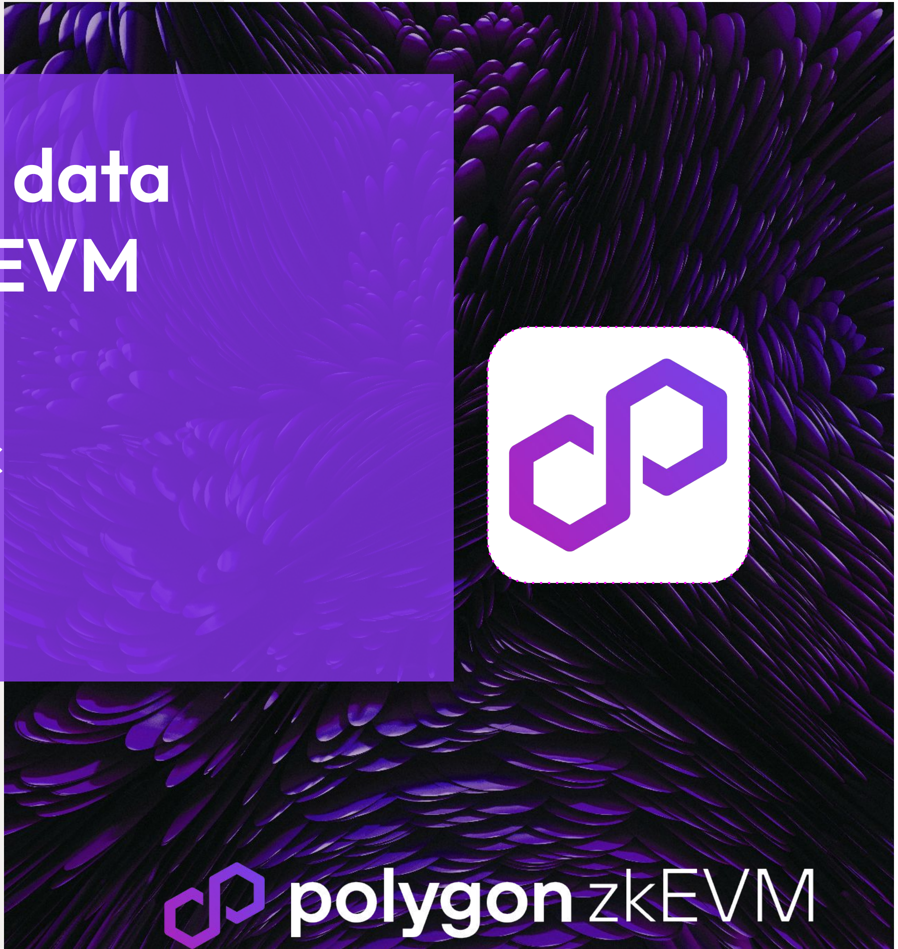
Introducing zkBlob and data compression to the zkEVM

