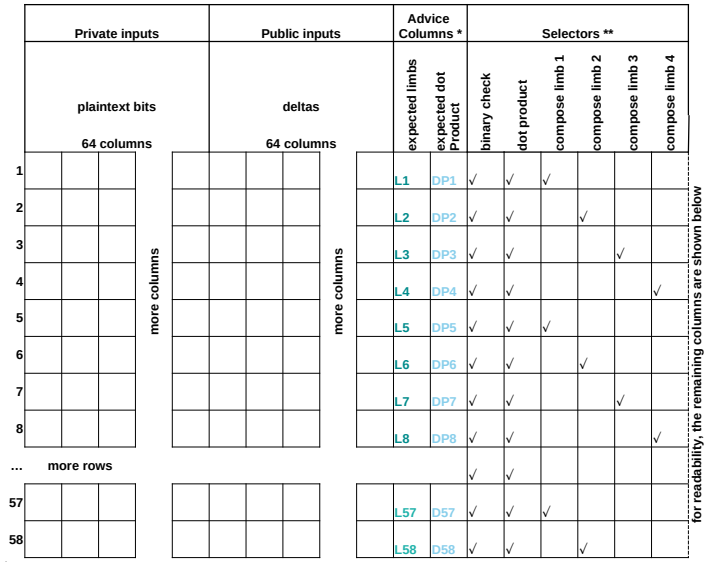


Description of computation which is activated when a selector is enabled ✓ :

binary check: asserts that each value in "plaintext bits" on this row is either 0 or 1

compose limb 1: composes "plaintext bits" on this row into an integer, shifts it left by 192, asserts that the result equals the "expected limbs" cell

compose limb 2: composes "plaintext bits" on this row into an integer, shifts it left By 128, asserts that the result equals the "expected limbs" cell

compose limb 3: composes "plaintext bits" on this row into an integer, shifts it left By 64, asserts that the result equals the "expected limbs" cell

compose limb 4: composes "plaintext bits" on this row into an integer, Asserts that the result equals the "expected limbs" cell

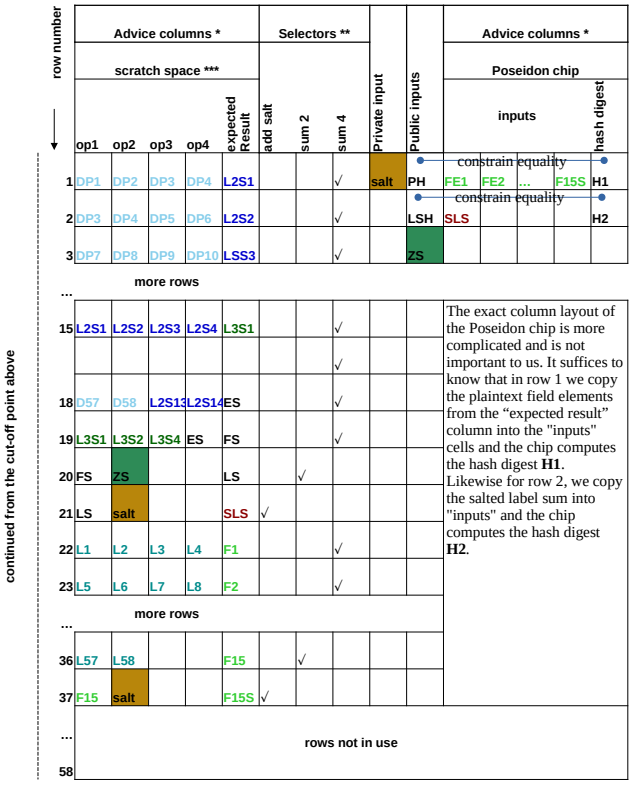
dot product: computes a dot product of all "plaintext bits" and all "deltas" on this Row, asserts that the result equals the "expected dot product" cell

(see **Selectors** columns below)

add salt: left-shift op1 cell by 125 (the size of the salt) and adds op2 to the shifted value, asserts that the result equals the "expected result" cell

sum 2: adds op1 and op2 together, asserts that the result equals the "expected result" cell

sum 4: adds op1, op2, op3, and op4 together, asserts that the result equals the "expected result" cell



The exact column layout of the Poseidon chip is more complicated and is not important to us. It suffices to know that in row 1 we copy the plaintext field elements from the "expected result" column into the "inputs" cells and the chip computes the hash digest **H1**. Likewise for row 2, we copy the salted label sum into "inputs" and the chip computes the hash digest **H2**.

* Advice columns can be thought of as "containing intermediate values" of in-circuit computations.

** When a selector is enabled (✓) on a row, it activates specific computation for that row

*** We to copy cells into scratch space because in halo2 it is much cheaper to perform computations on values which are placed in the same row. Computations on values in the same column are significantly more expensive

Explanation:

1. Computing salted label sum

Rows 1 thru 14. We copy 56 cells (out of 58) of the "**expected dot product**" column in chunks of 4 and find the sum of each chunk. We call each sum "level 2 sum" (the original 56 cells are considered level 1): **L2S1** thru **L2S14**

Rows 15 thru 17. We copy 12 (out of 14) **L2** sums from the "**expected result**" column in chunks of 4 and find the sum for each chunk. We call each sum "level 3 sum": **L3S1, L3S2, L3S3**

Row 18. We copy 2 remaining values from the "**expected dot product**" column and 2 remaining **L2** sums from the from the "**expected result**" column and find their sum. The sum is called "extra sum": **ES**

Row 19. We copy 3 **L3** sums and the "extra sum" **ES** from the "**expected result**" column and find their sum. The result is called the "final sum": **FS**

Row 20. We copy the final sum **FS** from the "**expected result**" column and the zero_sum **ZS** from the "**Public input**" column and find their sum. The result is the "label sum": **LS**

Row 21. We copy the label sum **LS** from the "**expected result**" column and the **salt** from the "**Private input**" column and compute the "salted label sum": **SLS**

2. Summing the limbs to obtain the plaintext field elements

Rows 22 thru 35. We copy 56 cells (out of 58) of the "**expected limbs**" column in chunks of 4 and find the sum of each chunk. Each sum is a plaintext field element: **F1** thru **F14**

Row 36. We copy 2 remaining values from the "**expected limbs**" column and find their sum. The result is the 15-th plaintext field element: **F15**

Row 37. We copy **F15** from the "**expected result**" column and the **salt** from the "**Private input**" column and compute the "field element 15 salted": **F15S**

3. Poseidon hashes and constraining equality

We provide field elements **FE1** thru **F15S** as inputs to the Poseidon chip. The chip outputs the hash digest **H1**. Likewise, we provide **SLS** as an input to the Poseidon chip and the output is **H2**.
Using halo2's internal mechanism, we **constrain (i.e. we assert) equality** between the public inputs and **H1** and **H2**