Dhirubhai Ambani Institute of Information and Communication Technology

SC-402

Introduction to Cryptography

2-Sided Cone Cryptosystem

Group Members:

Manan Sodha (202003021) Jenish Patel (202003023)

Preet Sheth (202003033) Dhruv Patel (202003034)

Piyush Parmar (202003038)

OVERVIEW :

- Basic Idea For Cryptosystem.
- Encryption Process.
- Decryption Process.
- Software Implementation
- Theoretical Results
- Application on a various field

Idea of Cryptosystem :

- In this project, We have tried to design a new Cryptosystem based on some tree properties (specifically using LCA(Least Common Ancestor)) while doing encryption. (Note: LCA of two nodes is the node that is the lowest (i.e. deepest) node that has both the given nodes as descendants.)

- It's a kind of block cipher but, here we will take a bunch of 2-2 elements and then we will encrypt them respectively.

- It is a symmetric Cryptosystem (where same key can be used to encrypt and decrypt the message), with encryption key changes in each step.

Encryption Process:

- ➤ In each step, we will take two elements from the plain text and we will put the whole diamond path (node1, LCA-tree1(node1,node2), node2, LCA-tree2(node1,node2)) and this will be the encryption of two elements.
- ➤ In the second step, we will convert the whole tree nodes using the function $[y = (a + c)x + (b + d)]$, where c and d are the root nodes of tree1 and tree2 respectively and the a and b are the LCA-tree1(node1,node2) and LCA-tree2(node1,node2).
- ➤ If plain text remains then repeat step 1.

Example:

First let us take an example of an encryption key tree as mentioned in the figure.

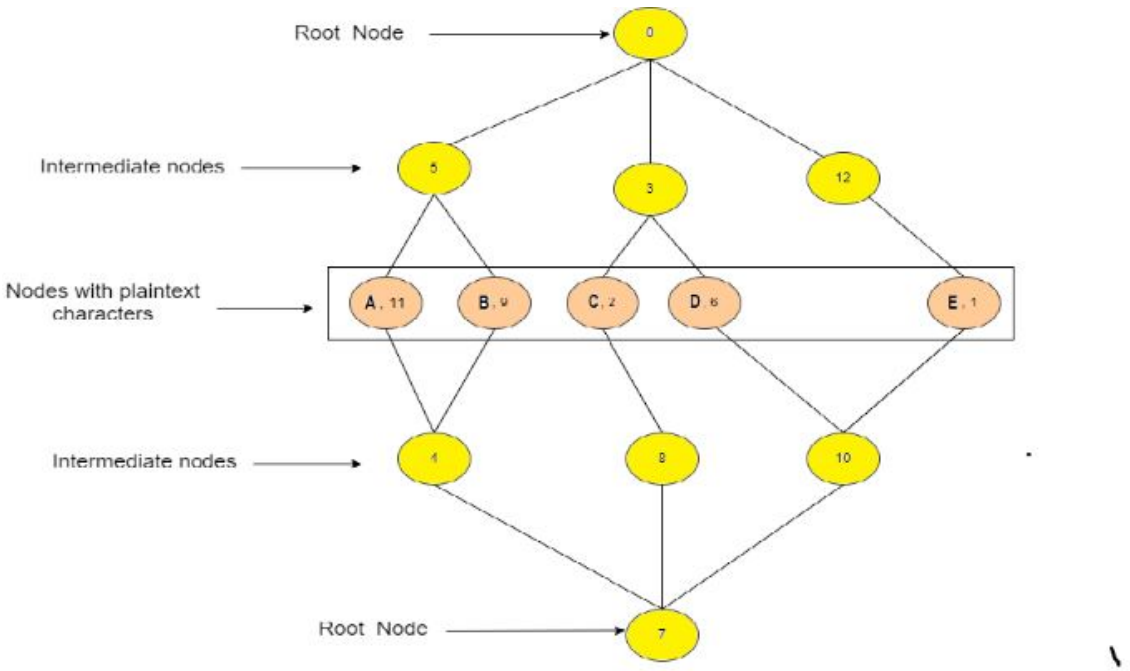

Figure 1: Visulization of Binary Tree

Now let us take the plain text as "abbdae" and see how it is encrypted:

Here, we first define root1→ root of the upper tree, root2→ root of the lower tree, lca1→ Lowest common ancestor of a and b in an upper tree, and lca2 → Lowest common ancestor of a and b in the lower tree.