Paris, 19 July 2023 ethCC

Carlos Matallana Espinar
Protocol & Integration Team Lead



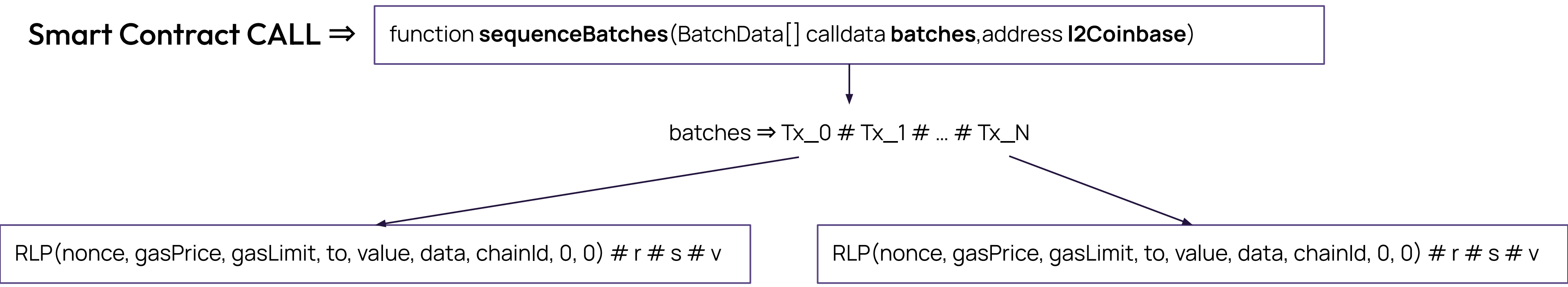
Presentation Outline

1. Intro
   - ○ Current approach
   - ○ L2 transaction cost
   - ○ EIP4844 & data compression
2. zkBlob
   - ○ Introduction
   - ○ Specification
3. Data compression
   - ○ Specification
   - ○ Examples
     - ■ ERC20 transfer

1

Intro

INTRO: Current approach



Smart Contract CALL ⇒

function **sequenceBatches**(BatchData[] calldata **batches**,address **l2Coinbase**)

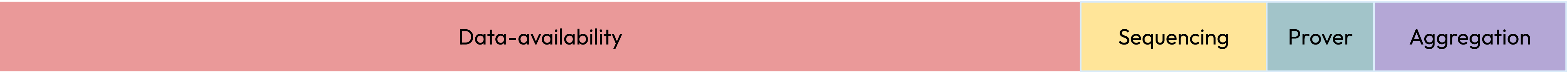
batches ⇒ Tx_0 # Tx_1 # ... # Tx_N

RLP(nonce, gasPrice, gasLimit, to, value, data, chainId, 0, 0) # r # s # v

RLP(nonce, gasPrice, gasLimit, to, value, data, chainId, 0, 0) # r # s # v

- Smart contract to receive array of batches (sequences)
- Each batch contains transactions
- Each transaction is posted on-chain in the following format:
  - **RLP(txFields) + signature**
  - Easy to compute signed message inside the **zkevm-rom**

INTRO: L2 transaction cost



- Data-availability
  - paid for every byte posted on-chain

- Sequencing
  - constant L1 transaction cost
  - shared among all batches (up to 128 kB per smart contract call)

- Prover
  - constant hardware costs
  - for each proof computed

- Aggregation
  - constant L1 transaction cost
  - shared among all sequences aggregated

INTRO: L2 transaction cost

Summary

	$ (%)			
	data-availability	**sequencing**	**prover**	**aggregation**
Ether transfer	0.238 $ (95.78 %)	0.010 $ (4.04 %)	0.00007 $ (0.03 %)	0.00038 $ (0.15 %)
ERC20 transfer	0.305 $ (94.92 %)	0.015 $ (4.73 %)	0.00018 $ (0.06 %)	0.00095 $ (0.3 %)
UniswapV2 swap	1.019 $ (96.22 %)	0.032 $ (3.05 %)	0.00136 $ (0.13 %)	0.00640 $ (0.6 %)

