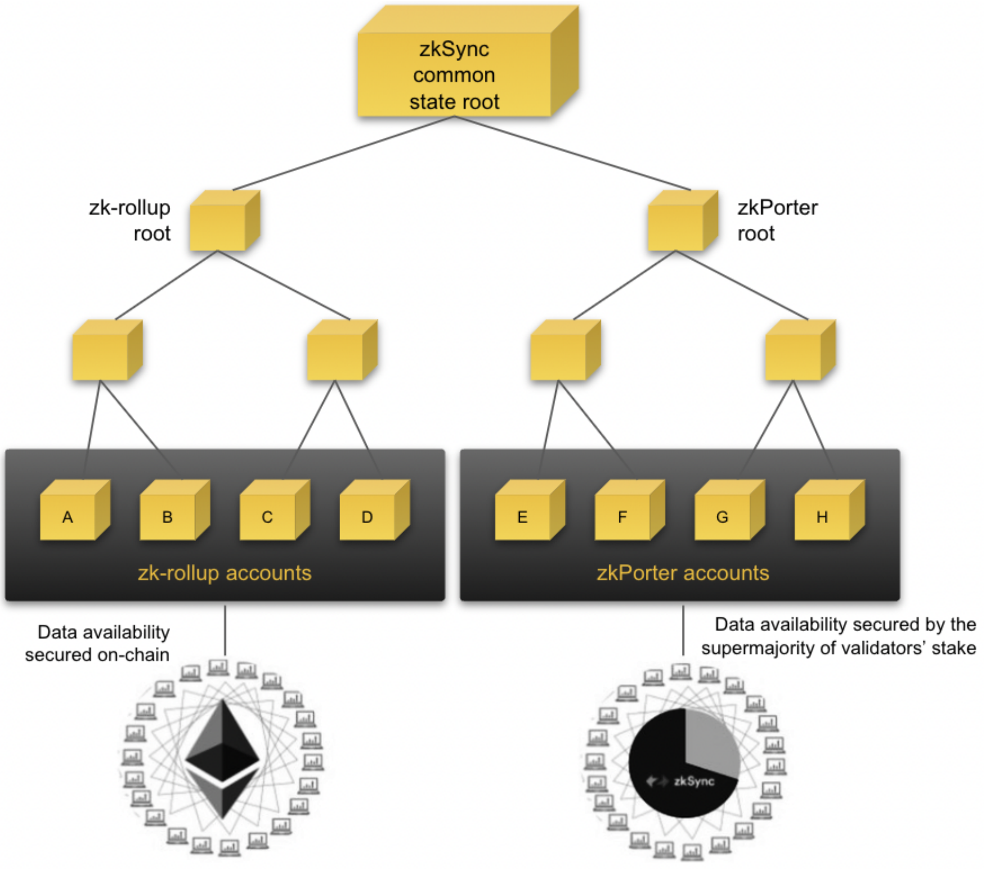
Goodbye zkSync 1.0, hello zkSync 2.0. zkSync 2.0 is an EVM-compatible zk-rollup being built by Matter Labs, powered by its own zkEVM. The current version of zkEVM includes multiple features that are worth pointing out.

- **Native support of ECDSA signatures -** Any account can be managed in L2 with the same private key that is used for L1
- **Solidity and Vyper support -** Ability to deploy existing codebase with little to no changes required (Support for Solidity 0.4.11 onwards and Full support for Vyper 0.3.3)
- **Web3 API -** zkSync API is almost fully compatible with Ethereum, allowing seamless integration with existing indexers, explorers, etc.
- **Support for Ethereum cryptographic primitives -** zkSync natively supports keccak256, sha256, and ecrecover via precompiles
- **Hardhat plugin -** Allowing easy testing and development of smart contracts on zkSync
- **Account abstraction -** The decoupling of the object holding tokens (the account) from the object authorized to move tokens (the signer) allowing for custom logic for signature checking for accounts and other features

Another feature recently announced by zkSync is zkPorter. This will allow users to choose between a zkRollup account featuring high security and an estimated 20x fee reduction compared to Ethereum, or a zkPorter account featuring stable transaction fees of just a few cents in a different security model. Both zkPorter accounts, and zkRollup accounts, will be able to interact seamlessly together under the hood.

