# ZK-ROLLUP

23.03.30 ZK-School

# Rollups

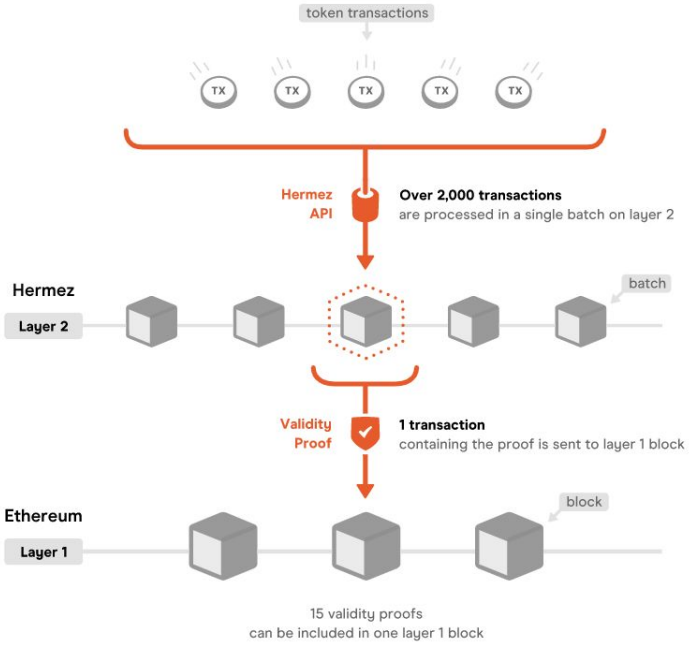


## ZK- Rollups

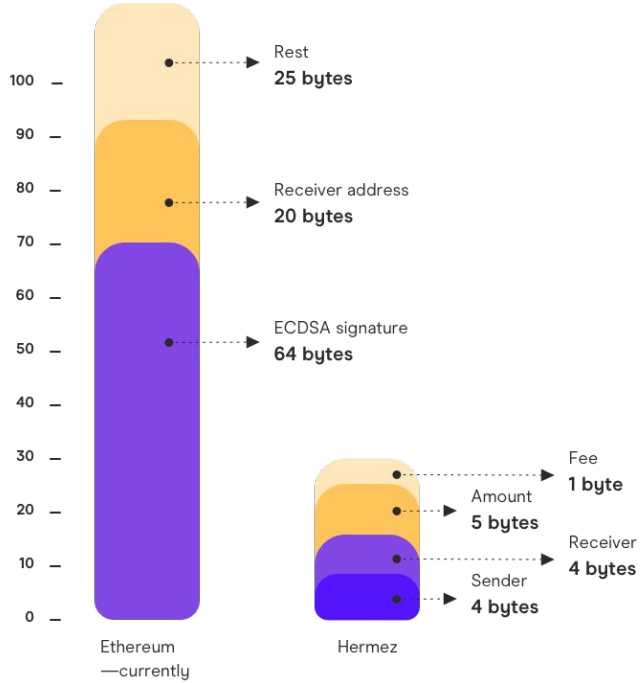
- Can only execute simple txns
- Each txn is cryptographically verified
- Finality achieved much quicker

## Optimistic Rollups

- Can execute Smart contracts
- Txns not verified initially, but anyone can challenge the validity of a txn.
- Txns are processed quickly, but achieving finality can take weeks




Bytes breakdown: vanilla Ethereum transaction (109+ bytes) vs zk-rollup transaction (14 bytes)



# Hermez v1.0: zk rollup : Payment , Token transfer

**Layer 1 — smart contract**

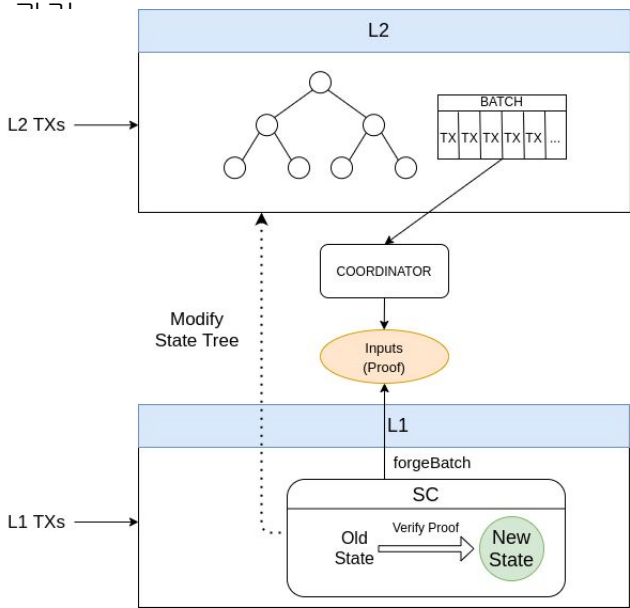
- Hermez smart contract: 1) User의 L1 tx (generate L2 account, Deposit, Withdrawal) 수집, 2) verify proof, 3) state update
- Consensus smart contract: 블록생성자를 뽑는 경매를 관리, 입찰비 관리
- Withdrawal Delayer smart contract: 시스템에 내장된 출금 보호 메커니즘 관리

