From Zero to zkEVM - L2 Scaling Solutions

v.1.0

December 11, 2023

1 Introduction to Layer 2

L2 layers are be established on top of the L1 infrastructure. These L2 layers have the authority to dictate their own state management procedure. For example, a payment system with simple transactions, a token transfer system or a EVM execution capability. Moreover, L2 layers have the autonomy to define their utilization of the underlying L1 architecture, encompassing two pivotal aspects: the L1 execution layers usage and the interaction with the data shards (when available). Regarding the current situation of data sharding, L1 data shards specification is still under develop (the **EIP 4484** is currently on the "review stage"). For this reason, currently, all the scaling solutions use only the availability of the execution layer.

For the sake of illustration, let's consider an instance of an L2 layer denoted by $L2_x$. In this scenario, $L2_x$ **transactions** form the foundational operations within $L2_x$. These transactions shape the state progression of $L2_x$. Figure 1 illustrates how the different states of $L2_x$ change with the execution of new sets of transactions.

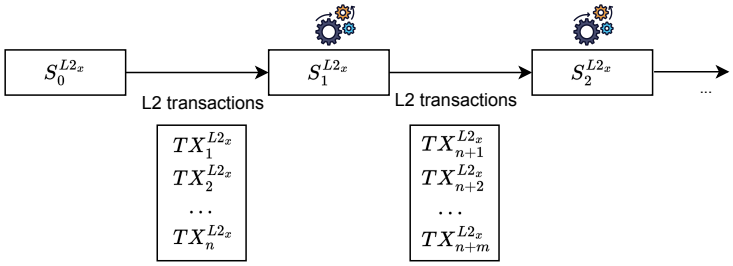

Figure 1: L2 State evolution via the execution of sets of L2 transactions.

Within this context, several questions naturally emerge:

(a) **Transaction Dissemination**: *How do users initiate and direct $L2_x$ transactions, and who are the receivers of these transactions?*

(b) **Transaction Data Availability**: *Is there a mechanism in place to make L2 transactions publicly accessible, and if so, what methods are employed to achieve this transparency within the $L2_x$ ecosystem?*

(c) **Transaction Processing and State Verification**: *Who bears the responsibility for processing L2 transactions, what methodologies govern this processing, and when is it publicly affirmed that a new state has been accurately computed within $L2_x$?*

(d) **Supported Application Complexity**: *What types of applications find their home within $L2_x$? Does this layer cater primarily to straightforward, basic processing, or does it have the capacity to facilitate more intricate operations like EVM-like opcodes?*

While it's important to note that the answers may not be unique due to the existence of different approaches (each one having its pros and cons), the following points will endeavor to provide responses to all of the questions posed.

Transaction Reception

In the context of transaction origination and reception, there exists various communication paradigms that may dictate how L2 transactions are disseminated:

- **Unicast**: This dissemination mode involves the creation of a one-way communication channel with a centralized receiver (See Figure 2). In this scenario, L2 transactions are directed towards a specific centralized recipient.

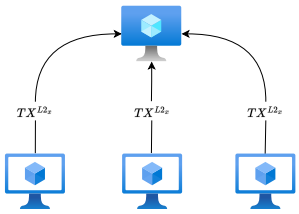

Figure 2: Schema of Unicast communication.

- **Peer-to-peer**: In a peer-to-peer network (See Figure 3), every participant, in a decentralized fashion, receives and processes L2 transactions. This approach emphasizes decentralized, distributed communication, ensuring that all network participants have access to the transactions.

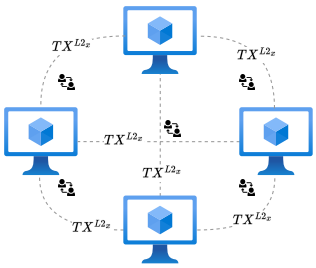

Figure 3: Schema of Peer-to-Peer communication.

- **Smart Contract**: Another approach to dissemination is through smart contracts within the L1 execution layer (See Figure 4). These smart contracts operate in a decentralized manner and play a role in managing and processing L2 transactions.

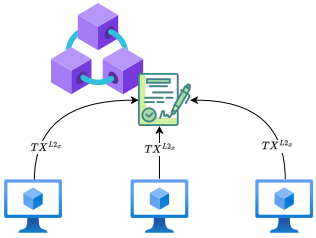

Figure 4: Schema of Smart Contract like communication.

