Swarm Intelligence based Faster Public-Key Cryptography in Wireless Communication (SIFPKC)

Arindam Sarkar
Department of Computer Science and Engineering
University of Kalyani, Kalyani, Nadia
Pin- 741235, West Bengal, India
arindam.vb@gmail.com

J. K. Mandal
Department of Computer Science and Engineering
University of Kalyani, Kalyani, Nadia
Pin- 741235, West Bengal, India
jkm.cse@gmail.com

Abstract— **In this paper a Particle Swarm Organization guided faster version of public-key cryptography in wireless communication has been proposed by minimizing total number of modular multiplication, so that faster exponentiation can efficiently implemented which in turn progress the time requirements of the encryption / decryption of large number of plain or cipher text. Minimization of number of modular multiplication itself a NP-hard problem that means there is no polynomial time deterministic algorithm for this purpose. So, in this paper improved version of optimization algorithm using soft computing tools has been discussed to minimize the modular multiplication. This proposed SIFPKC technique begins with an initial population comprises of set of valid and complete set of particles. Then some operators like particles local best and global best positions along with velocity updating rules are used to generate feasible valid particle from the existing one. Among several exponents the best solution reached by SIFPKC is compared with some of the existing techniques. Extensive simulation shows competitive results for the proposed algorithm.**

Keywords- Swarm Intelligence based Faster Public-Key Cryptography (SIFPKC), Particle Swarm Optimization (PSO), cryptography, exponent, public-key, addition sequence, modular multiplication.

I. INTRODUCTION

Various public-key cryptosystem (RSA, Diffie-Hellman Key Exchange, and DSA) [1, 2] make use of modular exponentiation for cryptographic reason. Any public-key cryptosystem comprises of a public key, private key and modulus, as given in equation (1).

$$c = p^{E} \bmod N \tag{1}$$

Where plain text (p) is a positive integer in the range of [0, 1, 2, 3, N-1] in RSA technique. N is a result of multiplication of two very large prime number and E is a randomly chosen positive number which satisfies this equation. In this exponentiation large numbers of multiplications are requisite. In contemporary cryptographic technique E value could be greater than 128 bits [6]. For this explanation in asymmetric key cryptography encryption as well as decryption is very computationally costly.

This work introduces different swarm and evolutionary approaches to reduce number of multiplication needed to compute c. This above discussed exponent difficulty directly can be mapped directly in to an addition chain for computing exponent. As an alternative of multiplication, for a particular exponent value addition chain of sequence of integer can be generated using following properties:

- Value of the first element of the sequence always 1.
- Each consequence elements is generated by addition of two earlier elements.
- Last element value is identical as the exponent E.

For example if we require to discover p^{97} then one possible incompetent method to calculate (px , … ,xp) 97 times. Following competent method for exponent calculation is through addition series 1-2-3-5-10-20-40-50-90-95-97 that leads to the following scheme:
$a^1 = a \rightarrow a^2 = a^1 a^1 \rightarrow a^3 = a^1 a^2 \rightarrow a^5 = a^2 a^3 \rightarrow a^{10} = a^5 a^5 \rightarrow a^{20} = a^{10} a^{10} \rightarrow a^{40} = a^{20} a^{20} \rightarrow a^{50} = a^{10} a^{40} \rightarrow a^{90} = a^{40} a^{50} \rightarrow a^{95} = a^5 a^{90} \rightarrow a^{97} = a^2 a^{95}$

Where sequence length is 11. So, only 11 multiplications are needed to calculate $p^{97.}$ But for the similar example one more addition sequence can be generated with length 10 i.e. 1-2-4-6-10-20-24-48-96-97 which leads to following sequence of computation:

.

$a^1 = a \rightarrow a^2 = a^1 a^1 \rightarrow a^4 = a^2 a^2 \rightarrow a^6 = a^4 a^2 \rightarrow a^{10} = a^6 a^4 \rightarrow a^{20} = a^{10} a^{10} \rightarrow a^{24} = a^{20} a^4 \rightarrow a^{48} = a^{24} a^{24} \rightarrow a^{96} = a^{48} a^{48} \rightarrow a^{97} = a^{96} a^1$.

So, from the example it is seen that for a particular exponent value there may be several addition sequence of different length.

For sinking number of multiplication sequence having tiny length for a particular exponent always be selected which is an optimization problem. There are more than a few deterministic [8] and stochastic [5] and heuristics based techniques proposed for solving this optimization problem.

