Grenze International Journal of Computer Theory and Engineering , June 2016

The Computationally Infeasible: Phniks Algorithm (Cryptography)

Nikhil Phalak[1] and Dr. Vijaya Musande[2]
[1]CSE-JNEC, Aurangabad, India
Email: phniks@gmail.com
[2]HOD, CSE-JNEC, Aurangabad, India
Email: vijayamusande@gmail.com

***Abstract*— In this paper I have given an algorithm (my original work) named as the 'PHNIKS Algorithm' which can be used to encrypt-decrypt data and breaking the cipher-text is computationally infeasible which makes it much strong. I have given proofs of the above claims in this paper and computed the required results. I have also given the brute-force attack and computed the time required to break the cipher-text assuming the highest possible speed today that is 33.86 PFLOPS of TIANHE-2 supercomputer (*source of information mentioned in the references*).**

***Index Terms*— PHNIKS Algorithm, bijective functions, cardinality, PFLOPS, functions, combinatorics, time complexity.**

I. INTRODUCTION

The following paper is the outcome of my original work in Cryptography, more precisely, an algorithm named '**PHNIKS**' algorithm which can be used for encryption-decryption of the message. I was inspired to work on this particular subject while studying the basics of Cryptography. I have jotted down my ideas and played with them to come out with the 'phniks' algorithm. I have analyzed the algorithm and explained every part of it in this report. The analysis part demonstrate its importance. I have used basic concepts from Algorithms and Mathematics, more precisely Combinatorics (basic counting). Mathematical background required is basic knowledge of functions and time complexity. This paper contains the stated algorithm and theory to retrieve permutations for encryption.

II. RELATED WORK OR LITERATURE STUDIES

This paper is my original work. I have used whatever knowledge I have gained while studying the regular courses in the college (like Cryptography and Combinatorics) or school courses (like basics of functions).

III. MOTIVATION

I am passionate about problem solving and playing with nuances of every idea that clicks my mind. This attitude of mine always keeps me self-motivated to work and innovate over any subject I study. I was

Grenze ID: 01.GIJCTE.2.2.25

working with the problem of getting all the permutations of {1,2,…,n} and try to index them to identify and retrieve any permutation as and when required. I worked over this idea of retrieving a permutation given its index for any n and succeeded. This is my innovation. Then I observed the bijection between set of permutations of {1,2,…,n} and all bijective functions from {1,2,…,n} to {1,2,…,n}. I have used this into the 'phniks' algorithm. To summarize, working over the above mentioned problem motivated me to design and play with this algorithm.

IV. PROBLEM DOMAIN

- MATHEMATICS (required : basic knowledge of functions and basic combinatorics).
- ALGORITHMS (required : computation of performance analysis like time complexity).
- CRYPTOGRAPHY (basics).

V. PROBLEM DEFINITION, STATEMENT AND INNOVATIVE CONTENT

❖ **PHNIKS ALGORITHM**:

- *Define a set V of all bijective functions on {1, 2, 3, … , r}.*
- *Choose a subset V' of V.*
  Note:
  - *V' is our key.*
  - *Choose the cardinality of V' to the number of partitions of the message chosen or any random value between 1 to r.*
- *Encrypt each partition of the message with some sequence of elements of V' and send the encrypted message.*
- *At the receiver end identify the elements of V' and apply inverses of the corresponding functions.*

❖ Retrieving a permutation of {1,2,…,n} given its index for any n.
Note: index may be any number between 1 and n! (n factorial), (including both 1, n!).

❖ Generating subset V'.