As you can see, zkSync 2.0 will have two core features worth pointing out - a zk-rollup and zkPorter. The zk-rollup will allow for on-chain data availability, while zkPorter will bring off-chain data availability. As such, the hope is that zkPorter fills the gap between zkSync and scalability when it comes to mainstream/mass adoption.

Figure 16: zkSync 2.0 design (including zkPorter)


zkSync common state root
zk-rollup root
zkPorter root
A
B
C
D
E
F
G
H
zk-rollup accounts
zkPorter accounts
Data availability secured on-chain
Data availability secured by the supermajority of validators' stake
zkSync

Source: Binance Research, Matter Labs

Looking closer at zkPorter, we can note that the data availability of zkPorter accounts will be secured (in the future) by zkSync token holders. They will keep track of the state on the zkPorter side by signing blocks to confirm the data availability of zkPorter accounts.

While zkSync originally created its own CAIRO-like language, called Zinc, we have seen them pivot away from it, instead focusing on the Solidity compiler, which allows for a simpler migration for L1 developers. Compared to competitors, we can see the reuse of more of the Ethereum toolset, making them increasingly Ethereum-compatible. In addition to that, zkSync takes advantage of its custom VM to deliver additional (non-EVM-compatible) features like Account Abstraction, which has long been a goal of the core Ethereum protocol - making zkSync one of the most interesting projects aiming to scale Ethereum at the moment.

StarkNet - The Iron Man of zkEVMs

StarkWare (the company behind StarkNet) has taken the probably most specialized approach in terms of zkEVMs from the projects presented here. As the name suggests, StarkNet's main aim is to build STARK-based solutions for the blockchain industry with security, trustlessness, and scalability as its foundational pillars. In contrast to Hermez, Scroll, and zkSync, StarkNet runs a custom smart contract virtual machine called "CAIRO VM". This VM comes with its own low-level language (Cairo). As such, StarkNet has lower compatibility with Ethereum.

With regards to StarkNet, all transactions on StarkNet are periodically batched, and a STARK proof is used to prove its validity on the Ethereum network. The decentralization of the network includes two aspects worth being aware of:

- The permissionless layer of Sequencers and Provers ensures that the network will be censorship-resistant
- The usage of STARK-proofs ensures that everyone will be able to verify the full StarkNet chain with low hardware requirements, regardless of the network's throughput, and without trusting any external entity

In addition to that, the teams at Nethermind and StarkWare have created a Warp transpiler, which is capable of transforming arbitrary Solidity code into Cairo VM bytecode, allowing for Solidity contracts to be ported to StarkNet. While this allows for some form of compatibility, there are several tradeoffs to be aware of - A more custom rollup will allow for better performance, but you will lose the benefit of the collective improvements made to the EVM by every other chain and rollup. In addition to that, if developers are writing contracts in both Cairo and Solidity, there is a chance that the tooling to support the interface between both will be negatively impacted.

The StarkNet Alpha version is currently preparing for "Regenesis" - an update that will expand the use of Cairo 1.0 across all of StarkNet. With it, StarkNet will start from a new genesis block with the existing state. By doing so, the company can capitalize on the new improvements that came with Cairo 1.0 (sequencers DOS protection, censorship resistance, gas metering - to name a few). In addition, StarkNet will be able to include reverted transactions in blocks and generate proof of their execution - driving further DOS protection. After Regenesis, the StarkNet system will be fully based on Cairo 1.0.