This paper explored soft computing based stochastic method to improve its capabilities in this search space. Improved and efficient versions of stochastic method with some novel variation are proposed.

The organization of this paper is as follows. Section II of the paper deals with problem statement. Some of the existing techniques to handle this problem have been discussed in section III. Proposed SIFPKC based strategy discussed in section IV. Experimental results are given in section V. Analysis regarding various aspects of the technique and results have been presented in section VI. Conclusions with future scopes are drawn in section VII and that of references at end.

II. PROBLEM STATEMENT

The problem tackled in this paper is the optimization of addition sequence length X for a given exponent E. So, aim is to find out addition sequence with minimum length (l). An addition sequence X with length l is defined as a sequence of positive integers $X= x_1, x_2, x_1,\ldots x_i,\ldots, x_l,$ with $x_1=1$, $x_2 = 2$ and $x_l = E$, and $x_{i-1} < x_i < x_{i+1}$, where each x_i is obtained by adding two previous elements $x_i = x_j + x_k$ with j, k< i for i>2. Notice that j and k are not necessarily dissimilar.

III. RELATED WORK

A number of deterministic and probabilistic algorithms were proposed to find minimum addition chain. Some of them are Binary [3], m-ary [4], adaptive m-ary, Power tree, The Factor Method, Window, Adaptive Window, Artificial Immune system , Genetic Algorithm [3, 4, 5, 9, 11, 12, 13]. Binary Method expands exponent to its binary version with length m. Then implementing a prearranged algorithm it is scanned from left to right or right to left and depending on the binary value of the scanned bit this algorithm computes fields squaring and multiplications function.

This technique is enhanced by scanning more than one bit at a time. This new strategy is known as Window method [4] where k bits are scanned at a time. It is based on k-ary expansion of the exponent, where the bit of exponent are divided into k- bit words.

The Adaptive Window strategy (different versions are [3, 4, 5, 8, 9, 10]) can fine-tune its technique according to the definite form of the given exponent. Here binary inputs exponent get divided into a series of variable having length zero and non zero digits, called window, which are get processed. This algorithm is valuable for exponents with bit length more than 128 bits.

Despite the fact that most of the methods ate deterministic but a few probabilistic heuristics approaches were also proposed [11, 12, 13]. A simple GA based approach [11] with encoded addition chain as a chromosome was presented. This GA based method uses simple selection strategy and one point crossover technique and applied on a small set of exponents and obtained a competitive result oppose to earlier discussed proposed deterministic approach. One more algorithms i.e. Artificial Immune System (AIS) [13] where only feasible

addition chains are considered has also been proposed which works by emulating the Colonial Selection Principle where the unsurpassed individuals are cloned and these clones are also mutated.

IV. PROPOSED SIFPKC STRATEGY

PSO is a swarm intelligence based optimizaion technique. In PSO potential solutions to the problem are called paricles. PSO based addition sequence optimization has alrady been proposed in [14] has some shortcomings like if (2 * $particle_{i-1}$ < exponent) and either (pBest's length>i) or (gBest's length>i) then using random (0,i-1) one random number. is generated which becomes the velocity value and this will be added to $particle_{i-1}$ to obtain $particle_i$. But this strategy is not useful to generate particle with better fitness. Because random (0,i-1) will always not generate maximum possible value to minimize the particle length. As for example if exponent = 81, using the existing PSO algorithm random (0,i-1) can generate 5, then $particle_i$ will be (25+5=30). Where particle length is 12 and fitness is 11 (length-1) (as illustrated in fig. 1).

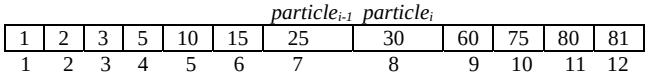

Figure 1. Particle construction using random value with fitness=11

But our proposed SIFPKC version tackles this problem using the operations.

$particle_{i=}\ particle_{i-1} + particle_{i-2}$
$velocity_i = i-2$

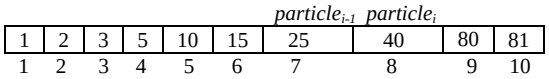

Figure 2. Particle construction using new rule with fitness=9

So, using SIFPKC algorithm value of $particle_i$ will be 40. Where particle length is 10 and fitness is 9 (as illustrated in fig. 2) which is better than former version [14]. This improved PSO also prevent the use of random (0,i-1) even if (2$particle_{i-1}$ >exponent) and ($particle_{i-1}$≠exponent) by using binary search mechanism for finding out most fittest element to add with $particle_{i-1}$ with the aim of produce particle with better fitness.