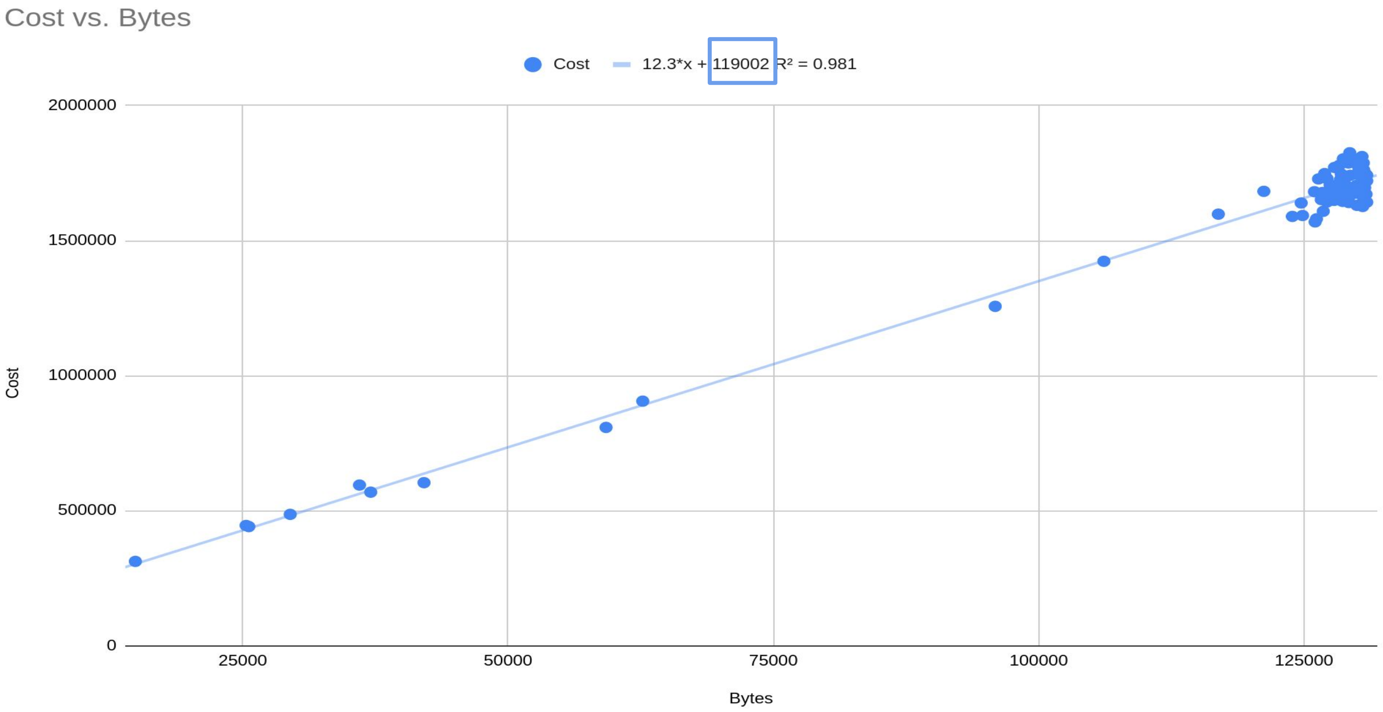
Cost computation

Assumptions ⇒

- 16 GAS byte calldata
- 128kB maximum in **sequenceBatches** SC call (geth limitation)
- Assume batches are 70% filled (**zk-counters** as larger limitant factor)
  - 350 eth transfers → 39630 bytes per batch
  - 140 ERC20 transfers → 24043 bytes per batch
  - 19 uniswapV2 swaps → 6956 bytes per batch
- Total transactions in one **sequenceBatches**
  - Eth transfers ⇒ (128k / 39630) * 350 = 1127 txs
  - ERC20 transfers⇒ (128k / 24043) * 140 = 745 txs
  - UniswapV2 swaps ⇒ (128k /6956) * 19 = 349 txs
- Conversions: 1 ETH = 1900$, gasPrice = 50 GWei

Cost computation

sequenceBatches SC CALL ⇒ 119 002 GAS (constant cost)



Cost computation

verifyBatches SC CALL ⇒

(get last 100 verifyBacthes events)

- 351 920 GAS (constant cost)
- 273 batches verified in one single call

Single batch proof ⇒

- 1 batch proof computation cost[1]: 0.0259 $

[1] https://twitter.com/eduadiez/status/1667221787178876963?s=46&t=2FzkB9sZOyy4irzxy70HVg

Do the numbers

Ether transfer

Data-availability →

- 157 bytes · 16 gas = 2512 gas
- 2512 gas * 50 GWei = 0.0001256 ether
- 0.0001256 ether * 1900 $ / Eth = **0.2386 $**

Sequencing →

- 119 002 gas / 1127 txs = 106 gas
- 106 gas * 50 GWei = 0.0000053 ether
- 0.0001256 ether * 1900 $ / Eth = **0.01007 $**

