# Info module

Rollup team

August 2022

## Contents



# 1 Purpose

The purpose of this small submodule is to record information and data (in separate modules) relative to

1. logging operations,
2. hashing operations,
3. and `CREATE` / `CREATE2` triggered deployments.

The information part allows us to uniquely tag individual operations of that kind and extract metadata relevant to this operation. For instance:

**For `LOG0`-`LOG4`:** the transaction number, the logger address, the log parameter ($\in \{0, 1, 2, 3, 4\}$), the size of the data to log, four topics (set to 0 for when there are fewer than 4 topics), the log number.

**For `SHA3`:** the size of the data to hash, the result of the hash, the hash number.

**For `CREATE`/`CREATE2`:** a bit which distinguishes between `CREATE` and `CREATE2` instructions, the address which runs the deployment, its nonce delta, the salt, the size of the initialization code, the resulting deployment address, the deployment number.

The Data part allows us to extract relevant data and submit it to the outside proving scheme.

## 1.1 Public data

This section is meant to give a brief overview of the way the data is structured in certain smaller modules such as the **log data module**, the **hash data module** and **the transaction call data module**. These "modules" are really data stores with which the RAM interfaces (to be precise: the mmio). The verifier generates commitments to them where we need to commit to

**Transaction call data:** a table of *padded* transaction call data. Below is a typical example

| TXNUM | $\mathsf{ADDR}^{\mathsf{hi}}$ | $\mathsf{ADDR}^{\mathsf{lo}}$ | CDS | INDEX | TXCD |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | $\mathtt{addr}_1^{\mathsf{hi}}$ | $\mathtt{addr}_1^{\mathsf{lo}}$ | $\mathsf{CDS}_1$ | 0 | $\times$ |
| 1 | $\mathtt{addr}_1^{\mathsf{hi}}$ | $\mathtt{addr}_1^{\mathsf{lo}}$ | $\mathsf{CDS}_1$ | 1 | $\times$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| 1 | $\mathtt{addr}_1^{\mathsf{hi}}$ | $\mathtt{addr}_1^{\mathsf{lo}}$ | $\mathsf{CDS}_1$ | $N_1$ | $\times$ |
| 2 | $\mathtt{addr}_1^{\mathsf{hi}}$ | $\mathtt{addr}_1^{\mathsf{lo}}$ | 0 | 0 | 0 |
| 3 | $\mathtt{addr}_3^{\mathsf{hi}}$ | $\mathtt{addr}_3^{\mathsf{lo}}$ | $\mathsf{CDS}_3$ | 0 | $\times$ |
| 3 | $\mathtt{addr}_3^{\mathsf{hi}}$ | $\mathtt{addr}_3^{\mathsf{lo}}$ | $\mathsf{CDS}_3$ | 1 | $\times$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| 3 | $\mathtt{addr}_3^{\mathsf{hi}}$ | $\mathtt{addr}_3^{\mathsf{lo}}$ | $\mathsf{CDS}_3$ | $N_3$ | $\times$ |
| 4 | $\mathtt{addr}_4^{\mathsf{hi}}$ | $\mathtt{addr}_4^{\mathsf{lo}}$ | $\mathsf{CDS}_4$ | 0 | $\times$ |
| 4 | $\mathtt{addr}_4^{\mathsf{hi}}$ | $\mathtt{addr}_4^{\mathsf{lo}}$ | $\mathsf{CDS}_4$ | 1 | $\times$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |

TXNUM is public data, its the order of transactions in the batch; TXNUM *starts counting at 1.* The address (which is separated into high and low parts) is that of the caller, i.e. the externally owned account that triggered the transaction. $\mathsf{CDS}_j = \mathsf{CALLDATA\_SIZE}_j$ is the size (in bytes) of the *unpadded* transaction call data of transaction number $j$. Transaction that don't have any call data (e.g. contract deployment transactions) occupy a single row with all fields set to zero (except for TXNUM and ADDR). For transactions that have call data, INDEX counts up from 0 to $N_j$, where $N_j + 1 = \lceil \mathsf{CDS}_j/16 \rceil$. The values contained in TXCD are the limbs that make up the padded transaction call data. Padding is up to the nearest multiple of 16 which is $\geq \mathsf{CDS}_j$.

