



**Deepak Kumar T A**

[deepakkumar.t.a@outlook.com](mailto:deepakkumar.t.a@outlook.com)

# DK Cryptographic Algorithm

## Overview

This is a simple symmetric key cryptographic algorithm, which uses a simple [0-9] Decimal array to transfer data from one machine to another. It can withstand attacks by attackers using Quantum Computers, or any other sophisticated computers.

## Goals


The goal of this algorithm is to send a text from Machine A to Machine B, even if intruders disrupt the communication, with huge computing abilities.

## Specifications

This cryptographic algorithm establishes communication between two Machines A and B. This uses a Decimal array [0-9], to repeatedly transfer data from A to B.

## The basic structure of DK Algorithm

This algorithm is symmetric. Machine A and Machine B hold identical cryptographic keys, with every digit, a unique number from 0 to 9. If Machine B wants to receive data from Machine A, Machine B sends a random number (0 to 9) to Machine A. Machine A performs permutation, with its initial cryptographic key, using the random number, and obtains Modified key A. Machine B, performs the same permutation, with its initial cryptographic key using the same random number and obtains its Modified key B. Now, Machine A and Machine B hold the same modified key. Machine A wants to send a digit 8 to Machine B. It creates a Jumbled Key from Modified



Key A, by keeping the digit position of 8 unchanged. It jumbles all other remaining digits. Machine B receives Jumbled Key from Machine A. Now by comparing Jumbled Key with Modified Key B, we can get the digit 8 in Machine B. Since, all digits except, position 8 are jumbled.

## Structure of Keys

All keys in this algorithm are an array of 10 digit places. It contains a unique number in each of its digit places. The numbers that are allowed in the key places are 0 to 9. Valid key contains a unique digit in each of its places as shown below. Invalid Key contains repeated digits in its places.

### Valid Key

1	0	5	6	9	4	3	8	2	7
---	---	---	---	---	---	---	---	---	---

### Invalid Key

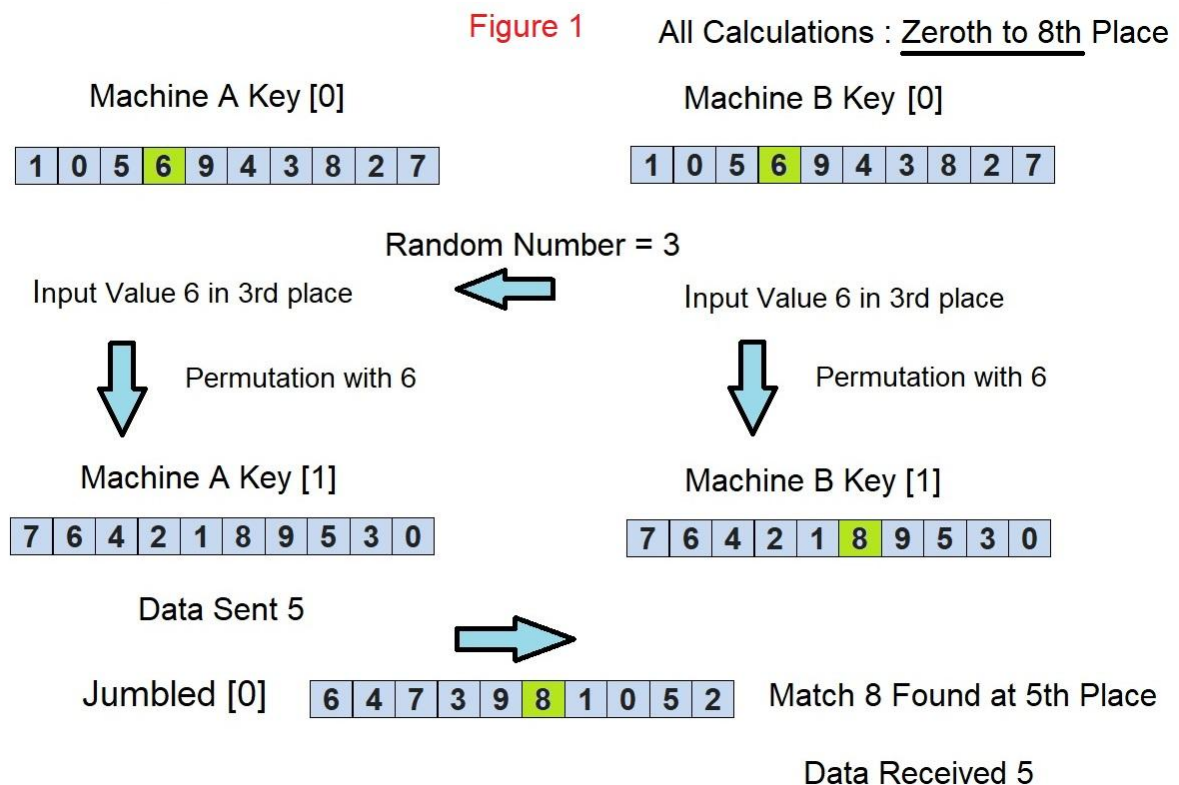
1	0	5	6	9	4	5	8	2	7
---	---	---	---	---	---	---	---	---	---

