An Efficient Parallel Implementation of Impossible-Differential Cryptanalysis for Five-Round AES-128

Debranjan Pal
Dishank Agrawal
Prof. Abhijit Das
Prof. Dipanwita Roy Chowdhury

IIT Kharagpur

SPACE 2019

Outline

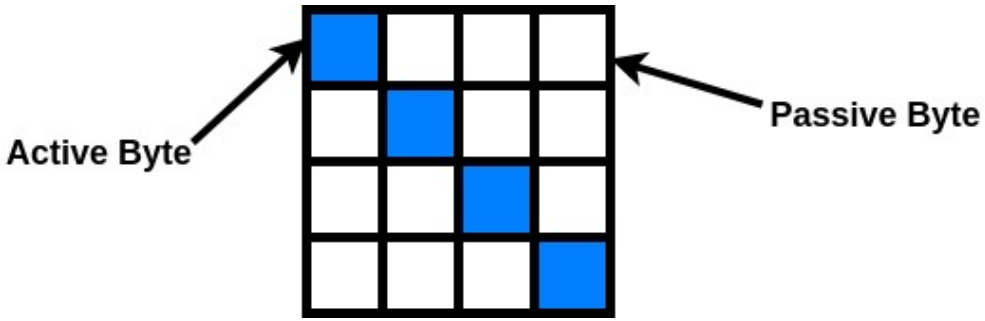
Outline

Impossible Differential Cryptanalysis

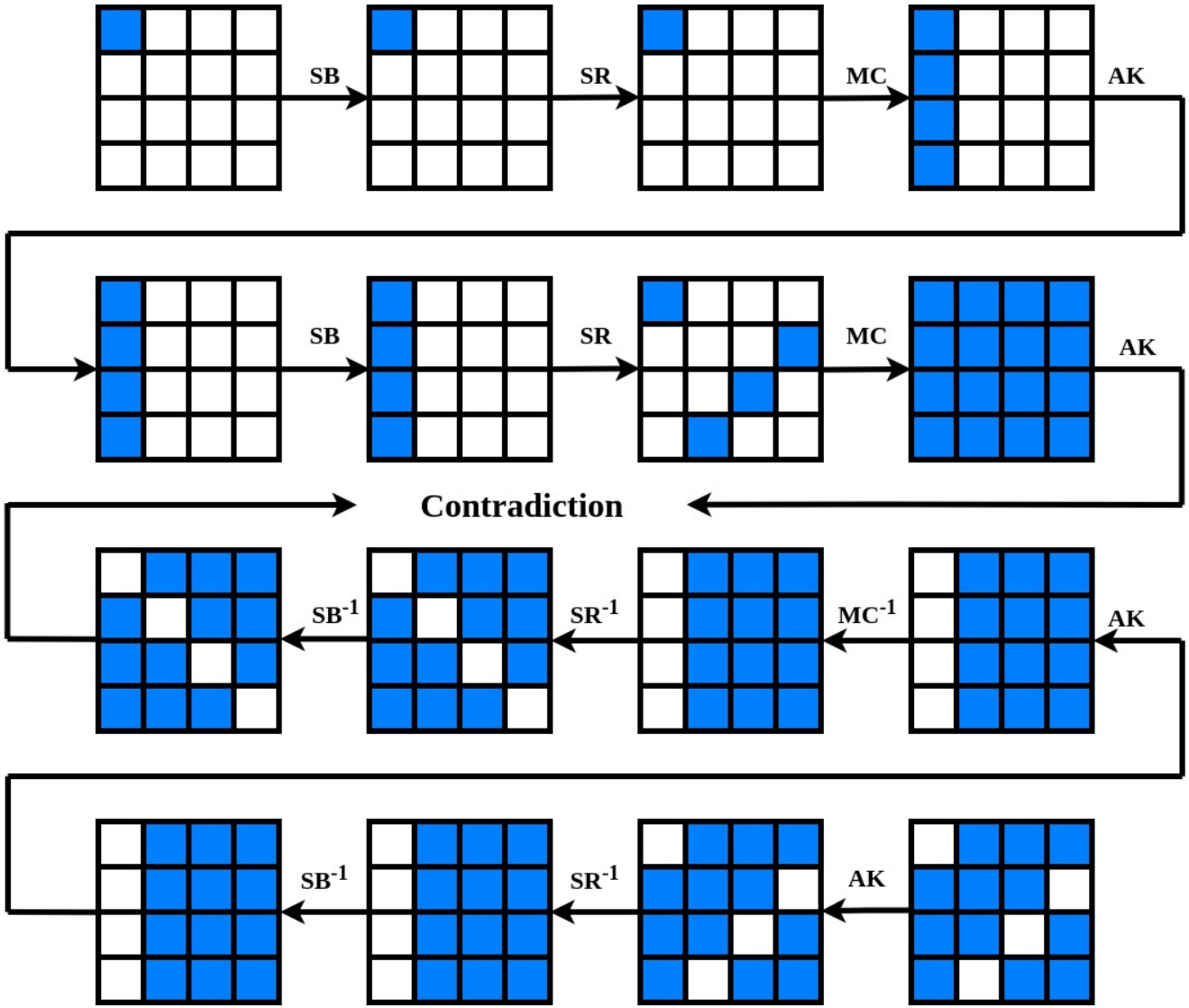
- Impossible differential cryptanalysis tries to exploit differentials in the cipher path, that can never occur.
- The subkeys for which this incident occurs can be eliminated, thereby reducing the keyspace.
- The correct keys never get eliminated.
- With the reduced keyspace we can do a bruteforce search to get the correct key.