Figure 17: Core features of StarkNet

Availability	
Smart contracts support general Computation & Composability	Yes
L1<>L2 Communication & Security through on-chain data	Yes
Solidity to Cairo Compiler	Yes
StarkNet Full Node	Yes
StarkGate Alpha: StarkNet Bridge	Yes
Range of Data Availability Solutions	Coming soon
Permissionless Sequencer and Prover	Coming soon

Source: Binance Research, StarkWare

zkBNB - The Future of BNB Chain?

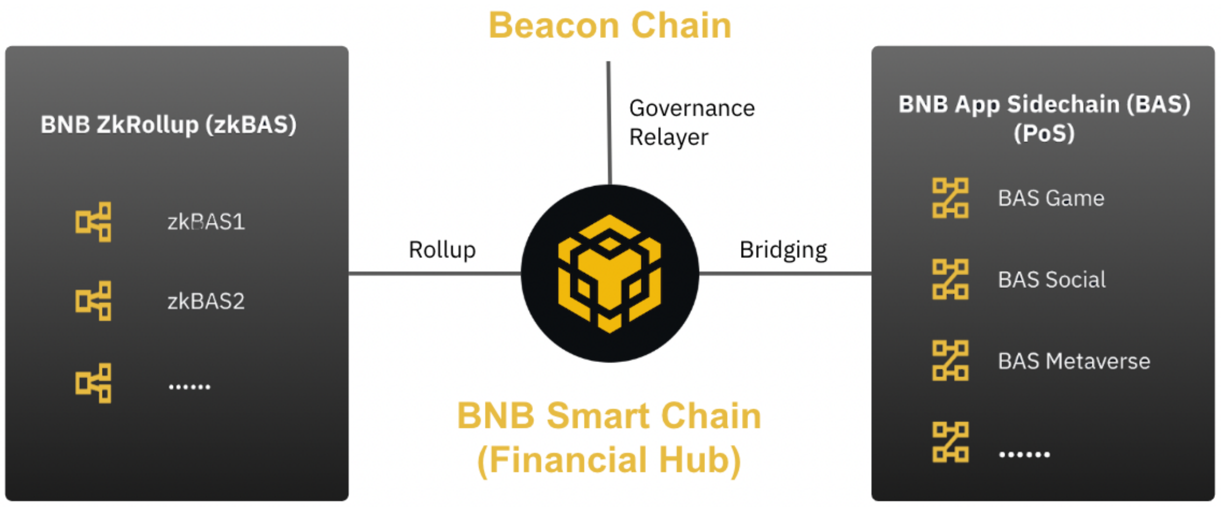
Ethereum is not the only chain that is scaling with the help of SNARKs. Leveraging the power of zk-rollups, BNB Smart Chain has introduced zkBNB, bringing further scalability to the blockchain. In line with other zk-rollups, zkBNB has the same capability to bundle hundreds of transactions into a single batch off-chain and generate a cryptographic proof. These proofs can come in the form of a succinct non-interactive argument of knowledge ("SNARK"), which can prove the validity of every single transaction in the rollup block. SNARK ensures that all funds are held on the BNB Chain while computation and storage are performed on BNB Sidechains. Furthermore, thanks to the use of zk-SNARK proofs, zkBNB shares the same security as that of BNB Smart Chain.

Thanks to the use of zk-SNARK proofs, zkBNB shares the same security as that of BNB Smart Chain

Currently, ZkBNB implements the following features:

