2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

A Verifiable PSO Algorithm in Cloud Computing

Tao Xiang, Weimin Zhang and Fei Chen

***Abstract*—In this paper, we study the verification problem of particle swarm optimization (PSO) when it is outsourced to the cloud, i.e. making sure that the cloud executes PSO algorithm as requested. A verifiable PSO algorithm and its verification algorithm are proposed. The proposed scheme does not involve expensive cryptography, and it is efficient and effective to verify the honesty of the cloud.**

I. Introduction

PARTICLE swarm optimization (PSO) is a population-based heuristic optimization algorithm introduced by Kennedy and Eberhart [1], [2]. It is a metaphor of the social behavior of animals such as bird flocking and fish schooling. PSO optimizes a problem by iteratively trying to refine candidate solution with regard to a given measure of quality. It makes few or no assumptions about the problem being optimized. PSO outperforms other population-based optimization algorithms, such as genetic algorithm (GA) and other evolutionary algorithms, mainly in its simplicity [3], [4], [5]. It has proved its capability and efficiency of solving various optimization problems [6], [7], [8].

PSO consists of particles and each particle has its position and velocity. The position represents a potential solution; the particle flies through search space at its velocity and dynamically changes the velocity according to its own flying experience and the flying experiences of other particles. The velocity and position updating rules can be formulated as follows.

$$V_i(t+1) = \omega V_i(t) + c_1 r_1 (P_i - X_i(t)) + c_2 r_2 (G - X_i(t)) \quad (1)$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \quad (2)$$

where $V_i(t)$ and $X_i(t)$ denote the velocity and position of the i-th particle at the t-th iteration, respectively. P_i is the best position found so far by the i-th particle. G the global best position found so far by all particles. ω is inertia weight [9]. r_1 and r_2 are uniformly distributed random numbers between 0 and 1. c_1 and c_2 are positive constants referred as acceleration constants.

The principle of PSO and its implementation are simple; notwithstanding, the running of PSO algorithm in practice is often extremely computationally intensive, as practical problems are usually extremely complicated (e.g. multimodal and high-dimensional). For this reason, when someone with limited resources has a complicated problem to be optimized by PSO algorithm, he can offload the computing to a third party with much more powerful computational resources. Especially in the coming age of cloud computing [10], a thin client such as smartphone can outsource this computing task to the cloud. There are great benefits of being doing so: the client can save the money on expensive hardware for running the algorithm; and he can get the desirable solution from the cloud with the help of outsourcing.

However, there are security and privacy challenges in cloud computing environments [11], [12]. Although outsourcing the running of PSO algorithm has attractive superiority, it induces a serious problem: how could the result returned by the cloud be verified? In other words, to reduce overhead cost, the cloud could possibly cheat the client by not faithfully running the PSO algorithm, or even by returning a random guess without executing the algorithm at all. There are some general solutions in principle for this problem in the field of complexity theory and cryptography [13]; however, most of them are infeasible in practice because they are based on heavy cryptographic operations and are extortionately expensive to be implemented [14].

In this paper, to the best of our knowledge, we consider the outsourcing of PSO algorithm and its verification for the first time. Unlike existing general solutions for verifiable computing relying on expensive cryptography, we propose a simple and efficient scheme. Some reasonable assumptions are made to simplify the problem, and the client can verify the honesty of the cloud.

The rest of this paper is organized as follows: Section II defines the problem and its security model. Section III proposes the solution in detail. Section IV gives the analysis of the proposed scheme. Experimental results are provided in Section V. Section VI concludes the paper.

II. The Problem Formulation

A. The Problem Definition

We consider a client has a complicated problem F to be optimized by PSO, then he outsources the execution of PSO algorithm with some parameters to the cloud who has much more powerful computational resources. After the cloud solves the problem by running PSO algorithm, it returns the result to the client. Upon receiving the result from the cloud, the client wants to verify whether the result is correct. In other words, whether the cloud has invoked the PSO algorithm faithfully.