Prover →

- 0.0259 $ / 350 txs = **0.000074 $**

Aggregation →

- 351 920 gas / (273 batches * 350 txs) = 4 gas
- 4 gas * 50 GWei = 0.0000002 ether
- 0.0000002 ether * 1900 $ / Eth = **0.00038 $**

INTRO: L2 transaction cost

ERC20 transfer

Data-availability → ■ 201 bytes · 16 gas = 3216 gas ⇒ **0.305 $**

Sequencing → ■ 119 002 gas / 1127 txs = 160 gas ⇒ **0.0152 $**

Prover → ■ 0.0259 $ / 140 txs = **0.000185 $**

Aggregation → ■ 351 920 gas / (273 batches * 140 txs) = 10 gas ⇒ **0.00095 $**

UniswapV2 swap

Data-availability → ■ 671 bytes · 16 gas = 10736 gas ⇒ **1.019 $**

Sequencing → ■ 119 002 gas / 349 txs = 341 gas ⇒ **0.0323 $**

Prover → ■ 0.0259 $ / 19 txs = **0.001363 $**

Aggregation → ■ 351 920 gas / (273 batches * 19 txs) = 68 gas ⇒ **0.0064 $**

INTRO: EIP 4844 & data compression

EIP 4844

- Directly reduce gas costs from 16 gas per byte to 3 gas per byte

Data compression

- Reduce amount of bytes posted on-chain
- Approach:
  - **Specific** transaction fields compression

2

zkBlob

polygon zkEVM

ZKBLOB: Introduction

New data structure ⇒

- Allow to use EIP4844 when added to Ethereum
- Add single verification polynomial commitment (EIP4844) or hash data (current approach)

Allows to ⇒

- Aggregate signatures for each batch
  - Add SNARK proof: all transactions signatures aggregated
  - SNARK proof verified in the zkBlob
- Decoupling decompression stage from the processing stage (Batch)
  - zkBlob to build transaction from compressed data
  - Encode parameters into a custom format to be ready for batches

Current approach

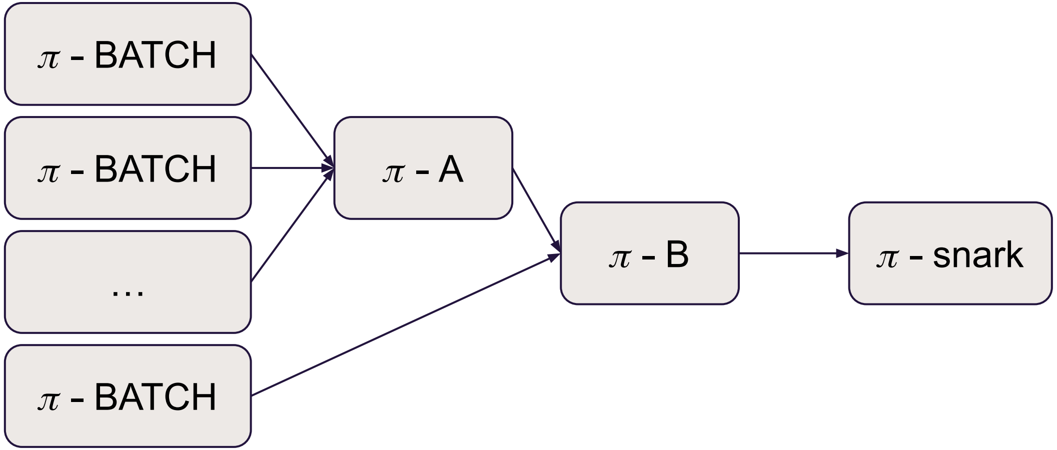
function **sequenceBatches**(BatchData[] calldata **batches**,address **l2Coinbase**)

