Intmax2

A Secure Blockchain Protocol Enabled by Lattice-Ordered Abelian Groups

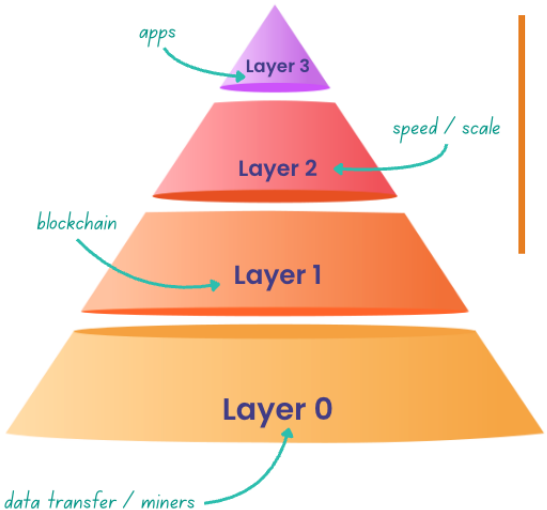
Denisa Diaconescu,

E. Rybakken, L. Hioki, M. Yaksetig, F. Silvasi, and J. Sutherland

TACL 2026 · Kraków, Poland | 27–31 July 2026

01 A bit of context

Architecture of Blockchains



Layer 2

- build on top of Layer 1
- scalability
- process transactions off-chain and settle them on-chain

INTMAX2

1. **Scaling solution for payments**
2. **Stateless and permissionless block production**
3. **Privacy using zk-proofs**
4. **Censorship resistant (modulo L1)**
5. **Upper limit of 7500 txs/s**
6. **Each tx can transfer unlimited number of tokens to an unlimited number of recipients**

02 Overview of the Protocol

User’s actions

1. **Deposit funds in the rollup**
2. **Transfer funds within the rollup**
3. **Withdraw funds from the rollup**

The Rollup Contract

Keeps track of the rollup state.

Manages deposits, transfers, and withdrawals.

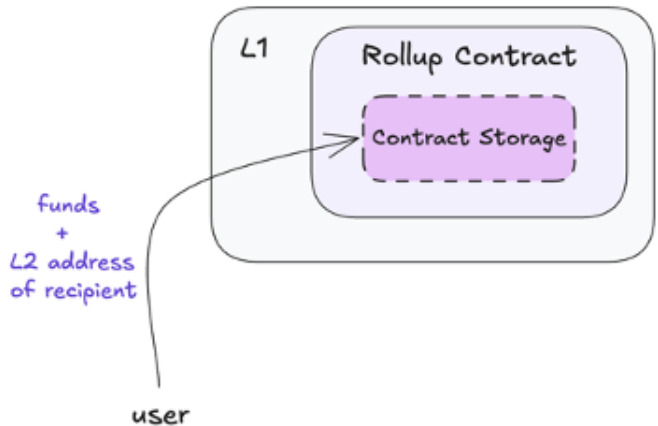
There are three types of "blocks":

$$\mathcal{B} := \mathcal{B}_{deposit} \coprod \mathcal{B}_{transfer} \coprod \mathcal{B}_{withdrawal}$$

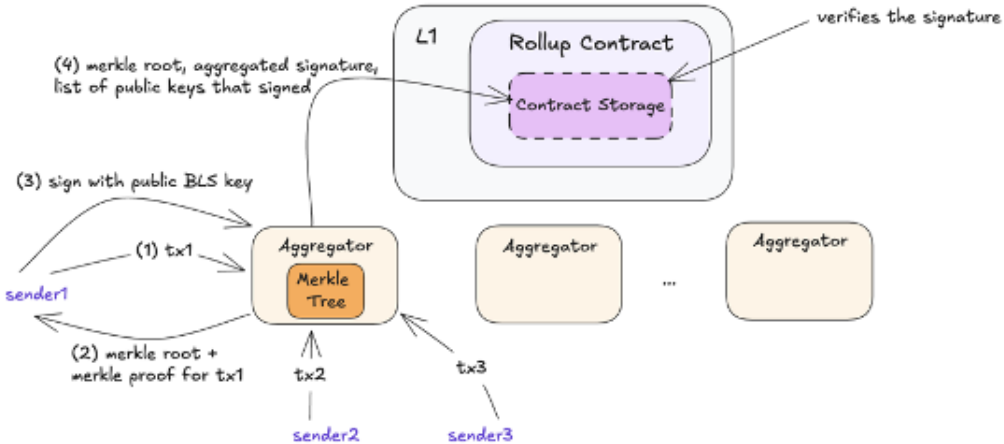
The internal state consists of the list of all blocks that have been added:

$$\mathcal{S}_{contract} := \mathcal{B}^*$$

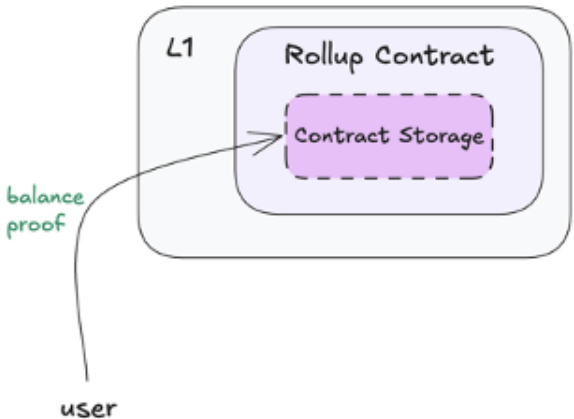
1. Deposit funds in the rollup



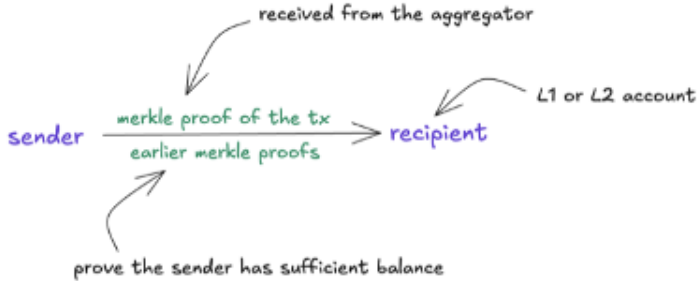
2. Transfer funds within the rollup



3. Withdraw funds from the rollup



Balance proof



Users generate offchain proofs of their own balances.

To prove their own balance, each user needs to keep track of all proofs they have received from aggregators and other users.

Balance Function

Balance Function

This is where all the magic happens.

The protocol uses **lattice-ordered abelian groups** for transaction values and account balances.

The balance function is used by

1. users to compute their own balance,
2. the rollup contract when processing a withdrawal.

Balance Function

> **The balance function takes** a balance proof and the current state of the rollup contract, and returns **the balance of each account** in the rollup that can be proven by the balance proof.

$$\mathrm{Bal} : \Pi \times \mathcal{B}^* \to \mathcal{S}$$

$$\mathcal{S} = \{b \in \mathcal{V}^{\overline{\mathcal{K}}} \mid 0 \leq_l b_k, \ \textit{for all } k \in \overline{\mathcal{K}} \setminus \{\mathrm{Source}\}\}$$

$$\overline{\mathcal{K}} := \mathcal{K}_1 \amalg \mathcal{K}_2 \amalg \{\mathrm{Source}\}$$

Source is a special account used to represent deposits and withdrawals.

Balance Function

Given a balance proof $\pi \in \Pi$ and the current rollup state $B_* \in \mathcal{B}^*$,
the account balances are computed in two steps:

1. **Extract a list of partial txs from π and B_*.**
   A partial tx consists of a sender, a recipient, and a (possibly unknown) amount.
2. **Compute the balances of every account**,
   by applying a state transition function on the list of partial txs.

Because some of the txs are unknown, we cannot know for sure what the balance of each account is.
Instead, **we will compute a lower bound on the balance of each account.**

Step 1: Extracting a list of partial transactions

Let π be a balance proof and B_* the current list of blocks.

The set of partial transactions is $\mathcal{T} \subseteq \overline{\mathcal{K}}^2 \times Maybe(\mathcal{V}_+)$.

> **Deposit block**. For a deposit block (r, v), we extract the partial transaction
>
> $$(r, v) \mapsto ((\text{Source}, r), v)$$

Step 1: Extracting a list of partial transactions

Transfer block. For a transfer block $(aggregator, extradata, C, S, \sigma)$, we extract, for each $(s, r) \in \mathcal{K}_2 \times \mathcal{K}$, where $s \neq r$, in lexicographic order, the partial transaction $((s, r), v)$, where

$$v = \begin{cases} t(r), \ where\ (\_, t) = \pi(C, s) & , \ if\ s \in S\ and\ \pi(C, s) \neq \bot \\ \bot & , \ if\ s \in S\ and\ \pi(C, s) = \bot \\ 0 & , \ if\ s \notin S \end{cases}$$