To simplify the problem, we make some hypotheses on PSO algorithm. First, without loss of generality, we assume F is a minimum optimization problem. Second, as we know, PSO algorithm has some parameters that should be determined before running, such as population size and velocity

The corresponding author is Tao Xiang (txiang@cqu.edu.cn).

Tao Xiang and Weimin Zhang are with the College of Computer Science, Chongqing University, Chongqing 400044, China (email: txiang@cqu.edu.cn; qwe4037@qq.com).

Fei Chen is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong (email: fchen@cse.cuhk.edu.hk).

This work was supported by the Fundamental Research Funds for the Central Universities (No. CDJZR13185501), the Natural Science Foundation Project of CQ CSTC (No. cstc2013jcyjA40001), and the Program for New Century Excellent Talents in University (No. NCET-12-0589).

limitation; to simplify the problem, we only consider that the client can config the stop criterion as other parameters are not strictly problem specific and usually have recommended configuration. Last, there are usually two stop criteria for PSO algorithm: reaching a predefined error measure or a maximum iteration number t; we follow the latter because it is widely accepted.

Based on these hypotheses, we can formally model the PSO outsourcing problem as

$$r = PSO\_outsouce(t) \tag{3}$$

where $PSO\_outsouce(*)$ is the PSO algorithm to be outsourced to the cloud for the optimizing problem F; t is a parameter whose value is provided by the client; r is the result returned by the cloud, and it is to be verified by the client.

B. Security Model

We assume that the cloud is semi-malicious, i.e. the cloud may deviate the protocol but will not tamper with the inside of PSO algorithm. This assumption is reasonable because the algorithm is usually implemented and provided by the client in real-world scenarios, and it can be sent to the cloud in the form of library or other executable format which is only readable. Upon receiving the outsourcing request from the client, the cloud may manipulate the parameters and then runs PSO algorithm with the manipulated parameters; after the running of PSO algorithm, the cloud may also change the return value and sends it back to the client; or the cloud may not even invoke PSO algorithm at all.

We also assume that the PSO algorithm is efficient for solving the problem F, and the value of t is not too small, otherwise there is no need for the client to outsource the computing. In this circumstance, the cloud is supposed to return a suboptimal solution which will not deviate much from the global optimum.

Finally, we follow the common assumption in cryptography that all the protocol and algorithms involved are public knowledge, except for the secret keys. The capability of any party, including the adversary, is bounded by its polynomial computational power and storage space.

III. The Proposed Scheme

A. General Framework

Generally speaking, the core of our proposed scheme is a verifiable PSO algorithm. In order to solve the formulated problem, there are three phases to fulfill the outsourcing of the proposed algorithm and its verification.

- **Request**: The client sends a outsourcing request to the cloud. The request contains the verifiable PSO algorithm and its arguments including additional information for subsequent verification.
- **Outsourcing Computing**: Upon receiving the request from the client, the cloud uses the arguments to run the verifiable PSO algorithm, and returns the output of it back to the client.
- **Verification**: When the client gets response from the cloud, he runs the verification algorithm to check whether the cloud has faithfully done his job.

B. Verifiable PSO Algorithm

We now describe the verifiable PSO algorithm in detail. The algorithm and its arguments are provided by the client and it is run by the cloud.

The input arguments include the maximum iteration number t, the signature $t\_sig = E_{sk}(t)$ of t by the user's private key sk, and the public key pk corresponding to sk. The output of the algorithm consists of a status value indicating whether the arguments from the client are correctly passed in, the optimized solution, and the public key pk. To further ensure the indistinguishability of status value for the cloud and the privacy for the client, all these values are encrypted by pk before being output.

In the algorithm, we first need to verify the arguments to check whether the value of t is really what the client requested. This can be done by verifying the signature of t, i.e. by decrypting t_sig with pk ($D_{pk}(t\_sig)$) and checking whether the decrypted value is identical with t. If the arguments verification succeeds, we then follow the procedures of traditional PSO algorithm to find the optimized solution G, and return the result $r = E_{pk}(true, G, pk)$.