**Layer 2 — 트랜잭션 실행 후 관리**

- state tree: account, balance → (검증 후) new verifiable valid state
- batch: transaction
- validity proof: SNARK

**Coordinator** = Sequencer, Operator, Prover, Block producer

1. (L2) Heremez smart contract에서 pending tx를 가져와서 실행함
2. (L2) validity proof 생성: 해당 트랜잭션이 올바르게 수행되었음을 증명
3. (L1) validity proof 검증 및 state update (smart contract)

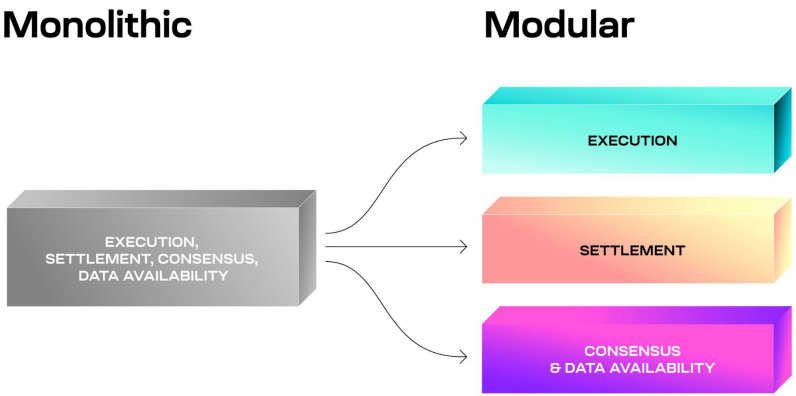


# Data Availability

Data availability: L2의 트랜잭션은 모두 L1에 공개(publish)됨.

state tree를 재구현하기 위해. state tree만 재구현 하면 되기 때문에

일종의 요약 data만 게시되어 fraud proof보다 더 많은 tx를 담을 수 있음 (scalability가 더 좋음)



# Account

Account

1. Regular Rollup Account
    - L1과 L2에서 모두 사용될 수 있는 account로 L1의 ethereum address로 만들 수 있음
    - 이 account에는 ethereum address와 babyjubjub public key가 포함됨
        - ethereum address: L1의 거래를 승인하는데 사용
        - babyjubjub public key: L2의 거래를 승인하는데 사용
2. Internal Rollup Account
    - L2에서만 사용할 수 있는 account로 babyjubjub public key만 가지고 있음
    - Internal account를 생성할 때는 ethereum add의 authorization이 따로 필요 없으며 babyjubjub key만 가지고 있으면 됨

# Tree

1. State Tree (account, balance)
    - Sparse merkle tree: zk rollup state를 나타내기위해 사용되며, root 값으로 구별됨
    - state tree(account)의 각 leaf에는 아래 data가 포함되어 있음
        - key: Merkle tree index (idx)
        - value: Hash(state)
          State hash = H(e0, e1, e2, e3)
          e0: tokenID, nonce, sign
          e1: balance
          e2: ay
          e3: ethAddr
    - hash 값은 key-value pair로 sparse merkle tree에 삽입됨
2. Exit Tree
    - 각 batch에는 사용자가 수행하는 모든 exits(L1 or L2 exit transaction)과 연관된 exit tree가 있음
    - exit tree는 state tree와 동일한 leaf 구조를 가짐
    - 자금 출금을 위해서 사용자는 exit tree의 left에 자신의 것이 있음을 증명해야하며, 이는 merkle tree proof를 제출하거나 zk proof를 제출하여 검증할 수 있음.
    - Layer 2에서 자금을 state tree에서 exit tree로 옮긴 다음, Layer 1에서 withdrawal transaction을 요청해야함