A. Encoding of paricle's with Fitness Calculation

Only valid feasible addition sequence gets encoded. Single dimension array of integer numbers represents an addition chain which is consider as a particle of PSO. For example if exponent =33 then particle can be represented as (1, 2, 3, 4, 8, 16, 32, 33) with fitness 7 (i.e. one less than particle's length). If *(fitness (paricle) < pBest) then pBest= fitness (paricle).*Also find best fitness of neighbor and assign it to gBest.

B. Initial population

Initial population consists of collection of valid addition sequence. Where 1st and 2nd element of each particle will be 1 and 2 respectively.

Initial_Population ( )

Input: *Exponent*
Output: *A valid set of particles*
Method:
for j=1 to POPULATION-SIZE do
Set $x^j_0 = 1$ and $x^j_1 = 2$, Initialization of 0^{th} & 1^{st} position of j^{th} particle.
Allocate $x^j_2 = x^j_1 + rnd(x^j_0, x^j_1)$ where rnd() returns a random integer in the interval.
Generate a complete $particle^j$ using,
($Particle^j$, l) = ***Paricle_Construction (****$Particle^j$, exponent)*
end for

C. Particle's Velocity

A velocity value is an array of same length of the particle. It shows which elements were added to obtain the current element in the particle. Let $particle_i$ and $velocity(particle_i)$ are the i^{th} element and its velocity respectively.

$particle_i = particle_{i-1} + velocity(particle_i)$. Consider the following fig. 3.

Particle										
1	2	3	5	10	15	25	50	75	80	81
0	1	2	3	4	5	6	7	8	9	10
Velocity										
0	0	0	1	3	3	4	6	6	3	0

Figure 3. Particle with velocity

If i=4 then $particle_{4=}$ $particle_3 + velocity(particle_4) = 5+5=10$.

D. Updation of Velocity

The particle position is affected by the best position of the particle i.e. pBest and its neighborhood's best experience i.e. gBest. After scanning value of each particle. Then at each step check the particle is affected by the rule in the velocity chain associated with (1) pBest or (2) gBest or (3) its own velocity.

Algorithm of each particle construction is given below.

Paricle_Construction ( )

```
for i=2 to length do
  if (2particle_{i-1} <exponent)
     if (prob(pBest)) then  /* Apply pBest velocity rule*/
          if (pBest's length<i) then
             particle_i= particle_{i-1} + pBest (Velocity (particle_i))
             velocity_i = pBest(velocity_i)
           else
             particle_i= particle_{i-1} + particle_{i-2}
             velocity_i=i-2
     if (prob(gBest)) then  /* Apply gBest velocity rule*/
            if (gBest's length<i) then
              particle_i= particle_{i-1} + gBest (Velocity(particle_i))
              velocity_i = gBest(velocity_i)
            else
              particle_i= particle_{i-1} + particle_{i-2}
              velocity_i=i-2
        else if (velocity_i<i) then/* Apply its own velocity rule*/
              particle_i= particle_i + velocity(particle_i)
             else
              particle_i= particle_{i-1} + particle_{i-2}
              velocity_i=i-2
   else
       if (particle_{i-1}≠exponent)
          diff= exponent- particle_{i-1}
    /*Perform binary search in the range of (q=i-2 to 0) to get particle having value==diff */
             if (particle_q==diff) then
                particle_i= particle_{i-1} + particle_q
                velocity_i=q
         else if (particle_q>diff) then
                while (particle_q>diff) do
                       q=q-1;
    particle_i= particle_{i-1} + particle_q
            velocity_i=q
    else if (particle_q<diff) then
                 while (particle_q<diff) do
    q=q+1;
```

$particle_i = particle_{i-1} + particle_q$
$velocity_i = q$
end for

Complete algorithm of the proposed SIFPKC is given below.

SIFPKC (P, i, exponent)
Input: *A partial particle = $p_1, p_2, ..., p_l = e$,*
where i represents the next position to be filled.
Output: *A feasible particle for a given exponent, with length l*
Method:

Construct particles initial population of size N.
repeat
for each $particle_i$ do
Calculate fitness ($particle_i$)
If (fitness ($particle_i$) < pBest) then
pBest= fitness ($particle_i$)
Assign the best neighbor's position to gBest.
Using velocity updation rules update the velocity and position of $particle_i$
end for
Repeat until termination criteria are not satisfied.