**Initialization code:** a table of *padded* initialization code (i.e. *init code*)

| TXNUM | $\mathsf{ADDR}^{\mathsf{hi}}$ | $\mathsf{ADDR}^{\mathsf{lo}}$ | CS | INDEX | $\mathsf{LACS}^{\mathsf{hi}}$ | $\mathsf{LACS}^{\mathsf{lo}}$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | $\mathtt{addr}_2^{\mathsf{hi}}$ | $\mathtt{addr}_2^{\mathsf{lo}}$ | $\mathsf{CS}_2$ | 0 | $\times$ | $\times$ |
| 2 | $\mathtt{addr}_2^{\mathsf{hi}}$ | $\mathtt{addr}_2^{\mathsf{lo}}$ | $\mathsf{CS}_2$ | 1 | $\times$ | $\times$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| 2 | $\mathtt{addr}_2^{\mathsf{hi}}$ | $\mathtt{addr}_2^{\mathsf{lo}}$ | $\mathsf{CS}_2$ | $M_2$ | $\times$ | $\times$ |
| 5 | $\mathtt{addr}_5^{\mathsf{hi}}$ | $\mathtt{addr}_5^{\mathsf{lo}}$ | $\mathsf{CS}_5$ | 0 | $\times$ | $\times$ |
| 5 | $\mathtt{addr}_5^{\mathsf{hi}}$ | $\mathtt{addr}_5^{\mathsf{lo}}$ | $\mathsf{CS}_5$ | 1 | $\times$ | $\times$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| 5 | $\mathtt{addr}_5^{\mathsf{hi}}$ | $\mathtt{addr}_5^{\mathsf{lo}}$ | $\mathsf{CS}_5$ | $M_5$ | $\times$ | $\times$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |

where $\mathsf{CS} = \mathsf{CODESIZE}$ is the size (in bytes) of the *unpadded* init code. The address (which is separated into high and low parts) is that of the caller, i.e. the externally owned account that triggered the transaction. INDEX counts up from 0 to $M_j$ where $M_j + 1 = \lceil \mathsf{CS}_j/32 \rceil$. Recall that recall that $\mathsf{LACS}^{\mathsf{hi}}$ is short for LEFT_ALIGNED_CODESUFFIX_HIGH and $\mathsf{LACS}^{\mathsf{lo}}$ is short hand for and LEFT_ALIGNED_CODESUFFIX_LOW respectively. $\mathsf{LACS}^{\mathsf{hi}}$ and $\mathsf{LACS}^{\mathsf{lo}}$ are limbs of (padded) initialization code; together they make up whole EVM word's worth of (padded) initialization code.

**Logs:** **TODO: settle the precise structure of this table** a table containing, for every log:

- the transaction number,
- its log index in the current transaction,
- its log parameter $(0, 1, 2, 3, 4)$,
- the topics,
- padded log data,
- the length of the log data;

*Remark* 1. Note the similarity between the commitments to call data and initialization codes. Yet we still ask that the verifier generate two separate commitments. The reason for this is that we will want to prove inclusion of the full initialization code commitment into the ROM. Mixing call data and init codes would make this more cumbersome. Doable, for sure, but more cumbersome. And would likely require us to introduce more exo columns to make room for more imported columns.

*Remark* 2. **TODO** Actually it is preferable to exclude "empty" rows from the init code commitment, otherwise we are in trouble if we want to do a full inclusion into ROM.

*Remark* 3. **TODO** The row with 0's, do we keep if for init codes ? I think we may have to. And it is good that such a row exists in the implementation of the ROM. But we then must also add it to the ROM spec.