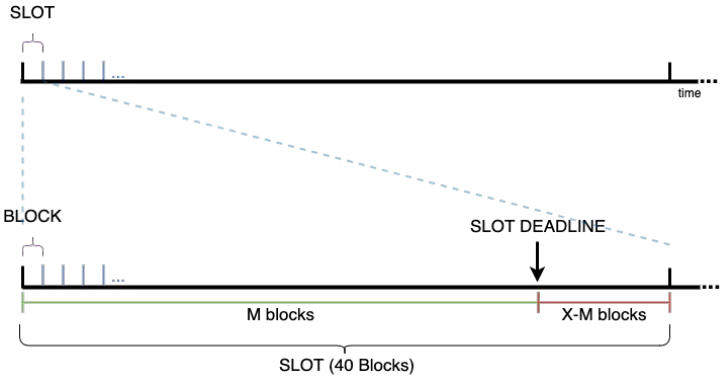
# Transaction

1. **L1 User transaction**: L1UserTx (ToForgeL1TxsNum, Position)
    - 사용자가 L1의 smart contract에 전송하는 트랜잭션
    - account 생성, deposit, withdrawal
    - smart contract는 유저에게 받은 tx를 pending queue에 추가하고, coordinator의 L2 BATCH에 포함되어 처리되도록 강제함
    - coordinator가 forging 중일 때 다음 batch에 포함될 pending queue은 frozen되고, 이후 들어오는 L1 tx는 다음 pending queue에 추가됨
2. **L2 transaction**: L2Tx (FromIdx, TokenID, Amount, Nonce, Fee)
    - Coordinator는 해당 트랜잭션이 유효한 트랜잭션인지 확인해야함
    - L2에서만 실행되는 트랜잭션
        i. Transfer: rollup account간의 자금 전송
        ii. Exit: state tree에 있는 자금을 exit tree로 전송
        iii. TransferToEthAddr: 수신자가 rollup에 존재하지 않을 경우 coordinator에게 임의로 보내어, coordinator가 수신자의 ethereum add에 직접 전송
    - 사용자는 L2 트랜잭션을 REST API를 통해서 직접 coordinator에게 전송해야함
    - 사용자는 트랜잭션을 전송할 coordinator를 선택할 수 있음
    - 트랜잭션 거래 수수료는 L2 거래 (transfer)에서 사용한 token과 동일한 token으로 지불함

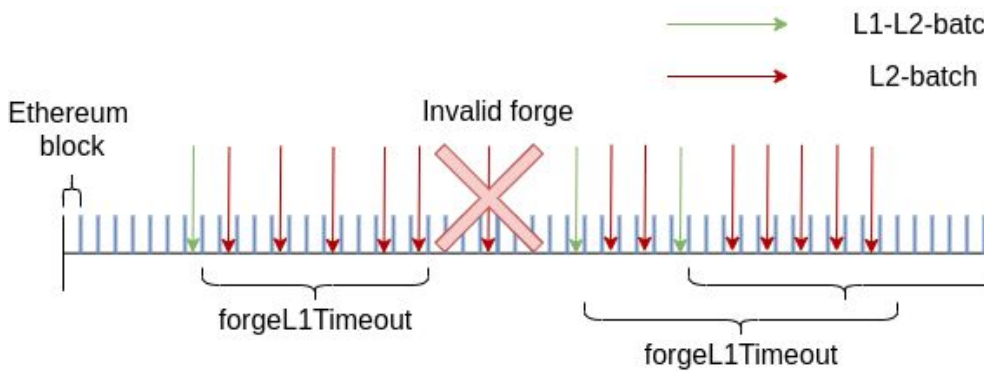
# Coordinator

Consensus smart contract

- Coordinator를 선정하는 auction을 실행하는 smart contract
- Slot: 40 Ethereum Blocks = ~ 10 mins
- Ethereum block = ~ 15s
- Auction — Coordinator
- 모든 slot에서 존재했던 node들이 auction 출전 자격을 얻음
- auction 동안 가장 높은 입찰가를 제시하는 node가 해당 slot에 대한 coordinator가 될 수 있음
- Coordinator가 되면 1) 트랜잭션 배치를 처리하고, 2) corectness를 증명하는 proof를 만들고 3) 처리 수수료를 획득할 수 있음