- First we will divide the text "abbdea" as → ab, bd, ea.
- Secondly, we'll encrypt 'ab', taking the diamond path of 4 nodes : [an LCA-tree1(a,b), b, LCA-tree2(a,b)].
- Encryption of 'ab' [11, 5, 9, 4]. 1
- Then, we will convert the whole tree into y = (root1 + lca1)*x + (root2 + lca2) mod $|C|$ →(Size of tree). Here y = 5x+11.

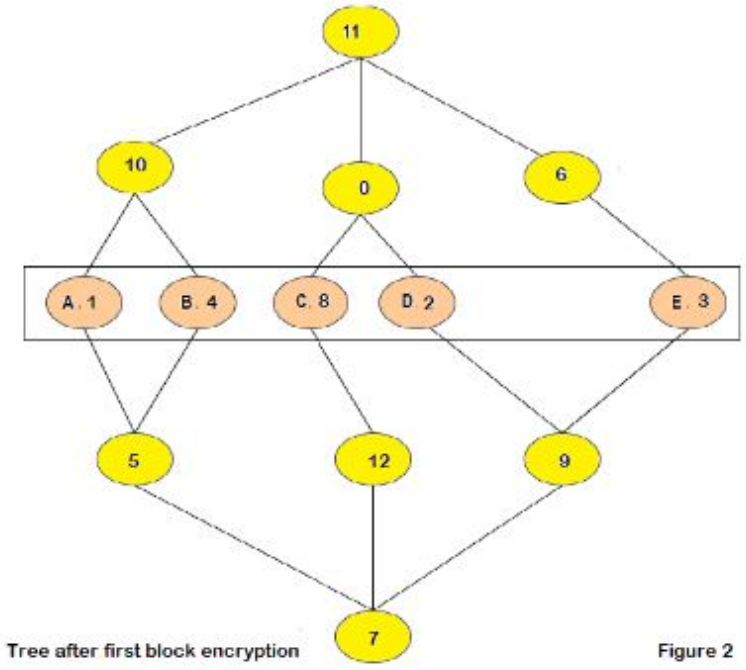

Figure 2: Updated Binary Tree

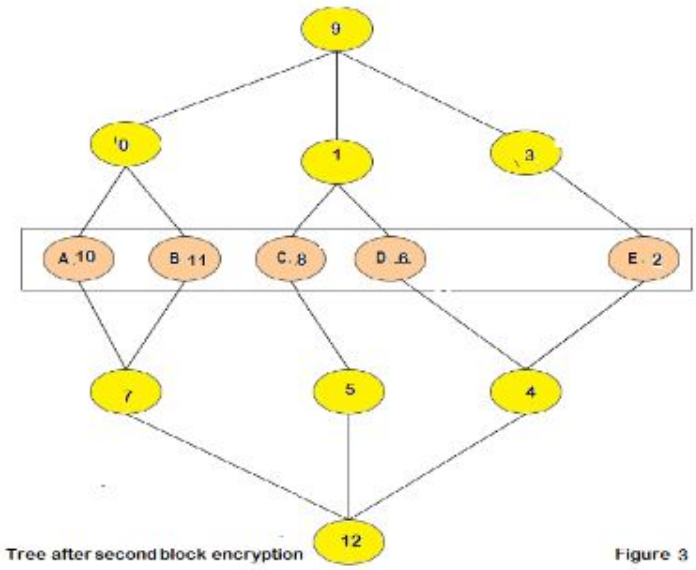

Figure 3: Updated Binary Tree

- Now we'll encrypt 'bd', as taking diamond path [4, 10, 11, 0, 2, 9, 7, 5]. 2

- Then we will convert the whole tree again y=(root1+lca1)*x + (root2+lca2) mod $|C| \rightarrow$(Size of tree). Here y=9x + 1. See Figure 3.

