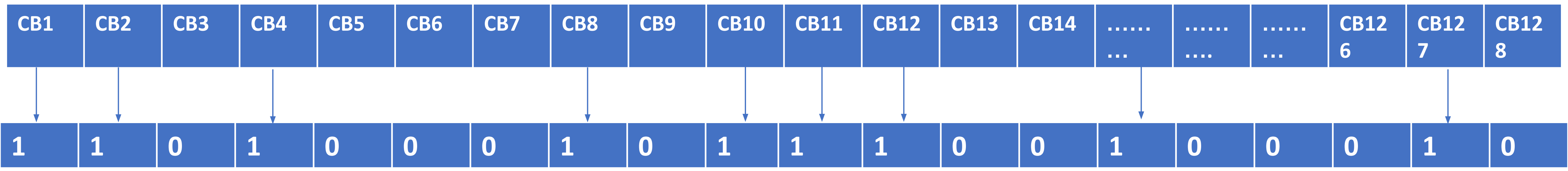
- 128 codebooks are used in tokenization process.
- Assume, each column is a codebook.
- CB1(codebook 1) to CB128(codebook 128).

CB1	CB2	CB3	CB4	CB5	CB6	…………	……………	CB127	CB128
1	7	7	7	.	.	.	.	4	6
5	0	1	2	.	.	.	.	2	8
6	5	6	8	.	. .	.	.	1	7
4	4	3	3	.	.	.	.	5	3
0	1	5	6	.	.	.	.	0	5
9	8	8	9	.	.	.	.	6	0
8	9	0	1	.	.	.	.	8	4
2	6	4	0	.	.	.	.	9	1
3	3	2	5	.	.	.	.	3	2
7	2	0	4	.	.	.	.	7	9

- 128 reverse codebooks are used in de-tokenization process.
- Assume, each column is a reverse codebook.
- RCB1(reverse codebook 1) to RCB128(reverse codebook 128).

RCB1	RCB2	RCB3	RCB4	RCB5	RCB6	....	....	RCB127	RCB128
4	1	6	7	.	.	.	.	4	5
0	4	1	6	.	.	.	.	2	7
7	9	8	1	.	. .	.	.	1	8
8	8	3	3	.	.	.	.	8	3
3	3	7	9	.	.	.	.	0	6
1	2	4	8	.	.	.	.	3	4
2	7	2	4	.	.	.	.	5	0
9	0	0	0	.	.	.	.	9	2
6	5	5	2	.	.	.	.	6	1
5	6	9	5	.	.	.	.	7	9

Codebook column selection process



➢ In above picture, 2nd row is a bit sequence of 128 bit key. 1st row refers to the columns of the codebook.

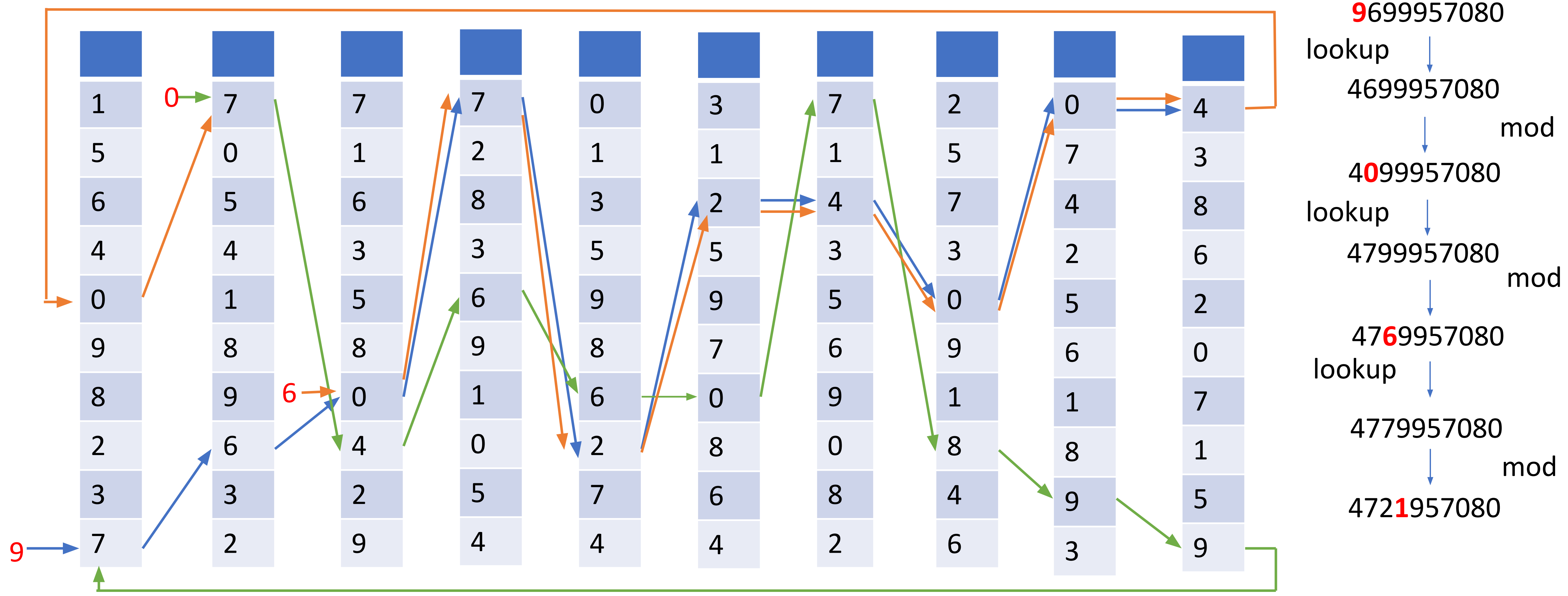
➢ Corresponding to set bits, columns are chosen from the codebook. These set of columns together forms an intermediate codebook at runtime, which is used in tokenization process.