The pseudocode of the verifiable PSO algorithm is given in Algorithm 1.

Algorithm 1 Verifiable PSO Algorithm

```
Input: $t$, $t\_sig$, $pk$
Output: $r$
 1: if $t$ != $D_{pk}(t\_sig)$ then
 2:     return $E_{pk}(false, 0, pk)$;
 3: end if
 4: Parameter initialization;
 5: for $i = 1$ to $t$ do
 6:     Calculate fitness function;
 7:     Update $P_i$, $G$;
 8:     Update velocities;
 9:     Update positions;
10: end for
11: return $E_{pk}(true, G, pk)$;
```

C. Verification Algorithm

When the client receives response from the cloud, he runs the verification algorithm to verify the result. As the result r is returned in encrypted format, the client needs to decrypt it using his private key sk. After that, he verifies the following points: 1) the returned public key is unchanged, 2) the arguments verification succeeds, and 3) the problem F is really optimized by PSO algorithm.

The first two points are quit easy to be accomplished, but the last one is a challenge because the solution found by PSO algorithm is suboptimal and it may be different from the real global minimum. For this reason, it is hard to find a deterministic method to verify the solution in general; however, under the assumptions we have made in this paper, we propose a simple and efficient way to verify the solution found by PSO algorithm. The basic idea is to disturb the received solution and check whether it is still optimal. Specifically, the client generates n disturbed values of G within the range of $[-\delta, \delta]$. If these n solutions are better than the cloud's response G by more than ϵ in a ratio greater than η, then the verification fails because the cloud may cheat in extremely high probability.

The pseudocode of the verification algorithm is described in Algorithm 2.

Algorithm 2 Verification Algorithm

```
Input: $r$, $n$, $\delta$, $\epsilon$, $\eta$
Output: verification result
 1: $(pass, G, pk') = D_{sk}(r)$;
 2: if ($pk'$ != $pk$) or ($pass$ != $true$) then
 3:     return $false$;
 4: end if
 5: $count = 0$;
 6: for $i = 1$ to $n$ do
 7:     Randomly generate a $X \in [-\delta, \delta]$;
 8:     if $(F(G) - F(G + X))/F(G)) > \epsilon$ then
 9:         $count = count + 1$;
10:     end if
11: end for
12: if $(count/n) > \eta$ then
13:     return $false$;
14: else
15:     return $true$;
16: end if
```

IV. Theoretic Analysis

A. Verifiability

Provided the cloud follows the algorithm specifications, he will be successfully verified by the client. First, if the cloud invokes the verifiable PSO algorithm with the arguments from the client, i.e. t, t_sig, and pk, the arguments verification will succeed since the decryption of signature t_sig with pk is t exactly; furthermore, the execution of the verifiable PSO algorithm will produce the desirable optimized solution as its output. After that, if the output is sent back to the client, it will obviously pass the checking of parameters in the verification algorithm; the optimized solution will also pass the verification in extremely high probability, as the solution is supposed to be very close to the global optimum under our assumptions and it is hard to find a better one in limited tries.

In the case that the cloud deviates from the algorithm specifications, it will definitely fail in the verification. Please find the detail in the subsequent part of statements.

B. Security

Under the assumptions in our security model, the cloud has the following two ways to cheat, and we will prove that both of them are infeasible.

One way for the cloud to cheat is that he returns a random guess r' of r without executing the verifiable PSO algorithm at all. Actually, the cloud only needs to randomly generate a solution G' and returns $r' = E_{pk}(true, G', pk)$ to the client. As G' is a random guess rather than the solution found by PSO algorithm, it is not optimal. The client is easy to find some better solutions compared with G' in the running of verification algorithm, so the verification will fail.