## Obtaining Session Key from Public Key

First to begin with we need to obtain a Session Key which is symmetric from the Public Key of Machine B. Machine A uses nonce ( a one-time Key ) to communicate with Machine B, and create a Session Key, which uniquely identifies the communicating Machines A and B. Machine B sends a Session Key to Machine A. Distributing Session Keys between Machine A and Machine B, is done by means of DK Cryptographic Algorithm. DK Cryptographic Algorithm gives a secure exchange of Session Key between Machine A and Machine B. Both Machines will have the same Session key.

## Basic DK Cryptographic Algorithm

Let's first understand the basic DK Cryptographic Algorithm, using a single-digit. This is explained figuratively in Figure 1. All Values are from 0 to 9. So a number 3, means it is position 4 in the array. It is **Zero-Based Array Indexing**.



Let's say we are transferring digit 5 from Machine A to Machine B. The Algorithm starts by obtaining symmetric keys on both Machine A Key [0] and Machine B Key [0]. Now Machine B pokes the Machine A, with a Random Number 3. Machine A receives this Random Number 3 and calculates the Input Value. Input Value is obtained from the position of Random Number 3 in Machine A Key [0]. Remember it is **Zero-Based Array Indexing**. Machine A calculates the Input Value. Input Value is found as 6, which is present in the position of Random Number 3 in Machine A Key[0].

Similar to Machine A, Machine B also calculates the Machine B Key[1]. Machine A Key[0] and Machine B Key[0] is the same, also Machine A Key[1] and Machine B Key[1] are the same, because of the Symmetric Keys used in this algorithm. We can get the Input Value 6 from Random Number 3. It is the same for both Machine A and Machine B in every iteration. Both Input Value and Random Number are present in both Machine A and Machine B. The Machine B Key[1] will be calculated in Machine B, with the same Input Value 6, with Machine B Key[0]. Machine B has this Random Number 3 with it because it is Machine B, which created the Random Number 3 on its side and sent it to Machine A.

## Permutation

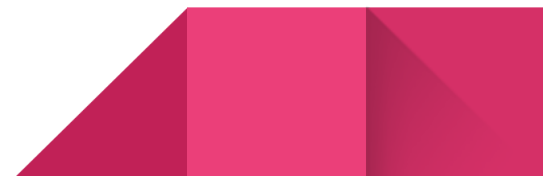
Using this Input Value, Machine A performs a Permutation ( Explanation and Calculation of Permutation will be explained later in the document). Permutation on Machine A Key[0] is done using the Input Value 6, to obtain the Machine A Key[1]. Similarly, Machine B Key[1] is calculated using the same Input Value 6, to obtain Machine B Key[1]. Machine A and Machine B have their own copy of Input Value and both perform Permutation in their own Machines.

## Jumbling the Machine Key Array

Now, to send Data 5 from Machine A to Machine B, we need to calculate Jumbled [0]. Jumbled [0] is calculated on Machine A alone. Jumbled [0] is calculated by altering digits in every position of Machine A Key[1] except in position 5. Jumbled [0] differs with Machine A Key[1] only at position 5. This Jumbled [0] is sent from Machine A to Machine B. Upon receiving Jumbled[0] on Machine B, we need to match Machine B Key[1] and Jumbled[0]. We will find that only at position 5, Machine B Key[1] and Jumbled[0] differ. This position is the actual Data sent from Machine A to Machine B, not the digit present inside the array Machine A Key[1] or Machine B Key[1]. Data 5 is thus sent from Machine A to Machine B.