➢ For each different key, there will be different set of columns. It means, for each unique key we can generate a unique codebook at runtime.

Tokenization flow for Input : 9699957080

Participating codebooks are given below, they are selected on the basis of set bits of key.

Tokenization process can employ multiple rounds(left to right) and then (right to left).



Tokenization flow Cont...

- ➢ 1st char start tokenizing from column 1 to column N.
- ➢ 2nd char start tokenizing from column 2 to N to 1.
- ➢ 3rd char start tokenizing from column 3 to N to 2. and so on.

Codebook sequence used for 1st char :

CB1-> CB2-> CB3-> CB4-> CB5-> CB6-> CB7-> CB8-> CB9-> CB10

Codebook sequence used for 2nd char :

CB2-> CB3-> CB4-> CB5-> CB6-> CB7-> CB8-> CB9-> CB10->CB1

Codebook sequence used for 3rd char :

CB3-> CB4-> CB5-> CB6-> CB7-> CB8-> CB9-> CB10->CB1->CB2

- ➢ It can be visualize as codebook itself is rotated right for each next input.

Tokenization flow Cont...

Codebook rotation for input : 9699957080

forward lookup---->	<----backward lookup
9 : 0 1 2 3 4 5 6 7 8 9	9 : 9 8 7 6 5 4 3 2 1 0
6 : 1 2 3 4 5 6 7 8 9 0	6 : 8 7 6 5 4 3 2 1 0 9
9 : 2 3 4 5 6 7 8 9 0 1	9 : 7 6 5 4 3 2 1 0 9 8
9 : 3 4 5 6 7 8 9 0 1 2	9 : 6 5 4 3 2 1 0 9 8 7
9 : 4 5 6 7 8 9 0 1 2 3	9 : 5 4 3 2 1 0 9 8 7 6
5 : 5 6 7 8 9 0 1 2 3 4	5 : 4 3 2 1 0 9 8 7 6 5
7 : 6 7 8 9 0 1 2 3 4 5	7 : 3 2 1 0 9 8 7 6 5 4
0 : 7 8 9 0 1 2 3 4 5 6	0 : 2 1 0 9 8 7 6 5 4 3
8 : 8 9 0 1 2 3 4 5 6 7	8 : 1 0 9 8 7 6 5 4 3 2
0 : 9 0 1 2 3 4 5 6 7 8	0 : 0 9 8 7 6 5 4 3 2 1

Tokenization flow cont...

- ➢ It means, for each input position, order of flow is different.
- ➢ There is no one to one mapping. Means 777 will not give output like 444.
- ➢ There is one to one fix mapping for each position.

  For example:
  Input 777 can be tokenized to 965 depending on the codebook mapping .
  7 at index 1 is mapped to 9.
  7 at index 2 is mapped to 6.
  7 at index 3 is mapped to 5.

- ➢ Though position wise fix mapping is also a very weak tokenization. It has been addressed in next slide.

Tokenization flow cont...

input[] = { 777 };

Perform left to right tokenization.

step1: input[0] ---tokenize to---> 9 ⇒ 977

input[0] = input[0] modadd input[1] ⇒ 677

step2: input[1] ---tokenize to---> 6 ⇒ 667

input[1] = input[1] modadd input[2] ⇒ 637

step3: input[2] ---tokenize to---> 5 ⇒ 635

Perform right to left tokenization as we did above for left to right.

It's a kind of chaining is done which is hiding the information about one to one position wise fix mapping. But mapping of last character is still visible. When we perform next round right to left tokenization, mapping of last character will also be hidden.

Chaining

- Chaining is achieved by mod N +/- operation. N is the base of input.
- Modulo addition is done while tokenizing.
- Modulo minus is done while de-tokenizing.
- Chaining and reverse lookup is done to achieve the complete random   different output for very similar input data.

Advantages

- Here codebook can be compared with s-box and encryption algorithm. The way encryption algorithm is known to the world, but if the key secure, hard to decrypt or guess the data/key/pattern. Similarly here even if codebook is known, hard to decrypt or guess the data/key/pattern.
- Smaller codebooks. Less memory footprint. More caching. Better performance during lookup.
- Easy to implement for any range of input from ASCII to UNICODE.
- Single code book will behave very different with each individual key. No need to have different code book for different columns of table having same data type. Just use the different key with same code book.