Transaction Data Availability

To achieve L2 data availability in Ethereum, currently, we can proceed in two ways:

- **Validium Approach**: In this approach (Figure 5), L1 transactions only contain a cryptographic summary (a hash) of the corresponding L2 transactions. This design does not allow a user to retrieve L2 data from another layer, such as the L1 top layer. Instead, L2 data is overseen and managed by a consortium of trusted entities known as *data managers*. Hence, users should rely on data managers to access L2 data.

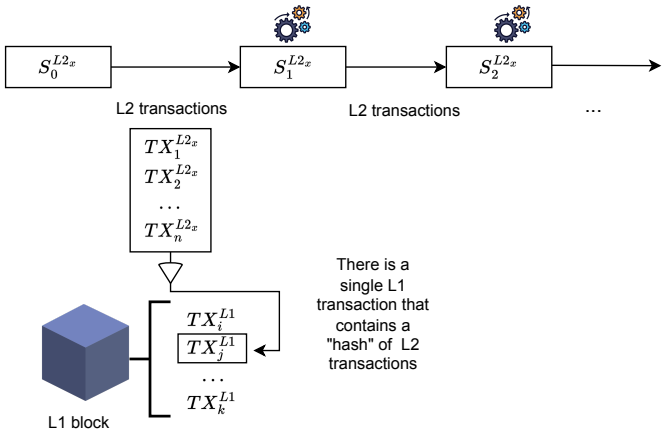

Figure 5: Data Availability with the Validium Approach.

- **Rollup Approach**: Alternatively, a **rollup** mechanism can be employed. This method involves writing L2 data to the public L1 execution layer, making the posted data publicly accessible to all participants on the blockchain network (See Figure 6). In this case, to provide data availability, we use a single L1 transaction that contains a batch of L2 transactions. This method is called a rollup, because we *roll up* a bunch of L2 transactions in a single L1 transaction.

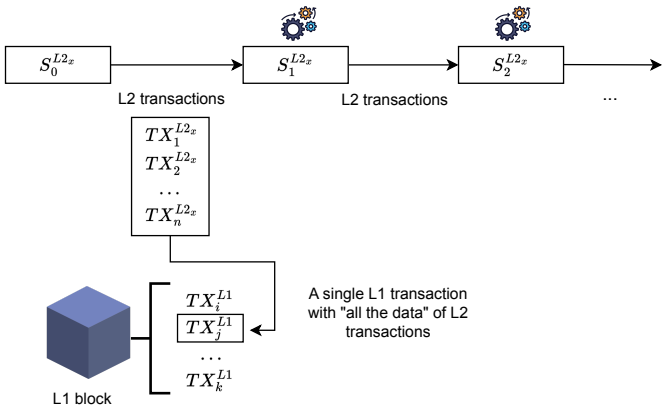

Figure 6: Data Availability with the Rollup Approach.

Transaction Processing and State Verification

In the context of computing the subsequent L2 state $S_{i+1}^{L2_x}$ based on a set of $L2_x$ transactions and the present state $S_i^{L2_x}$, several distinct approaches can be employed. These approaches encompass:

- **Centralized Execution**: One approach involves centralized execution (See Figure 7), where a central entity is responsible for processing L2 transactions and computing the new L2 state. In this approach, the process of finalizing state computation occurs quite rapidly, typically taking only a matter of "seconds or minutes". However, this approach introduces a significant challenge: *how external entities can effectively raise disputes with the L2 operator regarding the accuracy and correctness of an L2 state computation.*