- **Same Security as that of L1 -** The zkBNB shares the same security as BSC does. Due to the use of zkSNARK proofs, the security is guaranteed cryptographically
- **Seamless L1-L2 Communication -** BNB and BEP20/BEP721/BEP1155 tokens created on BSC or zkBNB can flow freely between BSC and zkBNB
- **Built-in instant AMM (Automated Market Maker) swap -** zkBNB allows digital assets to be traded without permission and automatically by using built-in liquidity pools
- **Built-in NFT marketplace -** Developers can build marketplaces for crypto collectibles and NFTs (non-fungible tokens) out of the box on zkBNB
- **Fast transaction speed and faster finality -** With performance a key priority for BNB Smart Chain, zkBNB puts up astonishing figures with an ability to support 100 million addresses and handle up to 10 thousand TPS
- **Gas Tokens -** The gas token on the zkBNB can be either BEP20 or BNB, with fees up to 10x lower
- **"Full exit" on BSC -** If a user feels that his transactions are censored by zkBNB, at any time, they can request a "full exit" operation to withdraw funds. This means users can withdraw funds at any time

Figure 18: zkBNB Design



Source: Binance Research, BNB Chain

As zkBNB offers straightforward token operations out-of-the-box, developers can now efficiently transfer BNB and other digital tokens (BEP20/BEP721/BEP1155) seamlessly between BSC and zkBNB. Resulting in faster execution of lengthy transaction lists while ensuring a seamless undisturbed experience.

With the release of zkBNB Testnet in November, the Mainnet is targeted to launch in Q1 of 2023. More about BNB Smart Chain's innovative projects can be looked at on BNB Chain's 2022 roadmap.

The Layer 3 Narrative

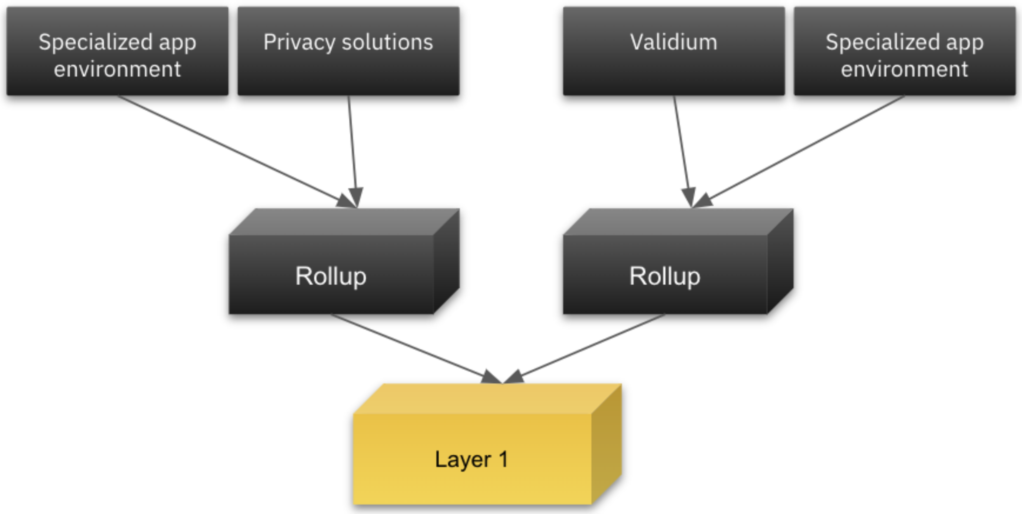
Okay, so we talked a lot about layer 2 scaling and zk-rollups in particular. We have seen how powerful zkEVMs can be and how promising their development seems for future Ethereum scaling. However, we have yet to touch on Layer 3 scaling.

The idea is straightforward - If we can build a layer 2 protocol that scales a layer 1 platform (yet relying on the L1 for security), then we can surely scale our layer 2 further by building a layer 3 protocol that anchors into layer 2 for security. While this sounds quite promising in theory, it is not as straightforward in practice. Considering SNARKs and STARKs and their design, just putting different layers on top of each other for further scaling doesn't make much sense. StarkWare - yes, the company developing StarkNet - proposed a sophisticated framework in which layers aren't just stacked on top of each other but assigned different purposes - In its article, StarkWare points out three possible designs for a world with Layer 3 scaling.