The other way for the cloud to cheat is executing the verifiable PSO algorithm with forged arguments. For saving computational overhead, the cloud may choose a t' ($t' << t$) instead of t to invoke the verifiable PSO algorithm. Because the cloud does not have the private key sk of the client, he needs to forge a pair of private and public keys sk' and pk'; then he signs t' with sk' and get $t'\_sig = E_{sk'}(t')$; finally he passes t', t'_sig, and pk' into the verifiable PSO algorithm. In this case, the forged arguments can pass the verification since $t' = D_{pk'}(t'\_sig)$ and a solution G' is obtained; however, the output of the verifiable PSO algorithm cannot pass the verification algorithm run by the client. Because if the cloud forwards the output $r' = E_{pk'}(true, G', pk')$ to the client, the verification will fail since the public key of client has been changed; if the cloud want to forge the output r' with his own choice, the situation is the same with returning a random guess as we have been discussed, and the verification will fail also.

C. Performance

As we can see from the description of our scheme, both the proposed verifiable PSO algorithm and its verification algorithm do not involve much heavy cryptographic computation. The computational efficiency of traditional PSO algorithm is almost inherited in the verifiable PSO algorithm, and the verification algorithm is also computationally efficient.

Besides, as the verifiable PSO algorithm is derived from traditional PSO algorithm and all the virtues of PSO are keep intact, the optimization efficiency of verifiable PSO algorithm is identical with traditional PSO algorithm. In other words, the quality of the solution found by them are the same given the same parameters.

Last but not the least, although public cryptography is employed here, we do not need public key infrastructure (PKI) to certificate the public key of the client. The client can generate any available key pairs in the scheme, and it will not degrade the verifiability or the security of the scheme.

V. Experimental Results

Six benchmarks being widely used in literature are adopted in our experiments, and their mathematical representations and related parameters configuration are listed in Table I. The population size is set to 30 throughout the experiments.

In the experiments, the configuration of parameters n, δ, ϵ, and η are of great importance for the verification. First, improper values will make the verification result false positive or false negative; Second, n should be as small as possible to save the computational overhead of the client in the verification. By extensive experiments we find that it is a great tradeoff between the verifiability and the performance of the verification algorithm by setting $n = 10$, $\delta = 0.1$, $\epsilon = 0.01$, and $\eta = 0.05$.

TABLE I. BENCHMARK FUNCTIONS

Function	Name	Mathematical representation	Dimension	Domain	Global Minimum
f_1	Sphere	$f_1(x) = \sum_{i=1}^{n} x_i^2$	30	$[-100, 100]$	0
f_2	Rosenbrock	$f_2(x) = \sum_{i=1}^{n-1} \Big(100 \;\; x_{i+1} - x_i^2\Big)^2 + (x_i - 1)^2\Big)$	30	$[-30, 30]$	0
f_3	Rastrigrin	$f_3(x) = \sum_{i=1}^{n} \;\; x_i^2 - 10\cos(2\pi x_i) + 10\Big)$	30	$[-5.12, 5.12]$	0
f_4	Griewank	$f_4(x) = (1/4000)\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos \;\; x_i/\sqrt{i}\Big) + 1$	30	$[-600, 600]$	0
f_5	Ackley	$f_5(x) = -20\text{exp} \;\; -0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\Big) - \text{exp} \;\; \frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\Big) + 20 + e$	30	$[-32, 32]$	0
f_6	Schaffer	$f_6(x) = 0.5 + \Bigg(\Bigg(\sin\sqrt{x_1^2 + x_2^2}\Bigg)^2 - 0.5\Bigg) / \;\; 1 + 0.001 \;\; x_1^2 + x_2^2\Big)\Big)^2$	2	$[-100, 100]$	0

As it is discussed in Section IV-B, if the cloud executes the verifiable PSO algorithm with forged arguments, he will undoubtedly fail in the verification. Therefore, we only concentrate on the situation when the cloud returns a random guess of optimal solution here. For each benchmark function, t is set to 5000; we flip a random coin b to simulate whether the cloud cheats; If $b = 0$ we run the verifiable PSO algorithm and get its output, otherwise we just generate a random guess as its output; the output is then used for verification. This process is repeated 5000 times for each benchmark functions.