## Permutation Function

The permutation function that has to be performed by both Machine A and Machine B is as follows. For Machine A, the calculation is as follows:



$\text{MacAInter} = (\text{MacAInput} + j) \% 10;$

$\text{MacAKeys}[i+1][\text{MacAKeys}[i][\text{MacAInter}]] = ((\text{MacAInter} + \text{MacAInput}) \% 10);$

Where,

$j$ , represents the index for the whole Array from 0 to 9

$i$ , represents the Key Number

**MacAInter** - Intermediate Value required while performing Permutation.

**MacAInput** - Input Value ( 6, As per above Example )

**MacAKeys[i]** - Machine A Key[i]

Machine B also performs the same Permutation, with its  $\text{MacBKeys}[i]$ , using  $\text{MacBInput}$  and  $\text{MacBInter}$ .  $\text{MacBInput} = \text{MacAInput}$  and  $\text{MacBInter} = \text{MacAInter}$ , for every iteration.

## DK Cryptographic Algorithm Explained with Example

Now we will get into detailed working of this algorithm, with an example, and understanding the Variables in the Code to run the simulation.

First, we will consider sending the text "Hi", [ *Stored as EncryptText in the Code* ] from Machine A to Machine B. This text is parsed and first "H" is sent from Machine A to Machine B. We can take the ASCII value of H, which has three digits. Let's suppose, consider it as ( 516 ), [ *Stored in array InputValues[ ] in the Code* ] for convenience.

Now we will take the input value of First Digit 5 to be sent from Machine A to Machine B as in Figure 2.

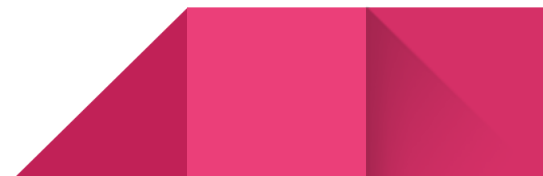
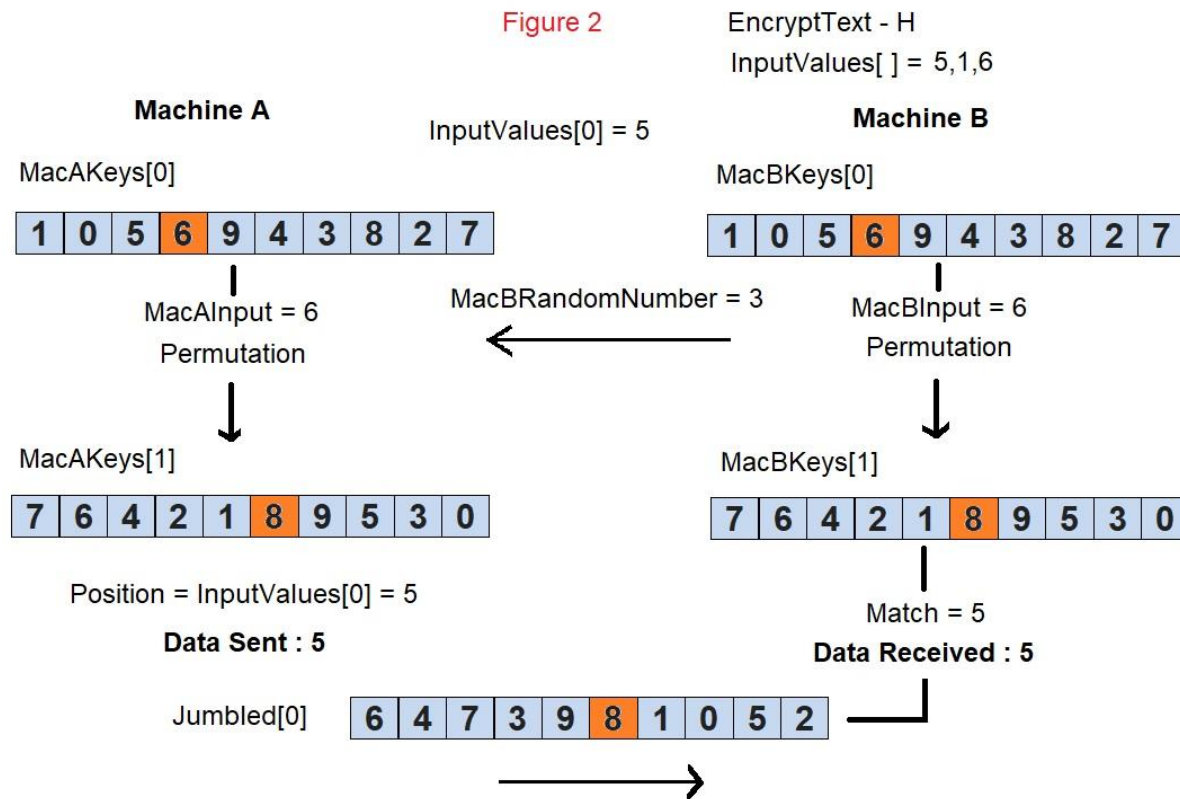