VI. PROBLEM FORMULATION AND REPRESENTATION

❖ **Let:**

- **P(r)** be the set of all permutations of the elements **{1, 2, 3, … , r}.**
- $\mathbf{S_{(P(r))}}$ be a specific permutation of all the elements of **P(r).**
- $\mathbf{P_r(k)}$ be the **kth** element in $\mathbf{S_{(P(r))}}$**.**
- $\mathbf{P_r(k)||r}$ **be** $\mathbf{P_{(r-1)}(k')}$
- $\mathbf{X((P_r(k)),j)}$ be the position of **j** in $\mathbf{P_r(k)}$**.**
- $\mathbf{Y(P_r(k)),j)}$ be the element at position **j** in $\mathbf{P_r(k)}$**.**
- **0** be the index of position **j** in $\mathbf{P_r(k)}$ if $\mathbf{X((P_r(k)),j)}$ is not known so far.
  Note : initially all positions are indexed **0 (since we don't know the permutation)**.
- **1** be the index of position **j** in $\mathbf{P_r(k)}$ if $\mathbf{X(P_r(k)),j)}$ is known.
- **F(k,r)** be the position of index **0** at which **r** occurs in $\mathbf{P_r(k)}$**.**
- **G(k,r)** be **k'** in $\mathbf{P_{(r-1)}(k')}$.

❖ **Formulae:**

1) **F(k,q) = k (modulo q).**

2) **2.1)** **G(k,q) = [k/q]+1 … F(k,q) is nonzero.**
Note : [ ] used above is the greatest integer function.

2.2) **G(k,q) = (k/q) … F(k,q) is zero.**

❖ **Generating keys:**

1) Choose any two different numbers **a, b** such that **1 <= a, b <= n**.
2) $\mathbf{key_{(i)} = Y((P_n(key_{(i-1)})),key_{(i-2)})}$**.**
   … recursive formula and Y(… , …) as defined above.
3) $\mathbf{key_{(0)} = Y((P_n(a)),b)}$**.**

4) $key_{(1)} = Y((P_n(b)),a)$.

Such recursively computed keys makes them dependent on the previous ones.

VII. Solution Methodologies And Problem Solving

❖ **Pseudocode for retrieving permutation of {1,2,…,n} with index k:**

1) Take 2 values :

- n : the cardinality of the set {1,2,…,n}.
- k : the index value of the permutation such that 1 <= k <= n! .

2) Declare two arrays of length n ;
   - Array 'ar' for storing the permutation with index k.
   - Array 'car' for storing the index of positions of positions (either 0 or 1).

3) Consider integers count=0, x and y.

4) Loop to get permutation with index k:

```
for(int i=0 ; i<n ; i++)
        {
                if( k % (n-i) == 0)          //% is modulus operation

                {
                        x=(n-i);
                }

                else

                {
                        x=( k % (n-i) );
                }

                if( k % (n-i) == 0 )

                {
                        y =  ( k / (n-i) ) ;
                }

                else

                {
                        y = ( ( k / (n-i) ) + 1 );
//assume ( k/(n-i) ) returns an integer value equivalent to greatest integer function
                }

                for(int l=0; l<(n); l++)

                {

                        if(car[l]==0)

                        {
                                count++;

                                if(count==x)

                                {
                                        ar[l]=(n-i);
```

```
                        car[l]=1;
                    }

                    if(count>x)
                    break;
                }

            }
            k=y;

            count=(0);
        }
```

5) Print array ar.

- **Formal proof for bijection between:**
  - **bijective functions from {1,2,..,n} to {1,2,..,n}**
  - **permutations of the set {1,2,…,n}**

    - let (t1,t2,…,tn) be some permutation of {1,2,…,n}.

      Consider the mapping

      i -> ti … for all i running from 1 to n; AND ;for different i and j values ti and tj are different.

      The above mapping is one-one and onto.

      Thus for every permutation (t1,t2,…,tn) of {1,2,..,n} , there exists unique bijective mapping from {1,2,..,n} to {1,2,..,n}.

      …#1

    - Consider the bijective mapping from {1,2,..,n} to {1,2,..,n}:

      i -> ti … for all i running from 1 to n.

      So for different i and j values ti and tj are different and sets {t1,t2,…,tn} and {1,2,…,n} are equal.

      Thus for every bijective mapping from {1,2,..,n} to {1,2,..,n} there exists a unique permutation (t1,t2,…,tn) of {1,2,..,n}.

      …#2

    - #1 and #2 implies the bijection between :

      - bijective functions from {1,2,..,n} to {1,2,..,n}
      - permutations of the set {1,2,…,n}.

- **Method to encrypt data :**

  Let the message be partitioned into p partitions and **let us choose the cardinality of V' to be p**.
  Note: if cardinality is chosen as some other number **k** then:
  - **if k > p : Generate keys only till key_p and encrypt.**
  - **if k < p : Generate all keys and repeat the sequence of keys for encryption.**

  Let $message_{pi}$ be the message in the i^{th} partition.
  Encode $message_{pi}$ using ASCII values of characters in some radix.