# Protection Mechanism



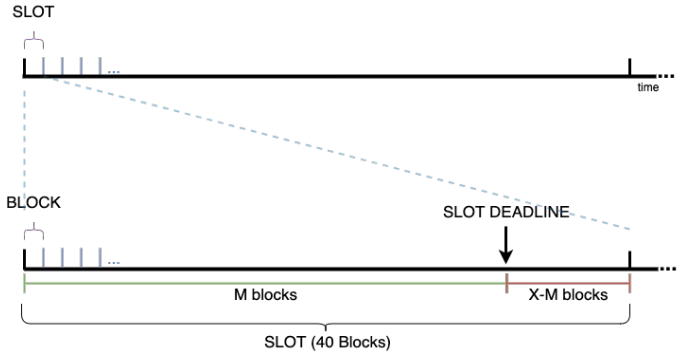

*Coordinator가 거래를 제대로 처리하도록 강제하기 위*

1. L1/L2 Batches
   - Coordinator가 처리해야하는 batches의 종류는 2개임
     - L1의 smart contract에서 pending queue에서 대기중인 tx: account, deposit, withdrawal
     - L2에서 발생하여 coordinator에게 직접 전송된 tx: transfer, exit
   - forgeL1Timeout
     - Coordinator가 L1의 pending tx도 주기적으로 처리하도록 만든 장치
     - L1 batch를 forge해야지 forgeL1Timeout이 시작되고, 데드라인 전까지 L2 batch를 forge할 수 있음 (forge: batch 처리, proof 생성, proof 검증)
     - forgeL1Timeout이 끝나면, 다음 L1 batch를 처리하기 전에 다른 L2 batch를 처리할 수 없음 (invalid forge)
     - forgeL1Timeout 사이에 다른 L1 batch를 처리하면 데드라인이 업데이트됨
2. Coordinator Override
   - Coordinator가 Slot 내에 forge하지 않은 경우, 다른 coordinator가 입찰없이 batch를 forge할 수 있음

# Protection Mechanism

*Coordinator가 거래를 제대로 처리하도록 강제하기 위한 장치*

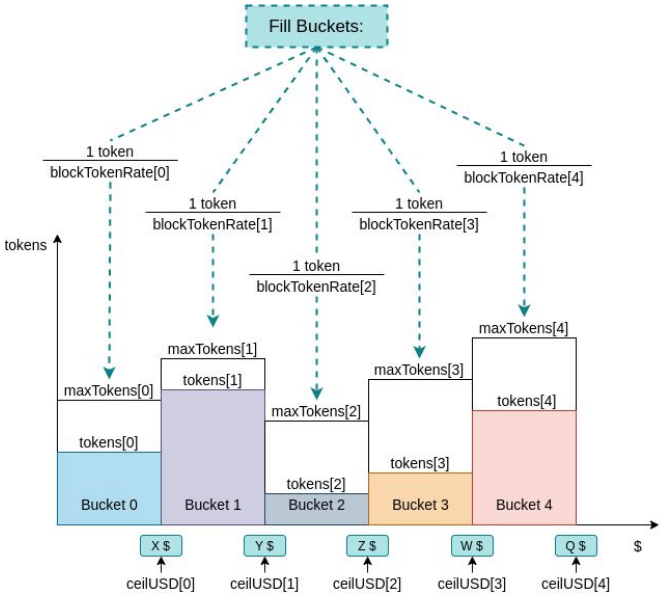
1. Coordinator Override
    - Coordinator가 Slot 내에 forge하지 않은 경우, 다른 coordinator가 입찰없이 batch를 forge할 수 있음
    - Slot deadline: Coordinator 행위를 감시할 수 있는 최대 기간
    - 한 명이라도 제대로 동작하고 있는 coordinator가 있다면, Hermez network가 실행됨을 보증하기 위한 장치



# Withdrawal

*사용자가 **Heremz zk rollup**에서 자금을 회수하기 위한 방법*

- 아래 트랜잭션을 연속으로 실행해야함
  - (Layer 2) Exit tx: 자금을 state tree에서 exit tree로 옮김
  - (Layer 1) Withdrawal: 자금을 smart contract에서 사용자의 ethereum address로 옮김
- exit tx를 보내면 자금은 hemrez smart contract에 보관되며, withdrawal data에는 이를 식별하는 unique data가 포함됨. smart contract는 여러번 인출하는 것을 피하기 위해 아래 데이터를 저장함
  identified uniquley by:
  - merkle tree index
  - number exit root
- Hermez Withdrawal Limit (Leaky bucket algorithm)
  - 자금을 출금하는 속도와 한도는 leaky bucket 알고리즘에 의해 결정되며, smart contract에 포함된 이용 가능한 credit에 따라 withdrawal이 지연될 수 있음



# Adding New Tokens

- ERC20만 가능
- Governance에서 토큰 추가에 따른 수수료를 결정하고, L2에서 허용된 토큰 목록을 관리함
- 최대 2^32 개의 토큰 수용 가능
- smart contract는 등록된 모든 토큰 리스트를 저장하고 있음
- token 0 = ETH

# Hermez contract & Circuit

Hermez는 Layer 1에 3개의 contract를 배치해놓았다. 이 중 Hermez smart contract가 대부분의 기능을 수행하고 있으므로, 이 contract의 기능과 동작에 대해 자세히 살펴보도록 하자.