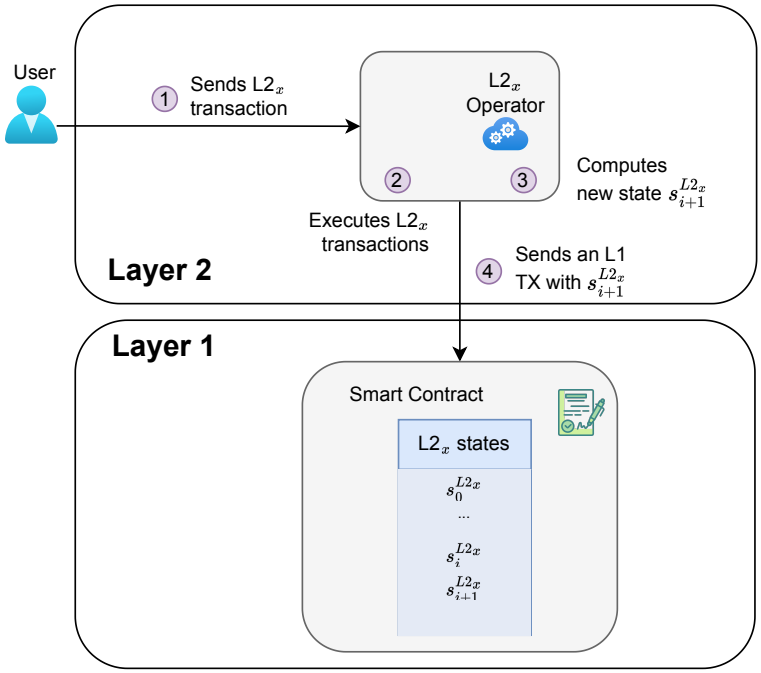

Figure 7: Centralized Execution Schema.

- **Optimistic Execution**: An alternative method is optimistic execution, characterized by a trust-first approach. In this scenario, L2 transactions are executed *optimistically*, that is, assuming they are valid, and only in cases of disputes the transactions are rigorously verified. An Optimistic L2 provides a decentralized execution mechanism that allows disputing the correctness of the L2 state computation. With this approach, there is a (large) period of time to allow anybody to send a **fraud proof** (See Figure 8), proving that the state was wrongly computed (for example, a double spending transaction). Under the this model, if those who execute L2 transactions (referred to as the **optimistic executors**) correctly perform their duties, they stand to earn ETH as a reward for their correct execution. However, in cases where an optimistic executor fails to execute transactions correctly, they face a penalty in the form of slashing and the actor that provides the fraud proof (referred to as the **state checker**) is rewarded for that task. This approach aims to strike a balance between efficiency and security.

- **Succinct Computation Verification (Zero-Knowledge Technology)**: The third approach leverages zero-knowledge technology to enable succinct computation verification. In this method, transactions are processed and verified in a manner that allows third parties to efficiently confirm the validity of computations without needing to replicate them. In the succinct execution verification model, instead of an optimistic executor, we have an execution prover (see Figure 9).

  The **prover** is the entity responsible for generating a proof the new L2 state is correctly computed from the previous state and a given set of L2 transactions (which is called a **batch**) by using zero-knowledge techniques. The **verifier** role, which have to confirm the validity of the proofs, is assumed by a smart contract located in the the Layer 1 that is also responsible for storing the set of states. The proof size is exceptionally succinct, consisting of just a few bytes. This compactness reduces data storage and transmission requirements. Verification time is notably brief, typically

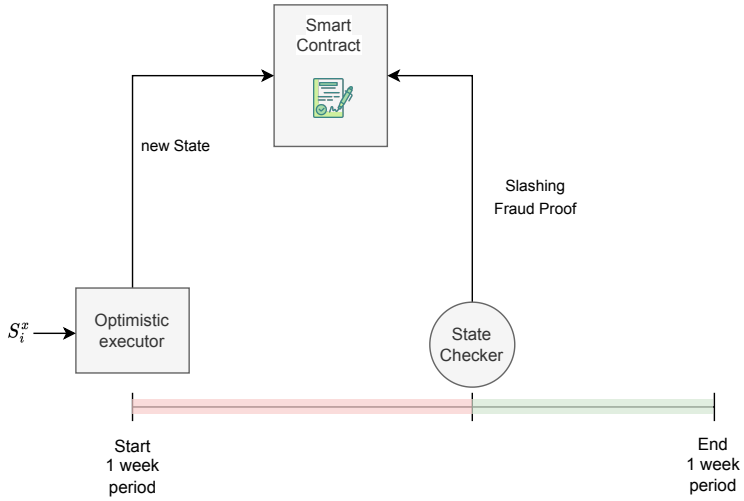

Figure 8: Optimistic Execution Schema of a Fraud Proof.

of the order of microseconds (μs). This inherent properties of the proof enable a smart contract to perform successful verification.