zkevm-rom

BATCH

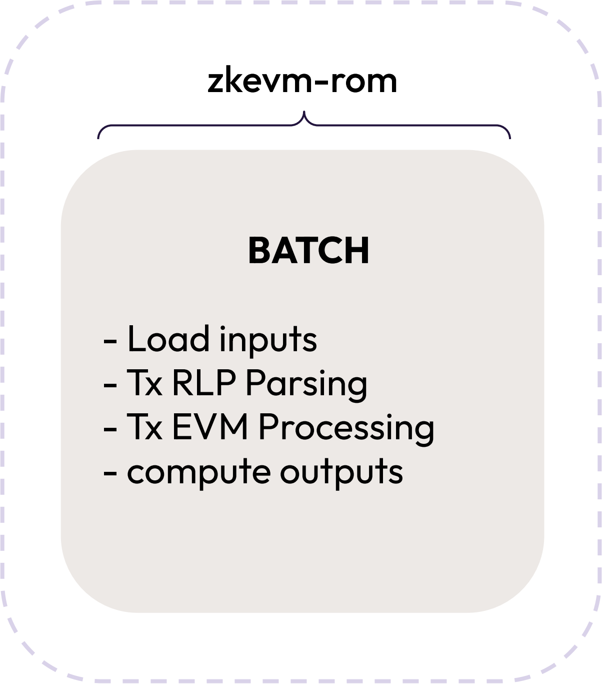
- Load inputs
- Tx RLP Parsing
- Tx EVM Processing
- compute outputs

Proof diagram



Current approach

```
function sequenceBatches(BatchData[] calldata batches,address l2Coinbase)
```