Figure 2



To transfer digit 5 [ *Stored as InputValues [0] in the Code* ], First Machine B pokes Machine A with a Random Number 3 [ *Stored as MacBRandomNumber in the Code* ]. Input Value in Machine A [ *Stored as MacAInput in the Code* ] is 6, which is obtained from Random Number 3 and Machine A Key [0] [ *Stored as MacAKeys [0] in the Code* ]. Input Value 6 is obtained from the position of Random Number 3 [ *Stored as MacARandomNumber in the Code* ] in Machine A Key [0].

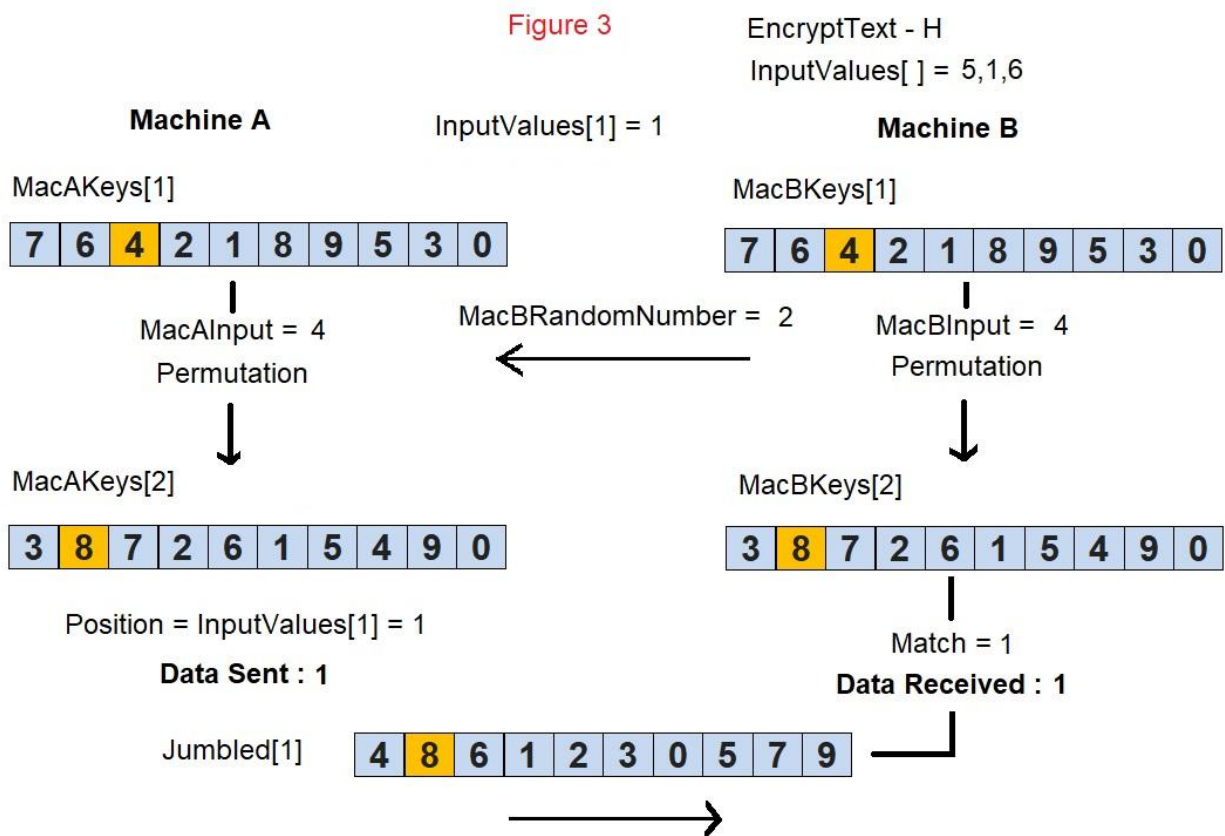
With Input Value 6, we have to do a Permutation ( as explained earlier with the Formula ) on Machine A Key [0] to obtain Machine A Key [1] [ *Stored as MacAKeys[1] in the Code* ]. Now, we get the value Position [ *Stored as Position in the Code* ]. Position holds the Digit 5, which has to be transmitted from Machine A to Machine B. Now we calculate the Jumbled [0] [ *Stored as the array Jumbled [0] in the Code* ]. Jumbled [0] is calculated by jumbling all values of the Machine A Key [1], except the Position of 5. This Jumbled [0] is sent from Machine A to Machine B.