When the verifier smart contract executes the L1 `verifyBatch()` transaction, we say that the next state, S_{i+1}^x, is **consolidated**. With this approach, the state computation is quickly considered final since it takes only "seconds or minutes to generate and validate the proof, offering enhanced privacy and scalability.

The **prover** is typically hosted on a cloud service, allowing for a elastic demand of resources and therefore making it easily accessible and scalable. Provers require a significant amount of computational resources, primarily in terms of RAM and CPU capacity, to generate cryptographic proofs.

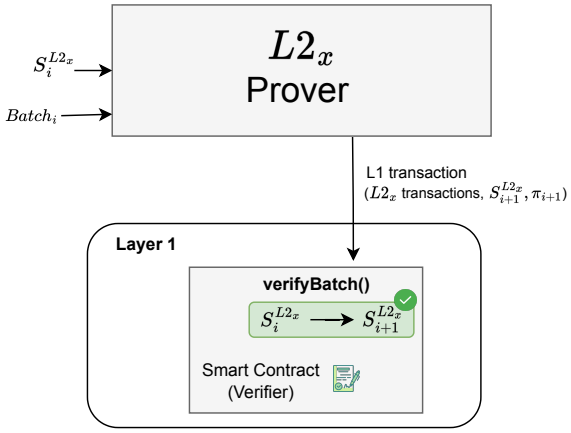

Figure 9: Succinct Computation Verification Schema.

However, since all the L2 data is somehow posted in the L1, the natural question that if we are scaling with rollups based on succinct verification arises. The answer is **yes**, because the smart contract execution resources for verifying a proof are much lower than executing individual transactions within a batch. In fact, the majority of the cost comes from the data availability, but we can work on improve its costs (using succinct verification of compressed data or proto-danksharding when available).

It's important to recognize that the choice of method can significantly impact the performance, security, and decentralization characteristics of the L2 ecosystem. Hence, careful consideration and alignment with specific use cases are essential when selecting the appropriate approach.

Summary of L2 Scaling Solutions

Scalability solutions for blockchains can be systematically categorized along two main dimensions:

- **Data Availability**: Whether the data from the L2 chain is available on-chain (in L1) or off-chain.
- **Mechanism to state correctness**: The methods employed to guarantee the correctness and validity of the L2 chain's state.

The table below offer a clear and concise overview of how various solutions align with the criteria of data availability and the mechanism for state correctness.

	Validity Proof	Fraud Proof
Data on-chain	zkRollup	Optimistic Rollup
Data off-chain	zkValidium	Plasma

One noteworthy feature of the zkValidium approach is its reliance on Zero-Knowledge processing to ensure that even if L2 operators withhold data from users, the cryptographic validation mechanisms guarantee the correct processing of transactions (See Figure 10). This means that, despite the lack of visibility into individual L2 transactions users can be guaranteed about correctness of the current L2 state. This approach is more cost-effective compared to the zkRollup approach, that consists on writing data directly to the L1 blockchain.

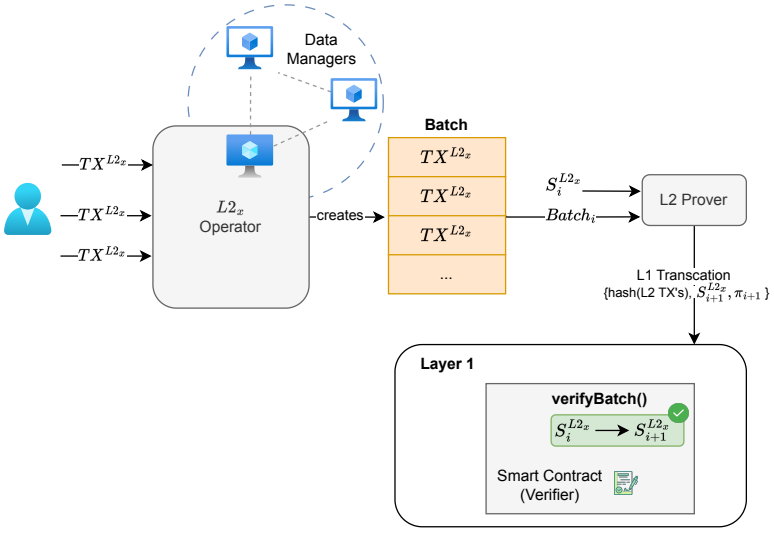

Figure 10: zkValidium Batch Verification Process.