- **L2 for scaling, and L3 for customized functionality -** In this vision, there is no attempt to provide further scalability. Instead, one layer of the stack helps applications scale, and then separate layers serve a customized functionality (e.g., security)
- **L2 for general-purpose scaling, L3 for customized scaling -** Customized scaling might come in different forms: specialized applications that use something other than the EVM to do their computation, rollups whose data compression is optimized around data formats for specific applications
- **L2 for rollups, L3 for validium -** Validiums are systems that use SNARKs to verify computation but leave data availability up to a trusted third party or committee. Validiums have a lower grade of security than rollups, but can be vastly cheaper.

The key argument is that a three-layer model allows for sub-ecosystems to exist within a single rollup, which in return allows cross-domain operations within that ecosystem to happen very cheaply, without needing to go through the expensive layer 1. A clear benefit of L3 scaling is better control by the app designer of the technology stack.

Figure 19: Layer 3 design possibility



Source: Binance Research, Vitalik Buterin

L3 promises hyper scalability, better control of the technology stack for various needs, and privacy, while maintaining the security guarantees provided by Ethereum.

The recursive concept it employs may be extended to additional layers for fractal layering solutions. Layer 3s are still a vision as of now, and getting Layer 2 scaling right should be a priority in the near term.

While still a vision, StarkWare isn't the only application thinking about Layer 3 scaling. zkSync, announced this month that it is aiming to deploy a new EVM-compatible layer-3 prototype called "Opportunity" on testnet early next year. zkSync envisions Layer 3s as trustless, customizable blockchain ecosystems powered by zkEVMs (such as their own HyperChains). In this vision, zkSync's layer 2 will provide the foundation necessary for Ethereum to scale without compromising its security or decentralization. zkSync's Layer 3 solution allows developers to choose from three data availability options, all using the same infrastructure for their project. The choice between price, performance, and security is up to the developer.

Conclusion

Concluding this report isn't straightforward, considering the multiple topics and projects we have touched on. At the end of the day, zkEVMs are one way to scale Ethereum, but they have tough competition with Optimistic rollups since they had time to establish themselves already. That being said, zk-rollups (especially combined with EVM compatibility) offer a lot of upsides to the Ethereum mainnet.

STARKs and SNARKs both have their own benefits, and while we see a higher adoption rate of SNARKs, this might change in the future, considering the technical benefits that STARKs bring. In a world of high crypto adoption, STARKs offer a clear edge with higher scalability and more transparency. That being said, as of now, SNARKs and recursive SNARKs, in particular, seem to be a good solution that can be built on further with the idea of Layer 3s. However, as so often, we should not stay too fixated on one type of solution, and STARKs might well be the solution to future developments of Ethereum.

STARKs are currently used for more specialized execution environments. Entering the debate if it is the holy grail to be fully Ethereum equivalent, simply compatible, or extremely specialized, we reject the narrative of some zk-rollups and influencers in the space that the more equivalent to Ethereum a solution is, the better it is. There are many different use cases and in some, a more specialized execution environment with low Ethereum equivalence makes sense from a technical perspective. In others, you want to make the transition from Ethereum to the Layer 2 as easy as possible - something that might already be achieved with high levels of compatibility. Not being fully equivalent doesn't mean that it's not already "easy" to do so since most opcodes are supported by projects like Hermez and zkSync. Another aspect to consider is that (at the moment) higher compatibility means slower execution times, while less compatible projects oftentimes benefit from faster execution times due to their improved infrastructure. We welcome both types of solutions and hope for developments along the spectrum rather than a polarized world of zkEVMs.