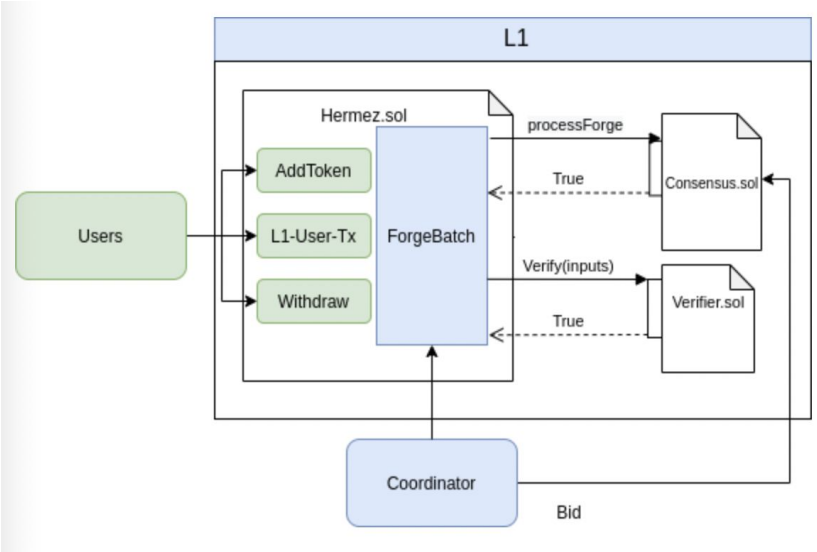
1. Hermez smart contract
   User의 Layer 1 트랜잭션 관리, 증명 검증, state update 등의 주요한 기능을 모두 수행한다.

2. Consensus smart contract
   Coordinator를 선정하는 auction을 실행한다.

3. WithdrawalDelayer
   Hermez smart contract에 할당된 출금액이 bucket의 credit을 초과할 경우, WithdrawalDelayer contract를 통해 지연 출금으로 처리된다 .

# Main Features

1. Handle L1-user transaction : addL1Transaction
2. Add new tokens to the rollup : addToken
3. Withdraw funds to the rollup : withdrawFunds, withdrawCircuit
4. Forging (generate L2’ tx batches, verify proof)
    1. Ask consensus algorithm for coordinator approval
    2. Add Layer 1 Coordinator Transaction :
    3. Verify validity proof : VerifierRollupInterface 호출, VerifierWithrawInterface 호출
    4. Ensure that these transactions are forged
    5. Forge batches: forgeBatch, updateForgeL1L2BatchTimeout
    6. Set a new state Merkle root
    7. Set a new state exit Merkle root : withdrawMerkleProof
    8. Delete the current frozen queue and freeze the next one : L1QueueAddTx, clearQueue



# Handle L1 User Transaction

addL1Transaction

- Layer 1에서 발생하는 모든 트랜잭션 (account 생성, deposit, withdrawal)은 queue에 추가되고, queue가 가득차거나 frozen되면 새로운 queue가 생성됨
- queue frozen: Layer 1의 트랜잭션을 추가할 수 없는 상태
- L1-L2-batch의 queue index는 항상 frozen되어 nextL1ToForgeQueue로 식별됨
  - L1-L2-batch: L1의 smart contract에서 pending queue에서 대기중인 tx
  - L2-batch: L2에서 발생하여 coordinator에게 직접 전송된 tx(transfer, exit)
  - Coordinator는 L1-L2-batch를 forge 해야만 forgeL1Timeout이 시작되어 L2-batch를 forge할 수 있기 때문에, nextL1ToForgeQueue로 따로 식별되는 것으로 보임

**Layer User Tx 추가하는 함수 호출하기**

- frozen 되지 않은 queue에 L1UserTx data 추가
  - queue index: nextL1FillingQueue
  - L1UserTx는 byte array data로 인코딩되어 queue에 추가됨
- reserved queue for the L1-coordinatorTx = MAX_L1_TX - len(L1-User_TXS)
  - MAX_L1_TX: L1TXQueue의 최대 길이(수용 개수)
  - MAX_L1_USER_TX: L1UserTx가 수행할 수 있는 최대 개수