Supported Application Complexity

Within Layer 2 solutions, the scope of supported functionalities can be categorized into two primary types:

- **Simple Processing** (Figure 11): This category encompasses solutions that involve the simple transfer of digital assets, such as tokens or payments, with relatively straightforward processing requirements.

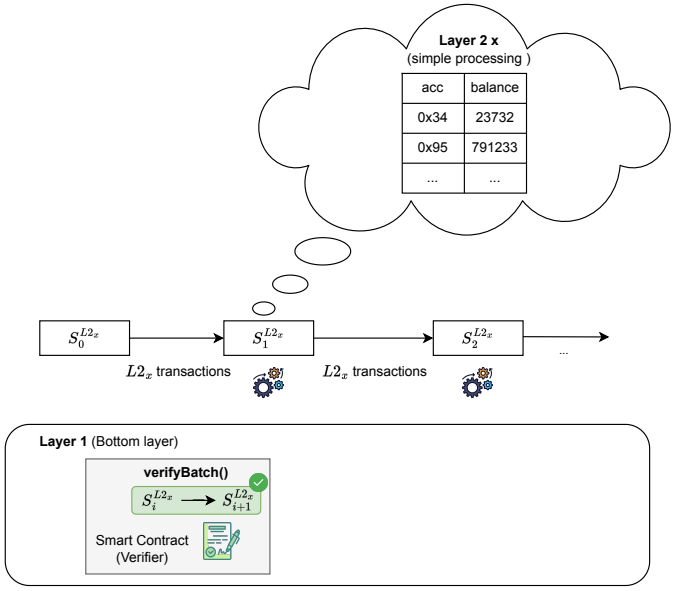

Figure 11: Simple Processing Schema.

- **General-Purpose Execution Virtual Machine** (Figure 12): This category extends the capabilities of L2 solutions to encompass a broader spectrum of functionalities. The hallmark of this category is the utilization of a General-Purpose Virtual Machine, such as the Ethereum Virtual Machine (EVM), enabling “rich processing" of transactions.

Answering the L2 Design Questions for the Polygon zkEVM

- **How users send L2 transactions and who receives them?**

  The zkEVM uses unicast to let user send their transaction (via RPC calls). The zkEVM also enables posting L2 transactions via a method in a smart contract as an anti-censorship measure (called “forced batches).

- **How L2 transactions are made publicly available (if so)?**

  The zkEVM is a rollup, the L2 data is available in L1.

- **Who process the L2 transactions and how, and, when it is publicly considered that a new state is correctly computed?**

  In the zkEVM, currently, there is a centralized aggregator node that proves the processing of the L2 transactions. However, this node cannot cheat because there is a succinct computation verification (using zero-knowledge technology).

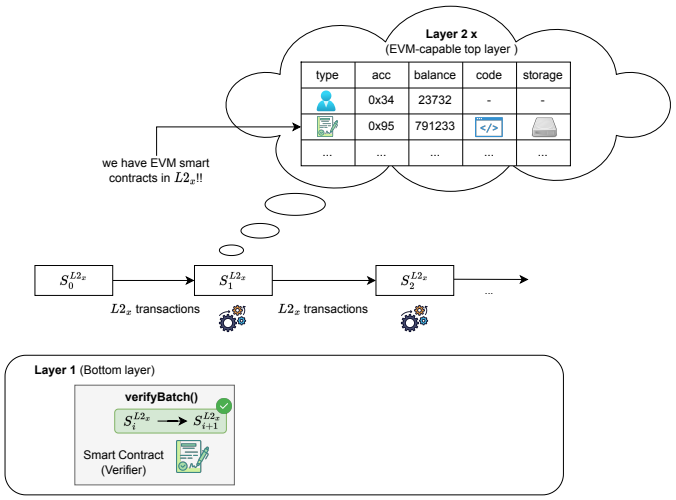

Figure 12: EVM-like Processing Schema.

- **What type of applications the L2 supports? The processing is simple or rich?**

  zkEVM is rich processing since it is an EVM.