On Machine B, We calculate the Same Permutation on Machine B Key [0] [ *Stored as MacBKeys[0]* ], with the same Input Value 6 [ *Stored as MacBInput in the Code* ]. This Input

Value 6, is obtained from the position of Random Number 3 [ *Stored as MacBRandomNumber in the Code* ].

Now Jumbled [0] is sent from Machine A to Machine B. When we compare Jumbled [0] with Machine B Key [1], we get a Match [ *Stored as Match in the Code* ] only at Position 5. This Position 5 is the Data received by Machine B from Machine A.

Similarly, We send the Next Digit 1 from Machine A to Machine B, as illustrated in Figure 3



Similarly, Digit 6 is also sent from Machine A to Machine B. On Receiving the three digits ( 5,1,6), we calculate Text 'H', from the three digits transferred.

## Acknowledgment and Hash Functions

On receiving Data from Machine A, Machine B acknowledges that it has received the Data. Hash Function is performed on the Input Text and the hash value is sent from Machine A to Machine

B with the same DK Cryptographic algorithm. Hash Value is verified by Machine B, and acknowledgment sent back to Machine A.

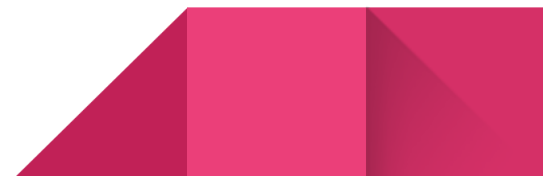
## Handling Error

When an Error value is received by Machine B ( Explained in Cryptanalysis ), Machine B reverts back one Key and performs the same process again. If it again gets an Error Value, reverts back two Keys. It does not reverse back more than 5 consecutive keys ( Design Consideration ) and if it reaches one value after Initial Cryptographic Key. Connection is dropped and a new Session Key is created.

## Cryptanalysis

First, we will see, what happens if someone interrupts the communication and messes with the Data.

- By getting the whole Communication Exchange between Machine A and Machine B, no Information can be derived, because it is always a Jumbled Array and a Random Number.
- By altering the Communication Exchange between Machine A and Machine B, none can be achieved, because alteration of digit value in the Jumbled Array will result in Error. Because, it has to be a valid key, with distinct values from [0-9].
- Even if the intruder alters the Data with a new valid Jumbled Array, none can be achieved. Because When Machine B receives the Jumbled Array, Machine B will compare it with its Machine B Key. If a wrong match is found, the Hash Data ( or Acknowledging every Character Sent, based on Design Consideration ) will be different, when that Data is verified.
- Even if an intruder tries a Brute Force attack on Jumbled Array, Error will be detected, on even one single altered Jumbled array.
- An intruder can't send any data across Machine A and Machine B, because, He does not know, which Jumbled Array to use to send a digit across from Machine A to Machine B.
- Altering Random Number is of no use. Because Mismatch will occur and it corrupts the Data Sent, which can be found on Hash verification.





Now, we will get into different types of attacks that can take place.

## Ciphertext only attacks

As explained earlier, Obtaining Cipher Text Only will be of no use. Because it is always a Jumbled Array of distinct digits from [0-9]. Frequency analysis can not be performed on the Cipher Text, because it is always a Jumbled Array.

## Known Plaintext attacks

Even if we have both Plain Text and Cipher Text, Key can't be detected as it changes every time a Data is sent and there is no clue left in the Cipher Text, that can help to detect the Key.

Chosen Plaintext Attack and Chosen Ciphertext Attack will be of no use, as explained above.

## Man-in-the-Middle Attack

Man in the Middle Attack will be of no use. Since Capturing the Key value in Key Exchange will not happen as the Exchange is secured by DK Cryptographic Algorithm.

## Quantum Computing

Now, we can securely transfer data between Machines, without bothering about any Quantum Computers or any other Super Computers that will ever come. Processing power is helpless with Deepak Kumar's Cryptographic Algorithm.