### 1.2 Exogenous data shape

| Exogenous data source and flags | | | $\mathsf{X}^{\mathsf{hi}}$ | $\mathsf{X}^{\mathsf{lo}}$ | X1 | X2 | X3 | X4 |
|---|---|---|---|---|---|---|---|---|
| ROM | 0 | 0 | $\mathsf{LACS}^{\mathsf{hi}}$ | $\mathsf{LACS}^{\mathsf{lo}}$ | $\mathsf{ADDR}^{\mathsf{hi}}$ | $\mathsf{ADDR}^{\mathsf{lo}}$ | INIT | $\emptyset$ |
| LOG | 0 | 1 | $\mathsf{LOG}^{\mathsf{hi}}$ | $\mathsf{LOG}^{\mathsf{lo}}$ | LOG_NUM | LOG_INDEX | $\emptyset$ | $\emptyset$ |
| STACK | 1 | 0 | $\mathsf{VAL}^{\mathsf{hi}}$ | $\mathsf{VAL}^{\mathsf{lo}}$ | OFFSET | SIZE | RET_OFFSET | RET_SIZE |
| TXCD | 1 | 1 | $\mathsf{TXCD}^{\mathsf{hi}}$ | $\mathsf{TXCD}^{\mathsf{lo}}$ | TXNUM | CDS | INDEX | $\emptyset$ |

Recall that recall that $\mathsf{LACS}^{\mathsf{hi}}$ is short for LEFT_ALIGNED_CODESUFFIX_HIGH and $\mathsf{LACS}^{\mathsf{lo}}$ is short hand for and LEFT_ALIGNED_CODESUFFIX_LOW respectively.

## 2 Columns

1. EXO_STAMP: stamp column of the public data info module; starts at zero, may stay there for a few rows; as soon as nonzero it increments by one with every row; abbreviated to EXO□;
2. SHA#: counts the number of `SHA3` instructions that took place in a batch;
3. LOG#: counts the number of `LOG0-LOG4` instructions that took place in a batch;
4. TX#: counts the number of transactions that are in a batch;
5. $\langle\#\rangle$: imported column; depending on the instruction, contains either SHA#, LOG# or TX#;
6. $\langle$ADDR$\rangle$
7. $\langle$EXO_IS_SHA3$\rangle$: imported binary flag that lights up for `SHA3` instructions; abbreviated to $\langle$X_SHA3$\rangle$;
8. $\langle$EXO_IS_LOG$\rangle$: imported binary flag that lights up for `LOG0-LOG4` instructions; abbreviated to $^\Diamond$X_LOG;
9. $\langle$EXO_IS_ROM$\rangle$: imported binary flag that lights up for temporarily successful deployments; abbreviated to $\langle$X_ROM$\rangle$;

## 3 Constraints

1. $\mathsf{EXO}\square_0 = 0$
2. IF $\mathsf{EXO}\square_i = 0$ THEN the entire row is zero
3. IF $\mathsf{EXO}\square_i \neq 0$ THEN
   (a) $\mathsf{EXO}\square_i = 1 + \mathsf{EXO}\square_{i-1}$
   (b) $\langle \mathsf{EXO\_IS\_LOG} \rangle_i + \langle \mathsf{EXO\_IS\_SHA3} \rangle_i + \langle \mathsf{EXO\_IS\_ROM} \rangle_i = 1$;
   in other words, precisely one of the flags is on; note that $\langle \mathsf{EXO\_IS\_LOG} \rangle$, $\langle \mathsf{EXO\_IS\_SHA3} \rangle$ and $\langle \mathsf{EXO\_IS\_ROM} \rangle$ are imports of binary flags;
   (c) $\mathsf{LOG\#}_i = \mathsf{LOG\#}_{i-1} + \langle \mathsf{EXO\_IS\_LOG} \rangle_i$
   (d) $\mathsf{SHA\#}_i = \mathsf{SHA\#}_{i-1} + \langle \mathsf{EXO\_IS\_SHA3} \rangle_i$
   (e) $\mathsf{DEP\#}_i = \mathsf{DEP\#}_{i-1} + \langle \mathsf{EXO\_IS\_TXCD} \rangle_i$
   (f) the constraint below enforces that $\langle \# \rangle_i$ (which is imported from the stack) contains the correct number: log number, sha number or deployment number.

$$\begin{aligned}
\langle \# \rangle_i \quad = \quad & \langle \mathsf{EXO_IS_LOG} \rangle_i \cdot \mathsf{LOG\#}_i \\
& +\langle \mathsf{EXO_IS_SHA3} \rangle_i \cdot \mathsf{SHA\#}_i \\
& +\langle \mathsf{EXO_IS_ROM} \rangle_i \cdot \mathsf{INIT\#}_i \\
& +\langle \mathsf{EXO_IS_DEP} \rangle_i \cdot \mathsf{TX\#}_i
\end{aligned}$$