V. EXPERIMENTTAL RESULTS

Performance of proposed SIFPKC approach is compared with its previous version [11, 12, 14] and other deterministic and heuristic based [3, 4, 5, 9, 13] approaches with the aspire to (1) show that performance is competitive (or even improved) with those provided by other heuristic-based or deterministic approaches and (2) to solve even more complex instances of the problem. Complexity of the problem depends on the value of exponent. This section highlighted on the best solution reached so far i.e. quality and statistical analysis for reviewing consistency. Different parameters value that are used in proposed SIFPKC approach are obtained by a trial-and error method by favoring the best overall performance.

A. Parameters used in proposed SIFPKC

Number of particles=30
Number of iterations=10000
Neighborhood configuration=global
Probability to apply the pBest's velocities rules=0.5
Probability to apply the pBest's velocities rules=0.25

The first set of experiments computes the total accumulated addition chains for a fixed set of exponents. An accumulated addition chain for a maximum value *P*, represents the sum of all addition chains obtained for all the exponents 1, 2, . . .,*P* as stated in following equation. An accumulated addition chain for a given exponent, represents the sum of all addition chains obtained for each number in the sequence defined by exponent stated in equation (2).

$$Accumulated\_Add\_Chain = \sum_{i=1}^{P} Optimal\_Add\_Chain \quad (2)$$

A smaller value of Accumulated_Add_Chain represents better performance by the algorithm.

In table I set of accumulated addition chains for all exponents less than: 512 ($e \in [1, 512]$), 1000 ($e \in [1, 1000]$), 2000 ($e \in [1, 2000]$), 2048 ($e \in [1, 2048]$) and 4096 ($e \in [1, 4096]$), is presented in order to compare the results with respect to earlier [14, 15] and proposed modified version of algorithm and some other heuristic [13] and deterministic [3, 4, 5, 9] algorithms that has been proposed so far.

Table I represents the optimal value and the best results reached by proposed SIFPKC and existing heuristics and deterministic approaches. In table II statistical results obtained in 40 independent runs for all exponent set consider so far by the proposed SIFPKC are listed. Binary method, adaptive window method or other non-deterministic heuristics methods [3, 4, 5, 9, 11, 12, 13] are not be able to obtained shortest addition

chain for optimizing a given exponent. So a given exponent is hard to optimize using these strategies. Whereas proposed SIFPKC have the capabilities to address and solve this problem.

TABLE I. COMPARISON OF BEST RESULTS OBTAINED BY THE FORMER VERSION OF PSO [14], AIS [13] , QUATERNARY, BINARY AND PROPOSED SIFPKC

Exponent	Optimal	Proposed SIFPKC	PSO [14]	AIS [13]	Quaternary	Binary
[1,512]	4924	4924	4924	4924	5226	5388
[1,1000]	10808	10809	10813	10813	5603	5812
[1,1024]	11115	11118	11120	11120	11862	12301
[1,2000]	24063	24086	24095	24108	25923	26834
[1,2048]	24731	24752	24765	24778	26664	27662
[1,4096]	54425	55578	55609	56617	58678	61455

In table II statistical results obtained in 40 independent runs for all exponent set consider so far by the proposed SIFPKC are listed. Binary method, adaptive window method or other non-deterministic heuristics methods are not be able to obtained shortest addition chain for optimizing a given exponent. So a given exponent is hard to optimize using these strategies. Whereas proposed SIFPKC have the capabilities to address and solve this problem.

TABLE II. STATISTICAL RESULTS OF PROPOSED SIFPKC

Exponent	Best	Average	Median	Worst
[1,512]	4924	4924	4924	4924
[1,1000]	10809	10811.74	10812	10816
[1,1024]	11118	11119.59	11120	11123
[1,2000]	24086	24091.72	24092	24108
[1,2048]	24752	24758.27	24758	24779
[1,4096]	55578	55597.61	55598	55613

Table III shows average length of addition chain for proposed SIFPKC vs. Binary, Quaternary technique where SIFPKC obtained shortest addition chain in exponent size compare to the Binary and Quaternary techniques.

TABLE III. AVERAGE LENGTH OF ADDITION CHAIN FOR PROPOSED SIFPKC VS. BINARY, QUATERNARY METHOD

Exponent size	Proposed SIFPKC	Binary	Quaternary
32	43	47	43
64	81	95	87
128	162	191	175
256	326	383	351
512	648	767	703
1024	1291	1535	1407

VI. ANALYSIS OF RESULTS

From table I it is clearly observed that proposed modified PSO performs well in all the cases. In larger range of exponent value i.e. [1-1000], [1-2000], [1-2048], [1-4096] proposed SIFPKC outperforms than existing PSO [14]. This proposed SIFPKC performs better than AIS [13], Quaternary and Binary [3, 4, 5] techniques in the entire considered exponent range. Overall findings suggest that proposed SIFPKC performs outstandingly well among all the techniques and its results close to optimal values in most cases.