Active Bytes Vs Passive Bytes

- **Active Bytes:** In plaintext or ciphertext state the bytes with non-zero differences (White box).
- **Passive Bytes:** In plaintext or ciphertext state the bytes with zero differences (Filled box).



4-Round Path of IDC



Outline

Possible Plaintext and Ciphertext Diagonals

- **Chosen Plaintext Pairs Vs Desired Pairs**

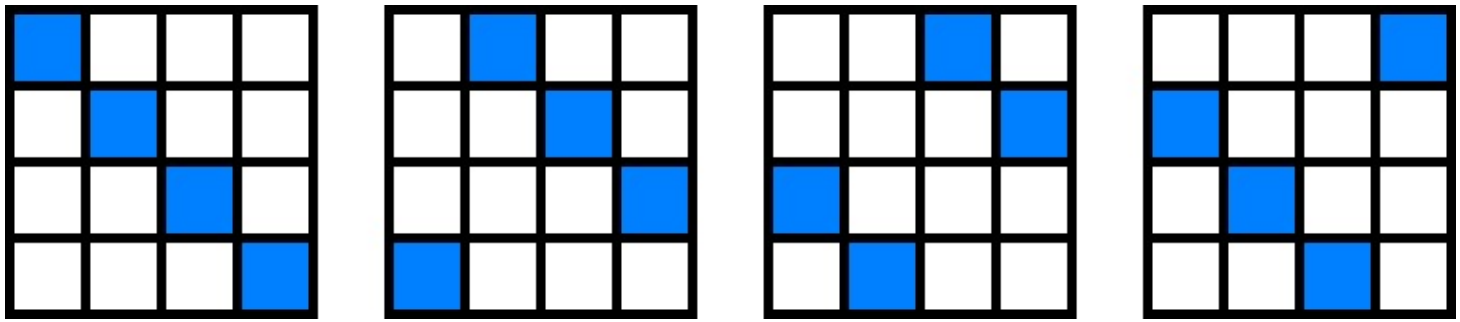


Figure: Plaintext Diagonals

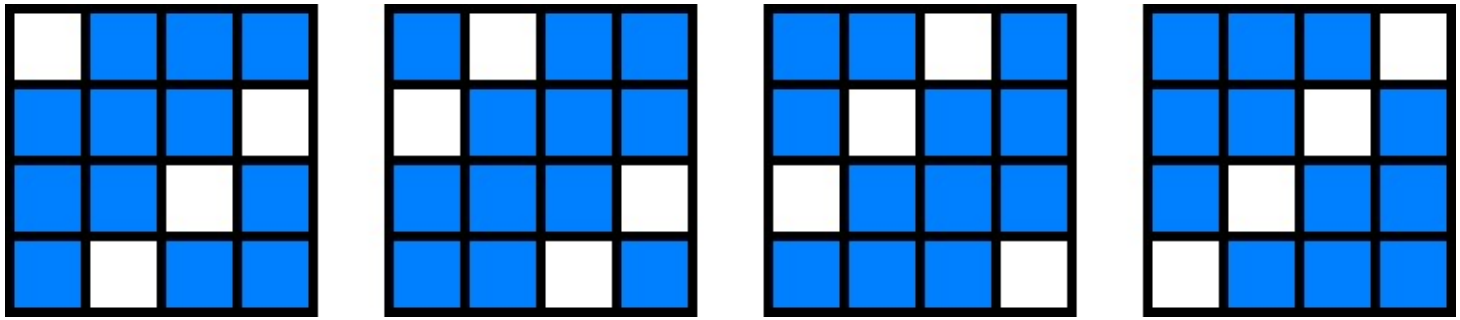
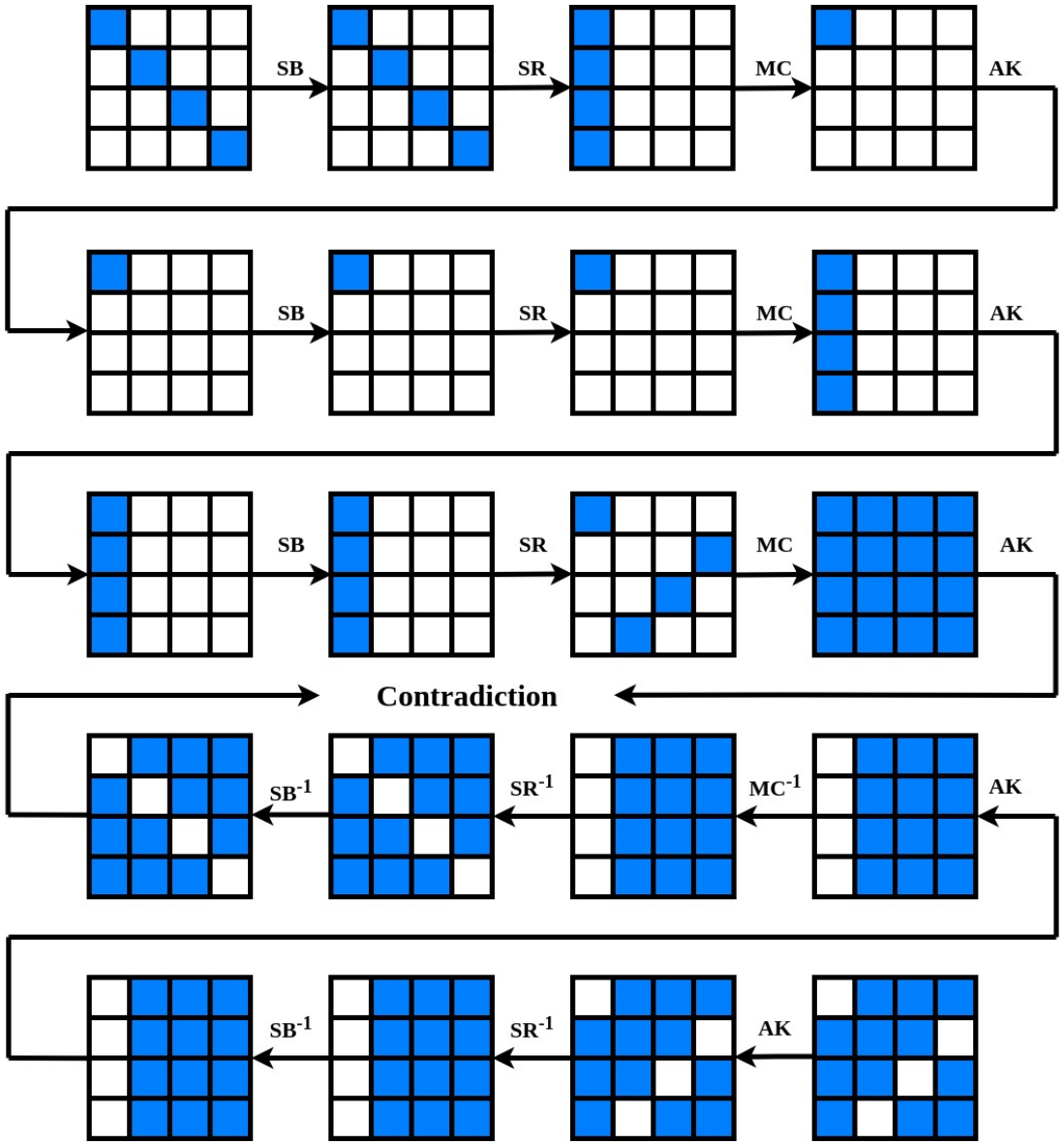
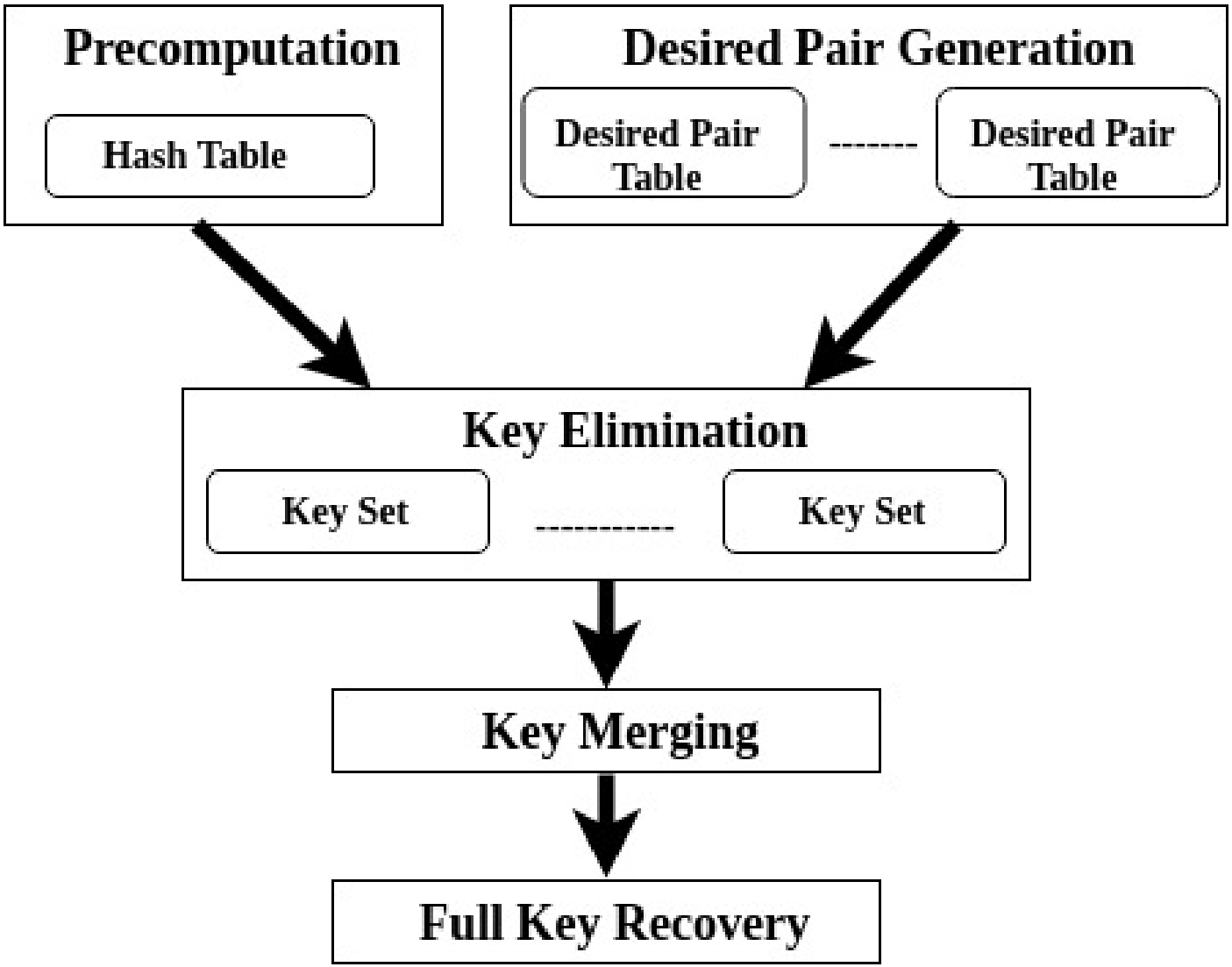