# Add new tokens, Withdraw funds to the rollup

**Add new tokens to the rollup**

addToken

- ERC20 토큰만 가능
- 누구나 새로운 token을 추가할 수 있음
- Token 추가 수수료는 HEZ로 지불하고 이는 governance address로 보내짐
- Governance에서 토큰 추가에 따른 수수료를 결정하고, L2에서 허용된 토큰 목록을 관리함
- 최대 2^32 개의 토큰 수용 가능
- smart contract는 등록된 모든 토큰 리스트를 저장하고 있음

**Withdraw funds to the rollup**

withdrawFunds

- Layer 1 Hermez smart contract이 가지고 있는 자금을 이더리움 주소로 출금하기 위한 거래
- 이를 위해서는 exit tree에 해당 요청이 leaf로 존재함을 증명해야함
- Withdrawal은 Layer 1 트랜잭션이 아니며 state 또는 exit tree에 영향을 미치지 않아 circuit에서 처리되지 않음

# Forging

**generate L2’ tx batches, verify proof**

1. Ask consensus algorithm for coordinator approval
    - 외부 consensus smart contract를 호출하여 coordinator가 forge할 수 있는 권한이 있는 사용자인지 확인함
2. Add Layer 1 Coordinator Transaction :
    - Coordinator가 L1-Coordinator-Tx를 추가함
    - CreateAccountEth, CreateAccountBjj
3. Verify validity proof : VerifierRollupInterface 호출, VerifierWithrawInterface 호출, withdrawCircuit
    - validity proof에 대한 circuit proof를 검증함
    - withdrawCircuit: exit state root로 출금 요청이 존재하는지 검증하기 위한 것
4. Ensure that these transactions are forged
5. Forge batches: forgeBatch, updateForgeL1L2BatchTimeout
    - L2 batch: L2 트랜잭션과 L1 coordinator 트랜잭션을 forge함
    - L1-L2-batch: L1 user 트랜잭션, L1 coordinator, L2 트랜잭션을 forge함. Coordinator는 첫 번째 frozen queue에 있는 모든 L1 트랜잭션을 forge해야함
6. Set a new state Merkle root
7. Set a new state exit Merkle root : withdrawMerkleProof
8. Delete the current frozen queue and freeze the next one : L1QueueAddTx, clearQueue

# Circuit (zk-SNARK)

**Circuit의 세 가지 모듈**

1. library: Hermez circuit에서 일반적으로 사용하는 structs

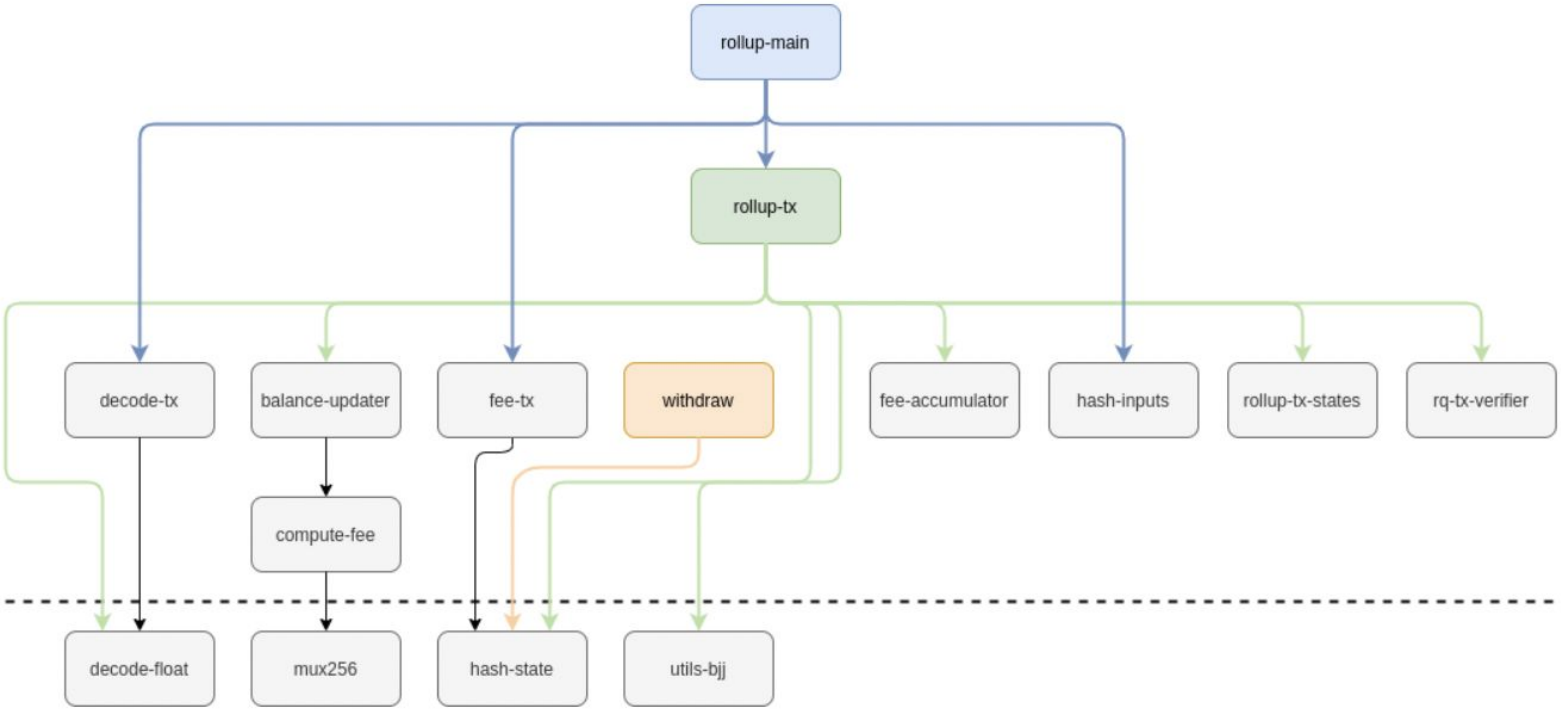
2. withdraw: user가 Hermez contract에서 자금을 인출할 수 있도록 하는 circuit
   사용자는 zk proof 혹은 merkle tree proof를 제출하여 출금할 수 있음. 두 방법 모두 기능면에서는 동일함