- Lastly we will encrypt 'ea', as [2, 4, 12, 7, 10, 0, 9, 3]. 3

- Therefore, the final cipher text will be [11, 5, 9, 4, 4, 10, 11, 0, 2, 9, 7, 5, 2, 4, 12, 7, 10, 0, 9, 3].

Decryption Process:

At this step, we have the complete initial 2-sided cone and the cipher text.

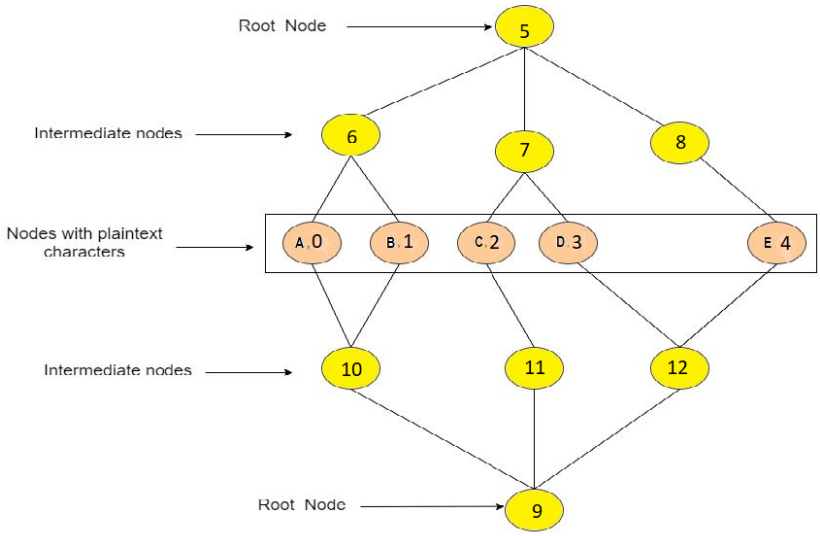

Figure 4: Updated Binary Tree

- Now, the decryption process will have one another array for each block of cipher text, as the blocks are not defined exclusively and we also need to find where each block ends.
- So here, we know the initial permutation of the whole array, i.e., the initial tree. So from that first we define another array "order" which will contain the order of each element of the permutation.
- With the use of an ordered array, we will get the first element of plain text. Now from the parent array of the lower tree, we will also get the last element of the cipher text, thus the block of the cipher will end when that node will arrive in the cipher text.
- And from the height array, we will also get the second plain text node, as in our cycle (which starts from the first plain text node), we will check the height for each node and as it arrives we just got that.
- After knowing the 1st and 2nd plain text characters, we need the LCA(Least Common Ancestor) in upper and lower trees, which we have stored already.

➤ At last, we will get the new permutation because we know all the four needed things(LCA1,LCA2,root1,root2).

This process is continued until we get to the end of the cipher text.

Example:

- Taking the cipher text as : [11, 5, 9, 4, 4, 10, 11, 0, 2, 9, 7, 5, 2, 4, 12, 7, 10, 0, 9, 3].
- This is the initial Tree 4. (tree without permutation), And applying permutation on this tree we get 1. So form this we will make the "order" array from the permutation array which is perm = [11,9,2,6,1,0,5,3,12,7,4,8,10]. The order array is defined as order[perm[i]] = i. Therefore it will be [5,4,2,7,10,6,3,9,11,1,12,0,8].
- Hence, cipher[0]=11, and order[11] = 0. Therefore we have to see the node '0' in the main tree 4.
- Now parent of 0 in the lower tree is 10, as we reach such a node in our cipher text whose order is 10 we will get the endpoint of the first two character ciphers. And, as the order[3]=10, we know that the cipher for first two characters is [11,5,9,4] and we will know the second character by just checking the height of each node as it is at the level of plain text we got the second character.
- Thus, we will get the LCA1,LCA2(as we already stored) and root1 root2 we will re-create the tree. And this process runs again for further elements.