Figure: Ciphertext Diagonals

5-Round Path of IDC



Phases of IDC Implementation



Outline

Precomputation of Tables

- **DesiredPairTable** Stores plaintexts indexed by corresponding to the ciphertext (generated after five rounds of encryption).

- **HashTable** After the one round decryption of pair Q_1 , Q_2 if we get R_1 , R_2 then HashTable stores 32 bit values of R_1 at index $\Delta R = R_1 \oplus R_2$.

Hash Table Generation

Algorithm 1 FormHashTable

1: $HashTable \leftarrow Empty$
2: **for** tuples (w, w', x, y, z) assigned to the worker node, **do**
3: $Q_1 \leftarrow$InitializeColumn(w, x, y, z)
4: $Q_2 \leftarrow$InitializeColumn(w', x, y, z)
5: $R_1 \leftarrow SB^{-1}(SR^{-1}(MC^{-1}(Q_1)))$
6: $R_2 \leftarrow SB^{-1}(SR^{-1}(MC^{-1}(Q_2)))$
7: $r_1 = \text{Extract}_{32}(R_1)$
8: $\delta_r = \text{Extract}_{32}(R_1 \oplus R_2)$
9: Append$(Hashtable[\delta_r],\ r_1)$
10: **end for**
11: **return**

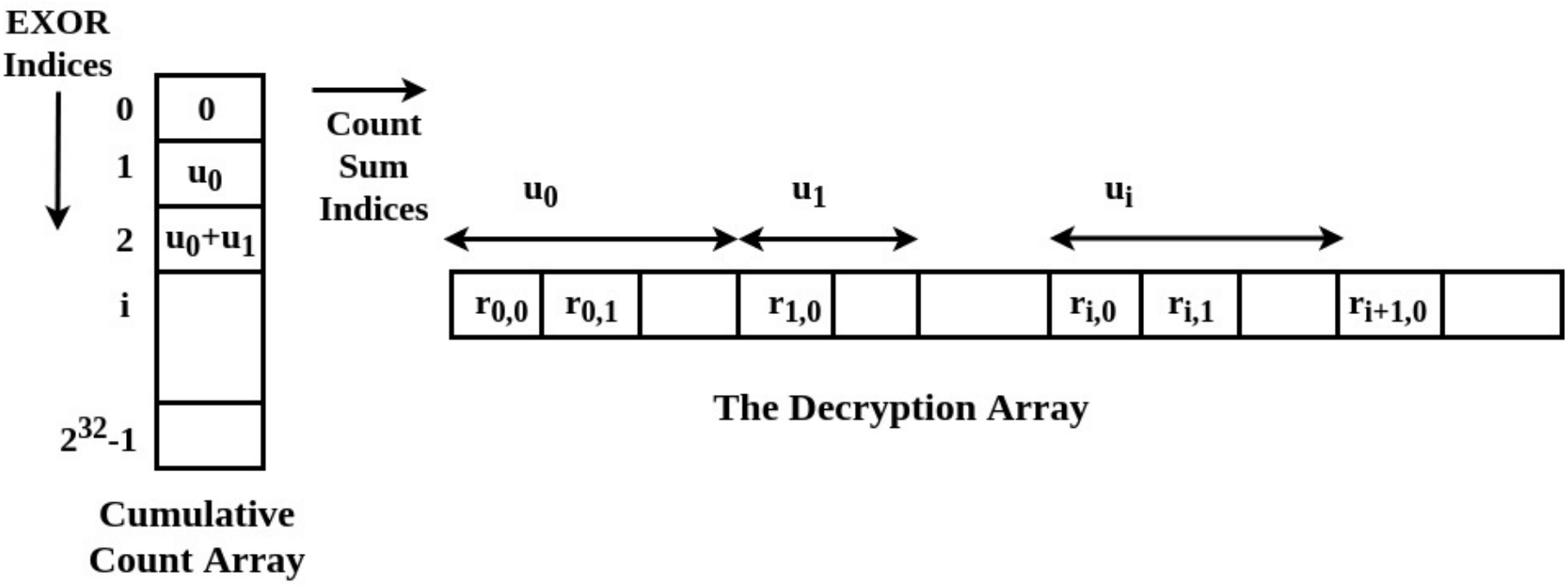
Outline

Hash Table Size Reduction

- The hash table stores an expected number 2^{40} of four byte across 2^{32} indices. So total size 4 TB.
- Instead of using both the pairs $((w, x, y, z), (w', x, y, z))$ and $((w', x, y, z), (w, x, y, z))$, only one pair used by enforcing the condition $w > w'$.
- Now the size reduces from 4 TB to 2 TB.