3. rollup-main: ZK-Rollup protocol에 설명된 모든 logic을 포함하는 main circuit

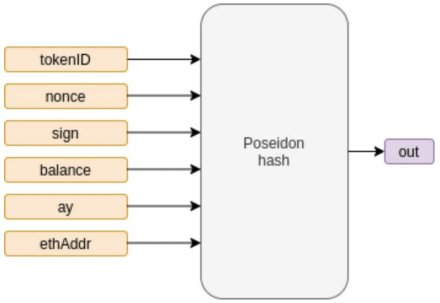
**Global variables**

- nTx: L1 또는 L2 트랜잭션의 최대 수용 개수
- nLevels: merkle tree depth
- maxL1Tx: L1 트랜잭션의 최대 수용 개수
- maxFeeTx: 수수료 트랜잭션의 최대 수용 개수

# Circuits Organization



# Circuits..

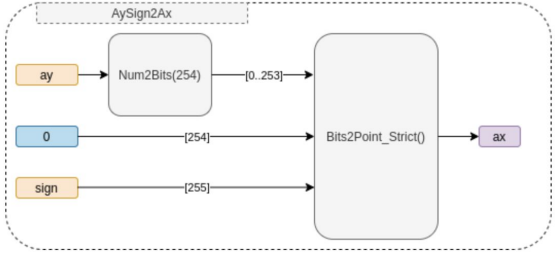



**hash-state.circom**

- state tree에 포함할 hash value를 계산하는 로직으로, state를 input으로 가져옴
- state tree의 leaf에는 는 account와 balance의 정보가 저장되어 있음

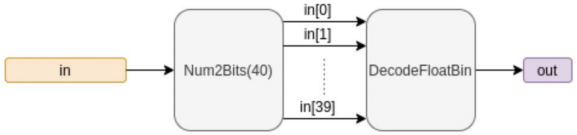
**utils-bjj.circom**

- BitsCompressed2AySign: bjjCompressed[256] 비트를 가져와서 account state에 삽입된 ay 와 sign을 검색한다.
- AySign2Ax: ay 와 sign을 얻어, ax 좌표를 계산한다.