The verification results are shown in Table II. It is found that for all benchmark functions, the correct verification ratios are greater than 99.9%, which demonstrates the feasibility of our scheme. In some cases, such as f_1, f_3, and f_4, the false positive ratios are greater than zero. The reason is that there are some chances for PSO algorithm to be trapped into local minima, and it cannot approximate to the global minima anymore. This scenario deviates from the assumption in this paper, and the verification algorithm has a higher possibility to estimate the response from honest cloud to be forged. The situation of non-zero false negative ratios are random, and it is bounded to be small theoretically.

TABLE II. EXPERIMENTAL RESULTS WHEN $t = 5000$

Function	Verification ratio		
	Correct	False positive	False negative
f_1	99.90%	0.01%	0.00%
f_2	99.98%	0.00%	0.02%
f_3	99.90%	0.02%	0.08%
f_4	99.91%	0.09%	0.00%
f_5	99.93%	0.00%	0.07%
f_6	99.94%	0.00%	0.06%

VI. CONCLUSIONS

To the best of our knowledge, for the first time, we study the outsourcing of PSO algorithm and its verification. We formulate the problem and its security model. Unlike existing general solutions for verifiable computing relying on expensive cryptography, we propose a simple and efficient scheme. The scheme mainly consists of a verifiable PSO algorithm and its verification algorithm. In the experiments, we discuss the parameters configuration of the verification algorithm, and experimental results validate its verifiability.

REFERENCES

[1] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proc. IEEE International Conference on Neural Networks*, Perth, Australia, Nov. 1995, pp. 1942–1948.

[2] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proc. International Symposium on Micro Machine and Human Science (MHS'95)*, Nagoya, Japan, Oct. 1995, pp. 39–43.

[3] P. J. Angeline, "Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences," in *Proc. International Conference on Evolutionary Programming*, London, UK, Apr. 1998, pp. 601–610.

[4] R. C. Eberhart and Y. Shi, "Comparison between genetic algorithms and particle swarm optimization," in *Proc. International Conference on Evolutionary Programming*, London, UK, Apr. 1998, pp. 611–616.

[5] J. Robinson and Y. Rahmat-Samii, "Particle swarm optimization in electromagnetics," *IEEE Transactions on Antennas And Propagation*, vol. 52, no. 2, pp. 397–407, 2004.

[6] R. Poli, "Analysis of the publications on the applications of particle swarm optimisation," *Journal of Artificial Evolution and Applications*, vol. 2008, pp. 1–10, 2008.

[7] M. Omran, A. P. Engelbrecht, and A. Salman, "Particle swarm optimization method for image clustering," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 19, no. 3, pp. 297–321, 2005.

[8] J. Hettenhausen, A. Lewis, and S. Mostaghim, "Interactive multi-objective particle swarm optimisation with heatmap visualisation based user interface," *Journal of Engineering Optimization*, vol. 42, no. 2, pp. 119–139, 2010.

[9] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. IEEE International Conference on Evolutionary Computation*, Anchorage, USA, May 1998, pp. 69–73.

[10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[11] H. Takabi, J. B. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security Privacy*, vol. 8, no. 6, pp. 24–31, 2010.

[12] Z. Xiao and Y. Xiao, "Security and privacy in cloud computing," *IEEE Communications Surveys Tutorials*, vol. 15, no. 2, pp. 843–859, 2013.

[13] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Proc. Annual Conference on Advances in Cryptology (CRYPTO'10)*, Santa Barbara, CA, USA, 2010, pp. 465–482.

[14] S. Setty, R. McPherson, A. J. Blumberg, and M. Walfish, "Making argument systems for outsourced computation practical (sometimes)," in *Proc. Network and Distributed System Security Symposium (NDSS'12)*, San Diego, CA, USA, Feb. 2012.