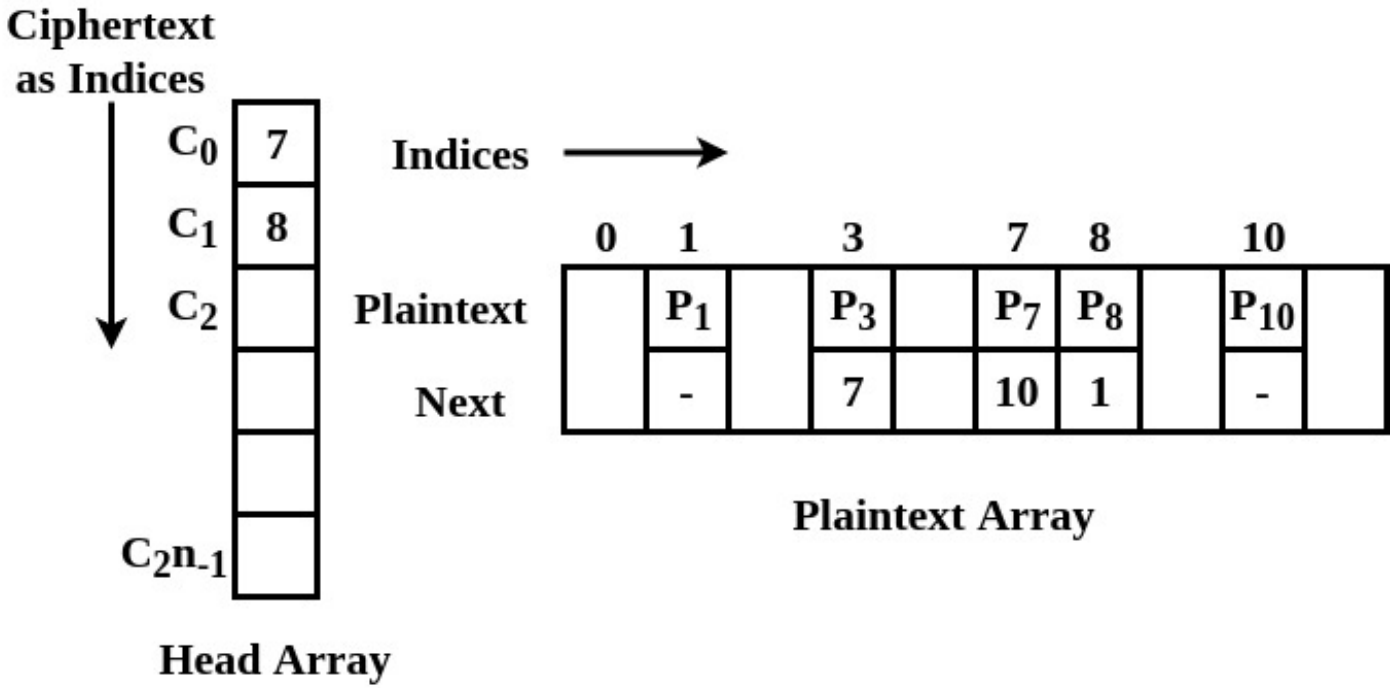
Outline

Hash Table Accessing



- **Decryption array** stores the r values generated after one round of decryption.
- **Cumulative count array** For an index r, let u_r denote the number of entries in the decryption array with the given r value.