VII. CONCLUSIONS AND FUTURE SCOPE

In this paper several modifications of existing PSO techniques has been done for finding the optimal addition sequence for a given exponent. Proposed SIFPKC revised the velocity updating and particle formation method of existing technique in [14]. This proposed SIFPKC method is also compared with the existing heuristics and deterministic approaches [3, 4, 5, 11, 12, 13, 14]. SIFPKC outperformed all techniques mostly in larger exponent i.e. [1-1000], [1-2000], [1-2048], [1-4096]. Furthermore, proposed SIFPKC approach able to generate optimal addition sequence on exponent where no other heuristic based approach has reported results in the specialized literature.

Future scope of the proposed SIFPKC is to apply other soft computing tools in this problem domain and to design better velocity updating rules for proposed SIFPKC approach. Another future scope is to analyze the parameters required by these algorithms to reduce the number of evolutions without affecting the efficiency and test the algorithms with larger exponent (more than 160 bits).

ACKNOWLEDGMENT

The author expresses deep sense of gratitude to the DST, Govt. of India, for financial assistance through INSPIRE Fellowship leading for a PhD work under which this work has been carried out.

REFERENCES

[1] Atul Kahate, Cryptography and Network Security, 2003, Tata McGraw-Hill publishing Company Limited, Eighth reprint 2006.
[2] ANSI X9.17 (Revised), National Standards for financial institution key management (wholesale), American Bankers Association, 1986.
[3] D. M. Gordon, “A survey of fast exponentiation methods,” Journal of Algorithms, vol. 27, no. 1, pp. 129–146, April 1998.
[4] D. E. Knuth, Art of Computer Programming, Seminumerical Algorithms. Addison-Wesley Professional, November 1997, vol. 2.
[5] J. Bos and M. Coster. “Addition chains heuristics”. Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdan, The Netherlands., Springer-Verlag:1–5, 1998.
[6] C. Kaya-Koc. “High-speed RSA implementation”. Technical report, RSA Laboratories, Redwood City, CA, 1994.
[7] S. V.-D. Kruijssen. “Addition chains, efficient computing of powers. Bachelor Proyect, Amsterdam, 1:13–50, 2007.
[8] N. Kunihiro and H. Yamamoto, “Window and extended window methods for addition chain and addition-subtraction chain,” IEICE Trans. Fundamentals, vol. E81-A(1), pp. 72–81, January 1998.
[9] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson, “Fast exponentiation with precomputation,” In R. A. Rueppel, (editor) Advances in Cryptology —EUROCRYPT 92 Lecture Notes in Computer Science, vol. 658, pp. 200–207, 1992.
[10] C. K. Koc, “Analysis of sliding window techniques for exponentiation,” Computer and Mathematics with Applications, vol. 30, no. 10, pp. 17–24, October 1995.
[11] N. Cruz-Cortés, F. Rodríguez-Henríquez, R. Juárez-Morales, and C. A. Coello-Coello. “Finding optimal addition chains using a genetic algorithm approach”. Lecture Notes in Computer Science, Computer Science Section, Electrical Engineering Department, CINVESTAV IPN, 2:1–8, 2005.
[12] L. G. Osorio-Hernández, E. Mezura-Montes, N. Cruz-Cortés, and F. Rodríguez-Henríquez. “An improved genetic algorithm able to find minimal length addition chains for small exponents”. In Proceedings of the IEEE Congress on Evolutionary Computation, pages 1–6. IEEE Press, 2009.
[13] N. Cruz-Cort´es, F. Rodr´ıguez-Henr´ıquez, and C. A. C. Coello, “An Artificial Immune System Heuristic for Generating Short Addition Chains,” IEEE Transactions on Evolutionary Computation, vol. 12, no. 1, pp. 1–24, February 2008.
[14] Alejandro León-Javier, Nareli Cruz-Cortés, Marco A. Moreno-Armendáriz and Sandra Orantes-Jiménez ,“Finding Minimal Addition Chains with a Particle Swarm Optimization Algorithm” MICAI 2009: Advances in Artificial Intelligence Lecture Notes in Computer Science, Volume 5845/2009, 680-691, DOI: 10.1007/978-3-642-05258-3_6, 2009..