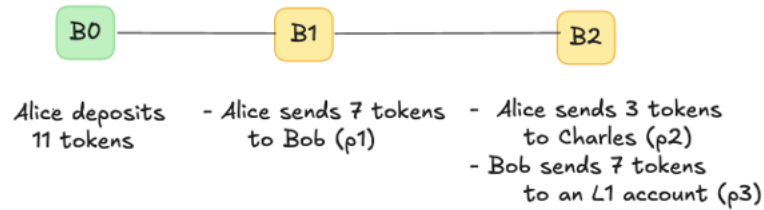
Note that we need the balance proof π for the extraction!

Step 1: Extracting a list of partial transactions

Withdrawal block. For a withdrawal block B, we extract the partial transaction

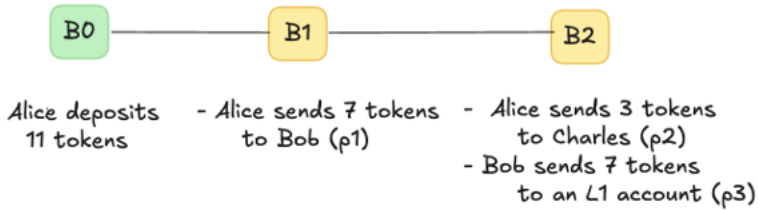
$$B \mapsto ((s, \mathrm{Source}), B_s)_{s \in \mathcal{K}_1}$$

Example



- **Alice's wallet:** $\pi_a = p_1 +\!\!+ p_2$
- **Bob's wallet:** $\pi_b = p_1 +\!\!+ p_3$
- **Charles's wallet:** $\pi_c = p_1 +\!\!+ p_2$

Example



$$TxsInBlocks(\pi_a, B^*)^{simpl} = [((\text{Source}, A), 11), ((A, B), 7), ((A, C), 3), ((B, A), \bot), ((B, C), \bot), ((B, L), \bot)]$$

$$TxsInBlocks(\pi_b, B^*)^{simpl} = [((\text{Source}, A), 11), ((A, B), 7), ((A, B), \bot), ((A, C), \bot), ((B, L), 7)]$$

Step 2: Computing the balances of every account

For any **complete transaction** $((s,r),v)$ and for all $b \in \mathcal{S}$ and $p \in \mathcal{K}$, we define

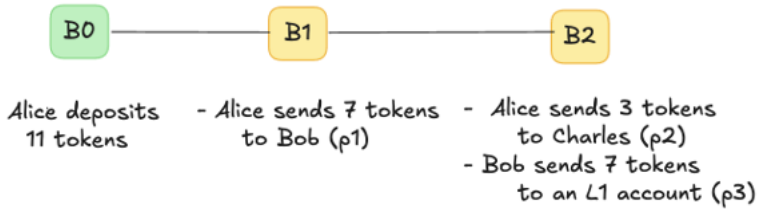
$$f_c(((s,r),v),b)_p := \begin{cases} b_p, & \text{if } p \neq s \text{ and } p \neq r \\ b_s - v \wedge b_s, & \text{if } p = s \text{ and } s \neq Source \\ b_r + v \wedge b_s, & \text{if } p = r \text{ and } s \neq Source \\ b_s - v, & \text{if } p = s \text{ and } s = Source \\ b_r + v, & \text{if } p = r \text{ and } s = Source \end{cases}$$

Step 2: Computing the balances of every account

The **transition function** for any transaction can be defined as follow:

$$f(T,b)_k := \begin{cases} f_c(T,b)_k, & \text{if } v \neq \perp \\ 0, & \text{if } v = \perp \text{ and } k = s \\ b_k, & \text{if } v = \perp \text{ and } k \neq s \end{cases}$$

Example



- **Alice's wallet:** $\pi_a = p_1 +\!\!+ p_2$
- **Bob's wallet:** $\pi_b = p_1 +\!\!+ p_3$
- **Charles's wallet:** $\pi_c = p_1 +\!\!+ p_2$

Example

Alice's wallet: $\pi_a = p_1 +\!\!+ p_2$

$$TxsInBlocks(\pi_a, B^*)^{simpl} = [((\text{Source}, A), 11), ((A, B), 7), ((A, C), 3), ((B, A), \bot), ((B, C), \bot), ((B, L), \bot)]$$