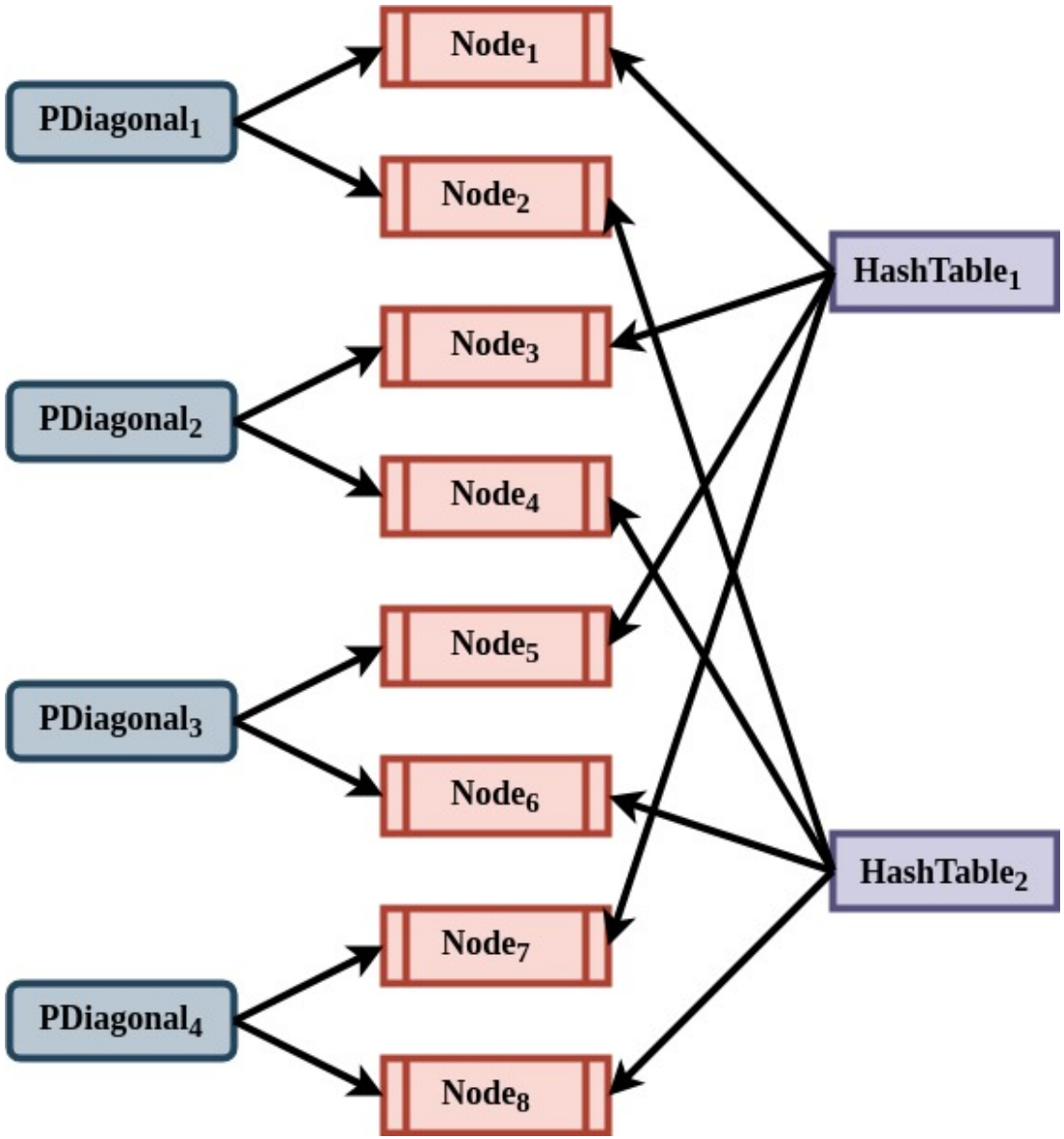
Desired Pair Table Accessing



- The extracted 32 bit plaintext p corresponding to the extracted 32 bit ciphertext c are stored as a linked list in an array called the **plaintext array**.
- The first index of each list is stored in the **head array**, the index of which is 32 bit ciphertext.

Outline

Task Distribution to Worker Nodes



Key Elimination

Algorithm 2 Key Elimination at Worker Node

1: $KeyArr \leftarrow$AllKeys{Initially mark all possible 4-byte keys as valid}
2: **for** each ciphertext diagonal CT_{Diag}, **do**
3: $DPT \leftarrow CreateDesiredPairTable(PT_{Diag}, CT_{Diag})$
4: **for** $c = 0$ to $2^{32} - 1$ **do**
5: **for** $P_1, P_2 \in DPT[c]$ **do**
6: $\delta_r \leftarrow \text{Extract}_{32}(P_1 \oplus P_2)$
7: **if** δ_r belongs to $HashTablePart$ **then**
8: **for** $r \in HashTablePart[\delta_r]$ **do**
9: Delete $r \oplus \text{Extract}_{32}(P_1)$ from $KeyArr$
10: Delete $r \oplus \text{Extract}_{32}(P_2)$ from $KeyArr$
11: **end for**
12: **end if**
13: **end for**
14: **end for**
15: **end for**

Key-Set Merging

- Take the intersection of the two key arrays calculated by a pair of nodes for a single plaintext diagonal.

- As the hash table is divided in two parts, two nodes for a given plaintext diagonal handle these two parts independently.

Full Key Recovery

- Finally, we have around $472 - 508$ partial keys for each of the diagonals.

- Combine them, and make a brute-force search on the reduced key-space to recover the correct full key.