Software Implementation: Code link

Two-sided Tree creation (random tree creation and labeling).

- The main focus of our project is to determine appropriate sizes for the upper and lower trees such that they contain approximately the same number of nodes as the plain text set, while also ensuring that a total graph size is a prime number.
- We will generate a tree using a Random *Prufer* Sequence, which is a unique sequence for all labeled trees and is of length (n-2), with any element between 1 and n present in it. This means that the total number of possible labeled trees is $n^{(n-2)}$.
- Next, we will define the edges, leaf nodes, and adjacency lists for both trees. By running a few depth-first searches (DFS), we will be able to obtain information such as the heights and number of leaf nodes in a subtree.
- Then, we will create the entire Diamond Graph, which will serve as the basis for our encryption process. We have also created functions to print the entire graph in 2D for analytical purposes, although this is not necessary for the encryption process.
- Finally, we will initialize the lowest common ancestor (LCA) values for each pair of plain text sets in both trees.

This tree will be available to both parties.

We perform some pre-computation for both trees to store the height of each node and the Parent of each node. (Just run DFS)

For Encryption and Decryption refer above steps.

Time complexity Analysis:

- $|P|$ size of plain text set. Let $|P| = N$
- $|C|$ total nodes in 2-sided tree.
- The size of the upper and lower tree will be approximately equal to the size of the plain text set thus $|C| \approx 3 * |P|$.Thus $O(|C|) = O(|P|) = O(N)$.
- H $\rightarrow$ height of the 2-sided tree.

It is possible to perform all of these operations within milliseconds even on a single device, without the need for multiple processors.

Pre-computation Time complexity(Whole Graph construction(and storing its feature)) :

- Tree construction is performed using **Prufer sequence O(N)** time.
- Storing the Heights of each node with just 1 DFS takes O(N) time, Similarly storing the number of leaf nodes for each node subtree O(N) time, and the parent of each node in the upper and lower tree takes O(N) time.
- Now we will store the LCA of each pair of nodes from the plain text set and thus the space complexity will be $O(N^2)$ and thus the time complexity for this initial storing is $O(N^2)$.

Thus the overall time complexity for Tree creation with all these features will be $O(N^2)$.

Time Complexity : $O(N^2)$

Space Complexity : $O(N^2)$

Encryption Time complexity : For each pair of blocks we need to store the path from that node to its LCA in both the upper and lower tree, And thus as the maximum height is H, for 2 characters we will require to store the elements of length 4H.

And thus for plain text size M, the time required would be M*2*H. Now as H will be approximately equal to log(N).

Thus Time complexity will be O($Mlog(N)$). Also, the length of cipher text is $O(Mlog(N))$ (space complexity).

Also the tree is relabeled for each 2 characters thus its time complexity is $O(N * M)$.

Time Complexity : $O(N * M)$

Space Complexity : $O(Mlog(N)$

Decryption Time complexity : Similarly the time complexity for decryption mainly depends on the length of the cipher text, which is approximately $O(Nlog(N)$.

Time Complexity : $O(N * M)$

Space Complexity : $O(Mlog(N)$

Theoretical Results:

- As our cryptosystem uses a tree structure for encryption, It will only get cracked if someone has access to the tree.
- There are in total $(n^{(n-2)})$ labelled trees possible with given n.
- It would be extremely hard to determine the correct tree without prior knowledge, making the encryption extremely difficult to crack.

Applications :

Figure 5: Application of cryptosystem

Financial cryptography is used in various areas related to financial transactions, such as:

1) **Online banking:** Can be used to encrypt and protect sensitive data (such as login credentials and financial transactions) during transmission over the internet.

2) **Mobile payments:** Can be used to encrypt and protect sensitive data(such as payment information and personal details) during transmission between mobile devices and the payment gateway.

3) **Digital currencies:** can be used (such as Bitcoin) to securely encrypt and protect transactions and ensure the authenticity and integrity of the blockchain ledger.