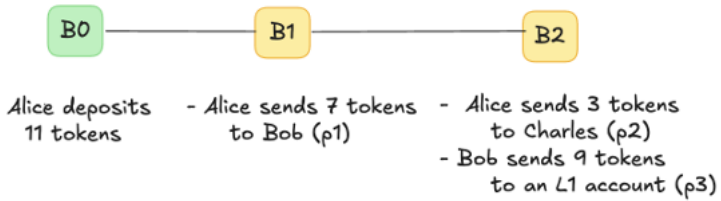
$$
\begin{aligned}
&(b_A = 0,\ b_B = 0,\ b_C = 0,\ b_{\text{Source}} = 0,\ b_{Lb} = 0) \xrightarrow{((\text{Source},A),11)} \\
&(b_A = 11,\ b_B = 0,\ b_C = 0,\ b_{\text{Source}} = -11,\ b_{Lb} = 0) \xrightarrow{((A,B),7)} \\
&(b_A = 4,\ b_B = 7,\ b_C = 0,\ b_{\text{Source}} = -11,\ b_{Lb} = 0) \xrightarrow{((A,C),3)} \\
&(b_A = 1,\ b_B = 7,\ b_C = 3,\ b_{\text{Source}} = -11,\ b_{Lb} = 0) \xrightarrow{((B,A),\bot)} \\
&(b_A = 1,\ b_B = 0,\ b_C = 3,\ b_{\text{Source}} = -11,\ b_{Lb} = 0) \xrightarrow{((B,C),\bot)} \\
&(b_A = 1,\ b_B = 0,\ b_C = 3,\ b_{\text{Source}} = -11,\ b_{Lb} = 0) \xrightarrow{((B,L),\bot)} \\
&(b_A = 1,\ b_B = 0,\ b_C = 3,\ b_{\text{Source}} = -11,\ b_{Lb} = 0)
\end{aligned}
$$

Double-spending

The withdrawing steps ensure that a user cannot double-spend by withdrawing the same funds twice.

The amount to be withdrawn is computed by applying the balance function to the balance proof and all previous blocks that have been added to the rollup.

Example



- **Alice's wallet:** $\pi_a = p_1 +\!\!+ p_2$
- **Bob's wallet:** $\pi_b = p_1 +\!\!+ p_3$
- **Charles's wallet:** $\pi_c = p_1 +\!\!+ p_2$

Example

Bob's wallet: $\pi_b = p_1 +\!\!+ p_3$

$$TxsInBlocks(\pi_b, B^*)^{simpl} = [((\text{Source}, A), 11), ((A, B), 7), ((A, B), \bot), ((A, C), \bot), ((B, L), 9)]$$

$$
\begin{aligned}
&(b_A = 0,\ b_B = 0,\ b_C = 0,\ b_{\text{Source}} = 0,\ b_{Lb} = 0) \xrightarrow{((\text{Source},A),11)} \\
&(b_A = 11,\ b_B = 0,\ b_C = 0,\ b_{\text{Source}} = -11,\ b_{Lb} = 0) \xrightarrow{((A,B),7)} \\
&(b_A = 4,\ b_B = 7,\ b_C = 0,\ b_{\text{Source}} = -11,\ b_{Lb} = 0) \xrightarrow{((A,B),\bot)} \\
&(b_A = 0,\ b_B = 7,\ b_C = 0,\ b_{\text{Source}} = -11,\ b_{Lb} = 0) \xrightarrow{((A,C),\bot)} \\
&(b_A = 0,\ b_B = 7,\ b_C = 0,\ b_{\text{Source}} = -11,\ b_{Lb} = 0) \xrightarrow{((B,L),9)} \\
&(b_A = 0,\ b_B = 0,\ b_C = 0,\ b_{\text{Source}} = -11,\ b_{Lb} = 7)
\end{aligned}
$$

Security · Safeness

The rollup contract is secure if every withdrawal request succeeds.
The rollup contract has sufficient balance for every withdrawal.

If a user has a balance proof which proves the in-rollup balance of one or more of their L1 accounts, they will be able to withdraw these balances to L1.

Breaking this security result is as hard as

- breaking the binding property of the authenticated dictionary scheme, or
- finding a collision of the hash function

Formal verification of the protocol

We formally verified the protocol and its safety property in Lean4.

The formalization

- Mirror the INTMAX2 white paper
- Open source
- Approx. 3.3 k lines of Lean
- Uses **mathlib** for handling the theory of l-groups.

INTMAX2

White paper

Blog post on
the Lean formalization

Thank you.