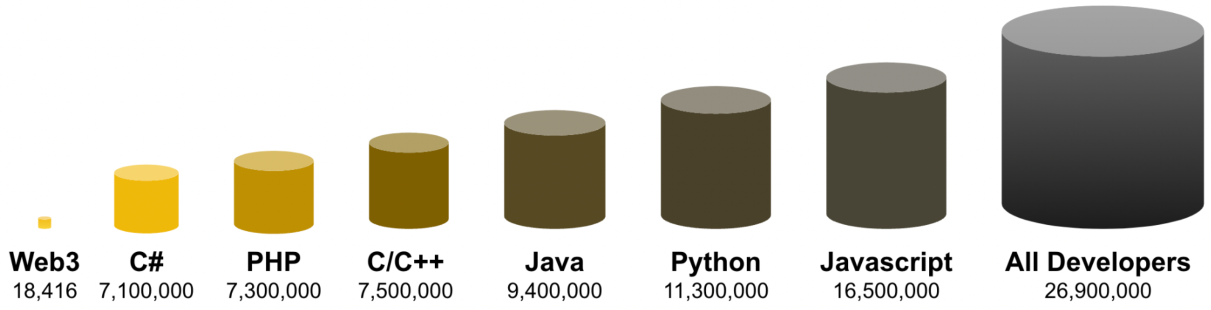
Considering the different types of compatibility with Ethereum, we further hope to see efforts from the Ethereum community to also make the mainnet overall more zero-knowledge friendly. Full adoption (and even the vision of Layer 3s) is something we're looking forward to over the coming years, and we expect multiple paths to scaling Ethereum based on zk-rollups to gain traction.

The benefits of doing so seem clear. Zero-knowledge proofs have brought clear benefits to privacy for multiple years already, and further evolution of the technology can help to foster a secure environment for people to operate in. The benefits don't end there, though. The same technology can also help to add multiple layers of security to, storage, files, and logins - increasing the number of obstacles hackers have to overcome to retrieve data. Organizations with sensitive data, such as hospitals or financial institutions, would see clear benefits from using zero-knowledge technology, and together with the use of blockchains accessing data can be made a lot safer.

There is another aspect worth mentioning, and that is the coding complexity of zkEVMs (and VMs) at the moment. Having to switch from Solidity to Cairo, Go, or Rust is not done overnight. As such, improving the ease of coding compatibility within Web3 is a great step forward that we welcome. In a perfect world, we would see WASM or RISC-V being built using C++, Solidity, Rust, or other coding languages. While we're not living in a perfect world yet, we might, however, in the future (at least in terms of Web3 coding).

As of now, the space is still extremely small, with only 0.07% (Figure 20) of developers being active in the Web3 space. Now imagine how many of these developers actually code in Cairo. Not many. A perfect zkEVM, in our view, would make it possible to reach a higher number of developers.

Figure 20: Web3 Developers vs. other coding languages



Source: Electric Capital, Binance Research

If we want to see higher adoption rates of Web3, we have to start thinking more about the developers (We get it, users are important - but they're just one part of the equation). zkEVMs with high compatibility already solve a great deal of problems, but as mentioned above, this is only one step in the right direction.

Closing Thoughts

Going forward, we might see further developments in the space with native zkEVMs competing for TVL in the short term but building the future of blockchains in the long term. At the same time, we will likely see more zkVMs, which can help develop the overall Web3 space as a whole, tackling unique problems with specialized solutions.

About Binance Research

Binance Research is the research arm of Binance, the world's leading cryptocurrency exchange. The team is committed to delivering objective, independent, and comprehensive analysis and aims to be the thought leader in the crypto space. Our analysts publish insightful thought pieces regularly on topics related but not limited to, the crypto ecosystem, blockchain technologies, and the latest market themes.

Stefan Piech, Macro Researcher

Stefan is currently working for Binance as Macro Researcher. Prior to joining Binance, he worked as an Equity Portfolio Manager at Cape Capital, a Swiss Family Office, and as an Equity Research Analyst for BlackRock's European and UK Hedge Fund. He has prior experience in both Private Equity and Venture Capital. Stefan started his career as Government Official for the District Government Muenster. Stefan has been involved in Crypto since 2019.

Read more
https://research.binance.com/en/analysis

Share your feedback
https://tinyurl.com/bnresearchfeedback