Let int_{pi} be the encoded value of $message_{pi}$.

Encryption of int_{pi} is $Y((P_n(key_i)),int_{pi})$

- **Method to decrypt data :**

At the receiver end retrieve **$Y((P_n(key_i)),int_{pi})$** get back int_{pi} as follows:-

$int_{pi} = X((P_n(key_i)),(Y((P_n(key_i)),int_{pi})))$

VIII. Results Analysis, Comparison and Justification

Time to compute subset V' while encryption

- Time required to break the cipher text at 33.86 PFLOPS, the speed of world's fastest supercomputer till date, TIANHE-2.

TABLE I. TIME AND SPEED ANALYSIS

VALUE OF n	SPEED (in GFLOPS)	TIME TO RETRIEVE PERMUTATION WITH INDEX k (in seconds)	CARDINALITY OF SET V'	TIME TO COMPUTE ELEMENTS OF V' (in seconds)
100	10	0.000001	10	0.00001
1000	10	0.0001	10	0.001
10000	10	0.01	10	0.1
100000	10	1	10	10
1000000	10	100	10	1000
100	100	0.0000001	25	0.0000025
1000	100	0.00001	25	0.00025
10000	100	0.001	25	0.025
100000	100	0.1	25	2.5
1000000	100	10	25	250
100	1000	0.00000001	50	0.0000005
1000	1000	0.000001	50	0.00005
10000	1000	0.0001	50	0.005
100000	1000	0.01	50	0.5
1000000	1000	1	50	50

(Source from which the speed information about TIANHE-2 is taken: **http://en.wikipedia.org/wiki/Tianhe-2**)

TABLE II. DECRYPTION WITH TIANHE-2

VALUE OF n	NUMBER OF PARTITIONS p OF THE MESSAGE	WORST-CASE TIME REQUIRED TO BREAK THE CIPHER-TEXT (in years)
100	**15**	**0.936 million**
100	**16**	**93.6 millions**
100	**17**	**9.36 billions**
100	**18**	**936 billions**
100	**19**	**93600 billions**
100	**20**	**9.36 quadrillion**
1000	**15**	**0.936 million quadrillion**
1000	**16**	**936 million quadrillion**
1000	**17**	**936 billion quadrillion**

Column 3 proves the computational infeasibility to break the cipher text for small values of n (like n=100) even with the highest achievable speed so far that is 33.86 PFLOPS.

Time complexity

TABLE III. TIME COMPLEXITY

Method	Time complexity
THE PHNIKS ALGORITHM	**$O(n^2)$**
RETRIEVING PERMUTATION WITH INDEX k	**$O(n^2)$**
GENERATING SUBSET V'	**$O(n^2)$**
ENCRYPTION	**$O(n^2)$**
DECRYPTION	**$O(n^2)$**

❖ Brute-force attack:

It is evident that every $message_{pi}$ is mapped bijectively to some number from 1 to n (including both), so there are n attempts to decipher $message_{pi}$.

So there are (n^p) attempts (worst case) to be made to decipher the whole message, where p is the number of partitions of the message.

IX. CONCLUSION

The strength of the 'PHNIKS' algorithm is demonstrated by the tabular data, computation time complexity and justification of the computational infeasibility (brute-force attack) to break the cipher-text with the requirement of (n^p) computations.

FUTURE WORK

Finding clever attacks for the 'PHNIKS' algorithm.

REFERENCES

[1] Bartle, r.g., sherbert, d.r., "introduction to real analysis", third edition, john wiley and sons, 2007.
[2] Tucker, alan, "applied combinatorics", fourth edition, john wiley and sons, 2004.
[3] Tanenbaum, a. S., wetherall, d. J., "computer networks", fifth edition, pearson, 2012.
[4] Reference to Tianhe-2 Supercomputer, http://en.wikipedia.org/wiki/Tianhe-2 .
[5] Consolidated Content review comments Paper should strictly formatted according to the single column standard format;Include more relevant references to substantiate the related works