Outline

Memory Requirement at Each Worker Node

Data structure	**Memory requirement**
Hash Table	48 GB
Head Array	16 GB
Plaintext Array	32 GB
Key Array	0.5 GB
Total	96.5 GB

Running Time

- The precomputation of the hash table requires 2^{36} one round encryptions(One time cost).
- For four choosen plaintext diagonals a total of $2^{32} \times 4 = 2^{34}$ five round encryptions needed(One time cost).
- To mount the attack we require 2^{33} desired pairs for a diagonal, so during key elimination total $2^{33} \times 4 \times 2 = 2^{36}$ EXOR calculations required.

Comparison of Performance for Full Key Recovery using IDC

Attack	**Data complexity**	**Memory Requirement**
Biham and Keller	$4 \times 2^{29.5}$	4 TB
Kakarla et al.	4×2^{32}	128.5 GB
This work	4×2^{32}	96.5 GB per node

Comparison With Respect to Time Required for Full Key Recovery

Implementation	**Type**	**System used**	**Time taken**
Kakarla et al.	Sequential	Centralized Server	48 hours
This work	Sequential	Single node, one thread	17 hours 20 minutes
	Distributed	Eight worker and one master nodes, one thread per node	2 hours 10 minutes
	Distributed	Eight worker and one master nodes, 32 threads per node	6.5 minutes

Outline

Conclusions

- In this work, we proposed a way to distribute the data and tasks to eight worker nodes.
- Here we have reduced significantly the running time over previous implementation of Kakarla et al.
- To the best of our knowledge this is the fastest implementation of IDC on five round AES-128.

Thank You

Questions?