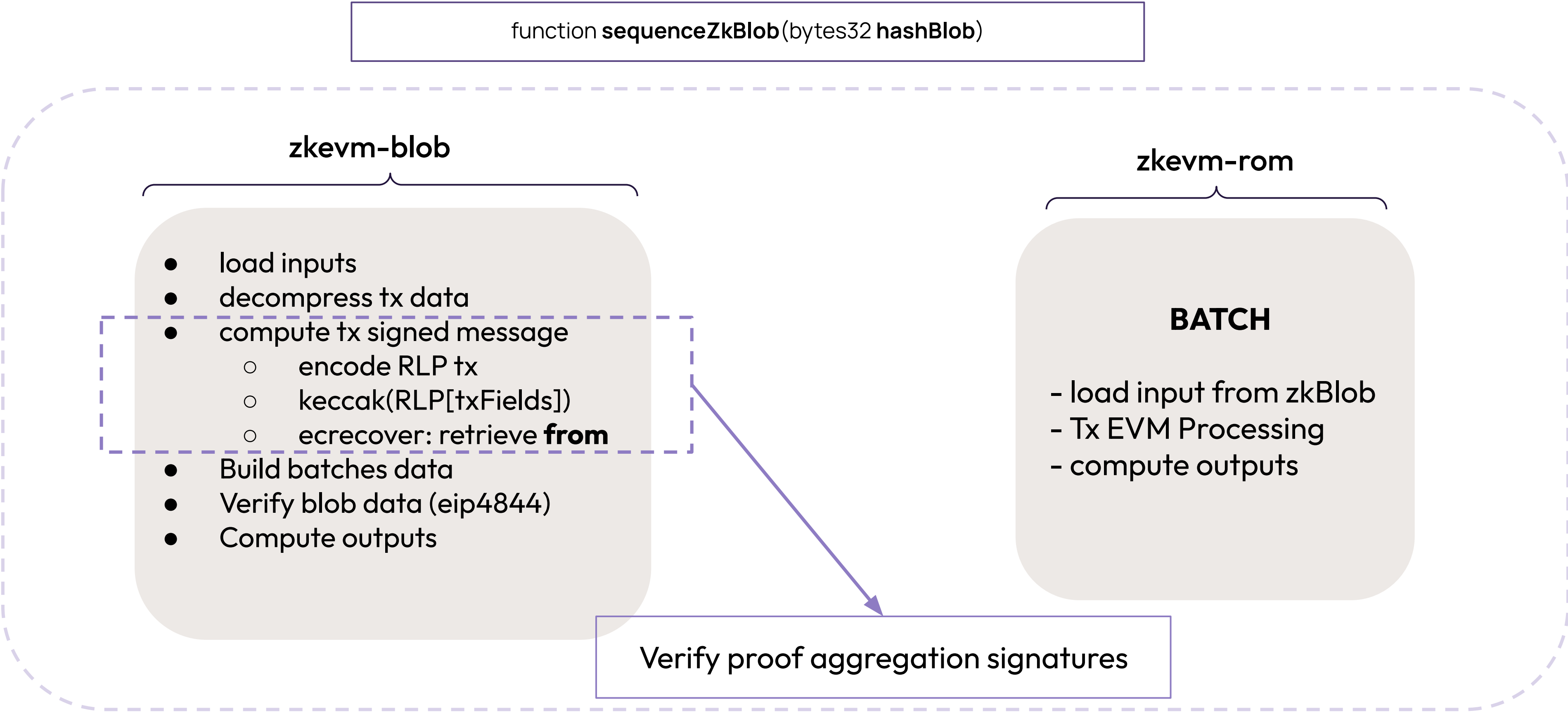


zkevm-rom

BATCH

- Load inputs
- Tx RLP Parsing
- Tx EVM Processing
- compute outputs

Add zkBlob



function **sequenceZkBlob**(bytes32 **hashBlob**)

zkevm-blob

- load inputs
- decompress tx data
- compute tx signed message
  - encode RLP tx
  - keccak(RLP[txFields])
  - ecrecover: retrieve **from**
- Build batches data
- Verify blob data (eip4844)
- Compute outputs