**decode-float.circom**

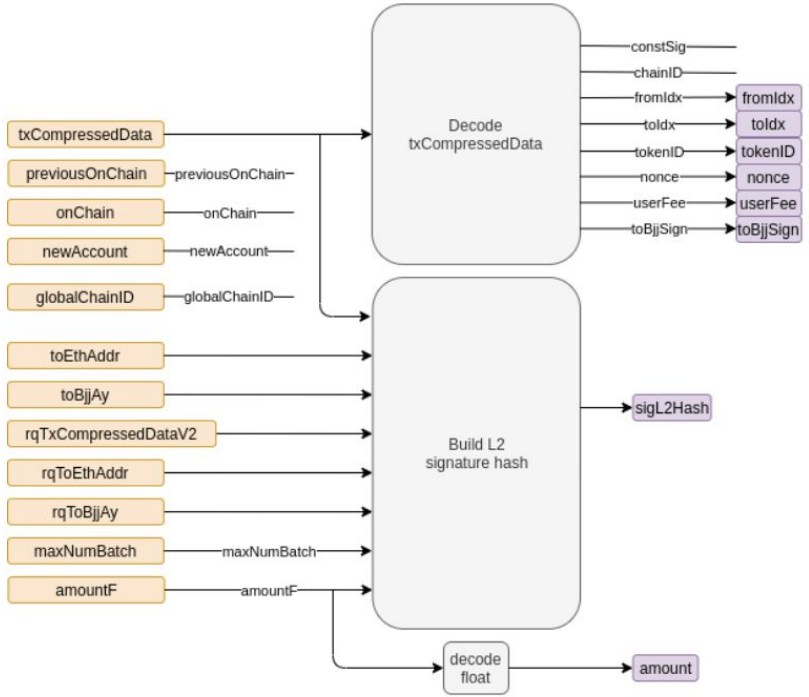
- float40 format을 가져와서 input에 넣어 large integer value 값으로 디코딩
- Steps:
  1. 40이하의 significant bits 가져
  2. 지수 계산
  3. mantissa 계산
  4. final large integer 계산



# Circuits..

**decode-tx.circom**

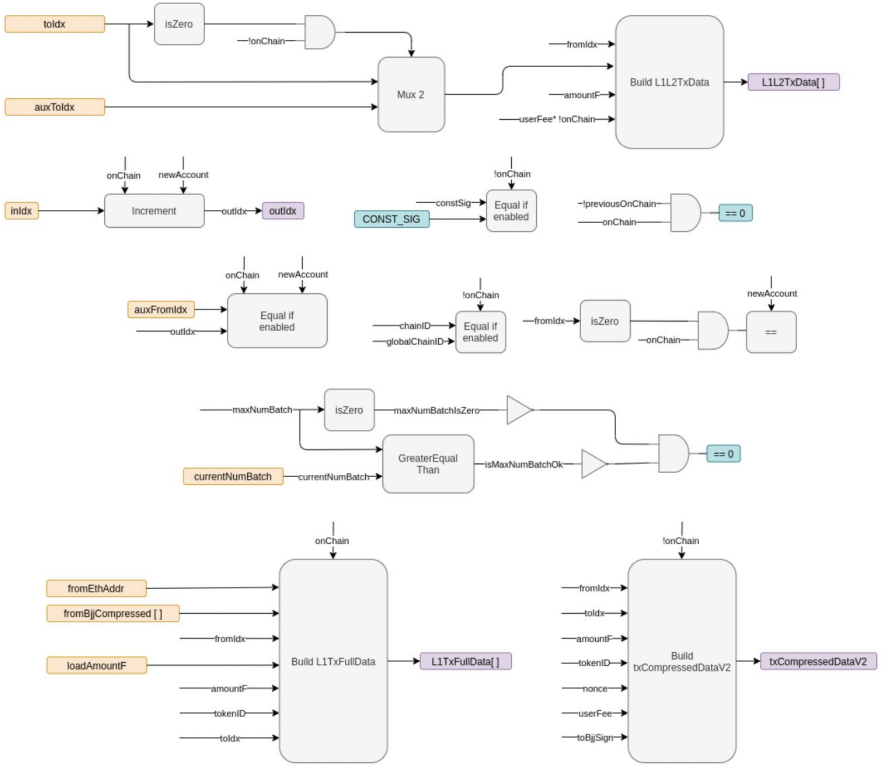
- **Decoders/Build**: 트랜잭션 데이터를 디코딩하고, circuit에서 사용할 data structures를 만듬
- decodes txCompressedData
- builds txCompressedDataV2
- builds L1-L2 data availability L1L2TxData
- builds message to sign by L2 transactions sigL2Hash
- build L1 full data L1TxFullData
-



# Circuits

Checks 항목들: 트랜잭션 fields에 대한 check를 수행함

- L1 transactions must be processed before L2 transactions
- only switching from L1 to L2 is allowed
- checks newAccount is set to true only when it is an L1 transaction and fromIdx is 0
- idx to be assigned to a new account creation is incremented and checked only if the transaction involves an account creation
- checks chainID transaction field matches globalChainID forced by the smart contract
- checks signatureConstant transaction field matches the hardcoded value CONST_SIG set in the circuit
- checks maxNumBatch signed in the transaction is greater or equal to currentNumBatch only if maxNumBatch != 0



Questions?!