zkevm-rom

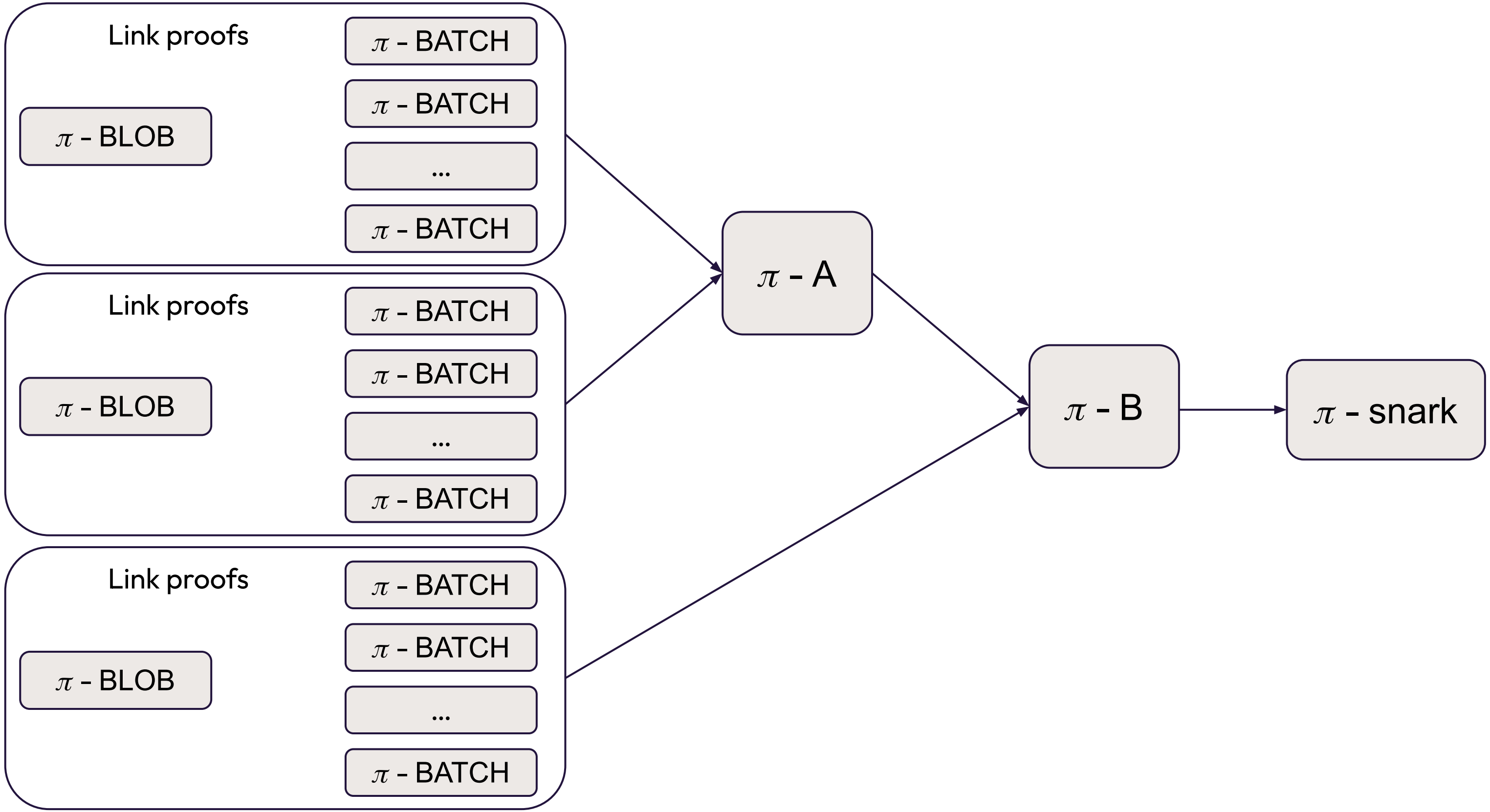
BATCH

- load input from zkBlob
- Tx EVM Processing
- compute outputs

Verify proof aggregation signatures

ZKBLOB: Specification

Proof diagram



3

Data compression

DATA COMPRESSION: Introduction

Remove signatures ⇒

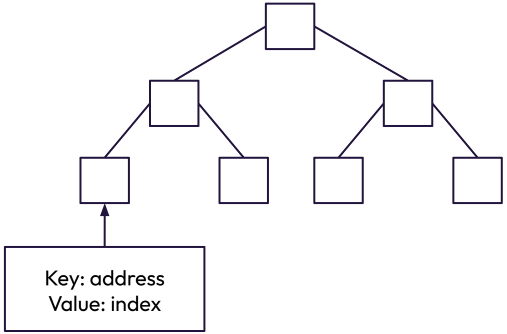
- Save 65 bytes (r, s, v) per transaction
- Cost data-availability SNARK proof shared at sequencing level by all transactions

Custom compression ⇒

- Set of custom functions: *F(input_bytes) = output_bytes*
- Reconstruct transaction bytes from compressed bytes
- Two new SMT structures
  - Address tree (address alias)
  - Data tree (32 bytes alias)

DATA COMPRESSION: Specification

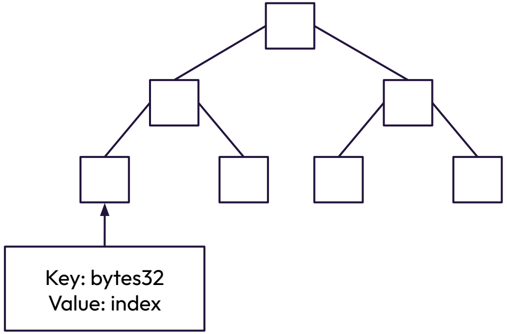
Address tree



⇒

- Incremental index
- Forced by the circuit
- Alias addresses

Data tree



⇒

- Incremental index
- Limited size
- Forced by the circuit
- Alias 32 bytes
- Useful for ***data*** field (siblings merke trees)

Custom function set

- Small set of functions
- Specific behaviour if executed in *data* field
- 3 bits → function definition

First byte		**Name**: Data < 32 bytes
3 bits (header)	5 bits (header payload)	Description
000	XXXXX	Read next XXXX bytes (up to 32 bytes)

First byte		**Name**: Large data
3 bits (header)	5 bits (header payload)	Description
001	XXXXX	Read next XXXX bytes which is the length of the bytes to read

First byte		**Name**: Small value
3 bits (header)	5 bits (header payload)	Description
010	XXXXX	Value is coded in the body

DATA COMPRESSION: Specification

First byte		**Name**: 32 bytes compressed
3 bits (header)	5 bits (header payload)	Description
011	XXXXX	Read next XXXX bytes which will be the index in the Data tree

First byte		**Name**: address compressed
3 bits (header)	5 bits (header payload)	Description
100	XXXXX	Read next XXXX bytes which will be the index in the Address tree

First byte		**Name**: value compressed
3 bits (header)	5 bits (header payload)	Description
101	XXXXX	Read next XXXX bytes which will be the mantissa and 1 byte for the exponent ($V = m \cdot 10^{e}$)

DATA COMPRESSION: Specification

First byte		**Name**:uncompressed address
3 bits (header)	5 bits (header payload)	Description
110	00000	Read 20 bytes and store it in the address tree

First byte		**Name**: uncompressed 32 bytes
3 bits (header)	5 bits (header payload)	Description
110	00001	Read 32 bytes and add it in the data tree

First byte		**Name**: data < 32 bytes pad right
3 bits (header)	5 bits (header payload)	Description
111	XXXXX	Read next XXXX bytes and pad right zeros until 32 bytes

Extra Rule

- Pad left until 32 bytes if decompression is in ***data*** transaction field
    - small value, address/data un/compressed, value compressed

DATA COMPRESSION: Example

Transaction fields: **ERC20 Transfer**

nonce: 0
gasPrice: 1000000000
gasLimit: 100000
to: 0x1275fbb540c8efc58b812ba83b0d0b8b9917ae98
value: 0
data:
0xa9059cbb000000000000000000000000617b3a3528f9cdd6630fd3301b9c8911f7bf063d000000000000
0000000000000000000000000000000000000000000000000064
chainId: 1000

Non-compressed: **rlp(nonce, gasprice, gaslimit, to, value, data, chainId, 0, 0)**

0xf86b80843b9aca00830186a0941275fbb540c8efc58b812ba83b0d0b8b9917ae9880b844a9059cbb0000000
00000000000000000617b3a3528f9cdd6630fd3301b9c8911f7bf063d000000000000000000000000000
00000000000000000000000000000000000648203e88080

Compressed: **txType # txBody ⇒ nonce, gasprice, gaslimit, to, value, data, chainId**

0x4140a10109a1010581044004a9059cbb810c01640203e8

Data availability gain

Data compression:

	Length (bytes)		
Tx Type	**Message to hash + signature**	**New full encoding**	**Gain**
ETH TRANSFER	111	15	**7.40x**
ERC20: TRANSFER	174	23	**7.57x**
UNISWAP: SWAP_EXACT_TOKENS_FOR_TOKENS	368	37	**9.95x**

EIP4844: **5x** gain

Summary (applying reduction)

	$ (%)			
	data-availability	sequencing	prover	aggregation
Ether transfer	0.0064 $ (37.98 %)	0.010 $ (59.35 %)	0.00007 $ (0.42 %)	0.00038 $ (2.26 %)
ERC20 transfer	0.0080 $ (33.29 %)	0.015 $ (62.03 %)	0.00018 $ (0.74 %)	0.00095 $ (3.93 %)
UniswapV2 swap	0.0204 $ (33.91 %)	0.032 $ (53.19 %)	0.00136 $ (2.26 %)	0.00640 $ (10.64 %)

	Eip4844 & data compression	
	before	after
Ether transfer	0.248 $	0.016 $
ERC20 transfer	0.321 $	0.024 $
UniswapV2 swap	1.058 $	0.060 $

Thanks !!

Carlos Matallana
Protocol & Integration Team Lead
carlos@polygon.technology
@KrlsMata

polygon zkEVM