Evolution as a Service: A Privacy-Preserving Genetic Algorithm for Combinatorial Optimization

Bowen Zhao, Member, IEEE, Wei-Neng Chen, Senior Member, IEEE, Feng-Feng Wei, Student Member, IEEE, Ximeng Liu, Senior Member, IEEE, Qingqi Pei, Senior Member, IEEE, and Jun Zhang, Fellow, IEEE

Abstract—Evolutionary algorithms (EAs), such as the genetic algorithm (GA), offer an elegant way to handle combinatorial optimization problems (COPs). However, limited by expertise and resources, most users do not have enough capability to implement EAs to solve COPs. An intuitive and promising solution is to outsource evolutionary operations to a cloud server, whilst it suffers from privacy concerns. To this end, this paper proposes a novel computing paradigm, evolution as a service (EaaS), where a cloud server renders evolutionary computation services for users without sacrificing users’ privacy. Inspired by the idea of EaaS, this paper designs PEGA, a novel privacy-preserving GA for COPs. Specifically, PEGA enables users outsourcing COPs to the cloud server holding a competitive GA and approximating the optimal solution in a privacy-preserving manner. PEGA features the following characteristics. First, any user without expertise and enough resources can solve her COPs. Second, PEGA does not leak contents of optimization problems, i.e., users’ privacy. Third, PEGA has the same capability as the conventional GA to approximate the optimal solution. We implements PEGA falling in a twin-server architecture and evaluates it in the traveling salesman problem (TSP, a widely known COP). Particularly, we utilize encryption cryptography to protect users’ privacy and carefully design a suite of secure computing protocols to support evolutionary operators of GA on encrypted data. Privacy analysis demonstrates that PEGA does not disclose the contents of the COP to the cloud server. Experimental evaluation results on four TSP datasets show that PEGA is as effective as the conventional GA in approximating the optimal solution.

Index Terms—Evolutionary computation, evolution as a service, combinatorial optimization, secure computing, privacy protection.

I. INTRODUCTION

E Volutionary algorithms (EAs), such as genetic algorithm (GA), are powerful tools to tackle combinatorial optimization problems (COPs) in science and engineering fields. Many problems faced by science and engineering can be formulated as COPs, such as synthetic biology, transport of goods and planning, production planning [1], [2]. EA has proven to be an powerful tool in handling COPs due to its global probabilistic search ability based on biological evolution such as selection and mutation [3], [4]. Applications of science and engineering have strong requirements for EAs to tackle optimization problems [2].

Limited expertise and resources of common users hinder them from tackling COPs through EAs effectively. In practice, most users facing COPs lack expertise, such as EAs and programming skill for EAs implementation. Also, EAs based on biological evolution require plenty of iterative operations to search the approximate optimal solution, which consumes abundance computing resources. In the sight of users, even though they have the need for COPs, fail to effectively solve the COP due to limited capability and resources.

One promising and elegant solution is that the cloud server renders an evolutionary computing service for users. The cloud server is equipped with sufficient computing and storage resources and can offer convenient and flexible computation services, such as training and inference of machine learning [5]–[8], named machine learning as a service (MLaaS). In MLaaS, users outsource tasks of training or inference to the cloud server and get results. The cloud server performs computing of training or inference. As computing provided by the cloud server, MLaaS does not require users to have expertise and sufficient computing resources. Similarly, users are able to outsource tasks of evolutionary computation to the cloud server and get optimization results even though they lack programming skills for EAs implementation and sufficient resource to perform EAs.

Privacy concerns are critical challenges for outsourcing computation of EAs to the cloud server just like MLaaS [5]–[8]. Optimization results of COPs are private information of users [9]–[11]. For example, optimization results of COPs for synthetic biology, transport of goods and planning, production planning involve private biologic information, planning of goods transportation and production to name but a few. The cloud server is not generally regarded as a trusted entity in an outsourcing computation scenario [9]–[12], such as iCloud leaking celebrity photos, Amazon Web Services exposing Facebook user records. Obviously, no user or company is willing to reveal biologic information and planning of goods transportation and production to others. Moreover, many
regulations stipulate to protect personal data. GDPR stipulates any information relating to an identified or identifiable natural person is private and should be protected. Also, contents of COPs should be regarded as private information. Given contents of the COP, the cloud server holding EAs can obtain the optimization results, which breaches privacy regulation.

To tackle privacy concerns of outsourcing computation of EAs, in this paper, we define a novel computing paradigm, evolution as a service (EaaS), the cloud server rendering evolutionary computing service for users without sacrificing users’ privacy. Broadly speaking, the cloud server encapsulates EAs as a service. Users outsource tasks of evolutionary computation to the cloud server and adopt privacy-preserving methods (e.g., encryption cryptography) to protect privacy. The cloud server performs evolutionary operations and returns optimization results to users. In EaaS, the cloud server cannot learn users’ contents of COP and optimization results. Also, users are not required to have expertise of EAs and sufficient resources. In short, EaaS enables users convenient and flexible solving COPs without sacrificing privacy, which relieves the dilemma between evolutionary computation and privacy concerns.

The vital idea of EaaS is that users outsource encrypted contents of the optimization problem to the cloud server, and the cloud server renders an evolutionary computation service for users over encrypted data. Technically, the implementation of EaaS suffers from several challenges.

First, the cloud server requires to perform evolutionary operations without sacrificing users’ privacy. EA involves basic evolutionary operations including population initialization, evaluation, selection, crossover, and mutation \[9\], \[13\]. The population initialization requires randomly generating several hundred or thousands of individuals, and each individual represents a possible solution. Arguably, when the cloud server has no knowledge about contents of the COP, it is not a trivial task to generate possible solutions. Furthermore, if the cloud server has difficulty in initializing the population, it is also challenging to perform evaluation, selection, crossover, and mutation operations as the latter relies on the initialized population.

Second, the cloud server can evaluate the fitness value of each individual in the population but fails to learn the possible solution. In EAs, the fitness value determines the quality of solutions and is a crucial metric to select dominant individuals. To protect users’ privacy, it should prevent the cloud server from obtaining users’ possible solutions \[9\]. Unfortunately, if the cloud server has no knowledge of possible solutions, it fails to evaluate the fitness values of individuals in the population.

Third, the cloud server can select dominant individuals without knowing individuals’ fitness values. EA is inspired by the process of natural selection, so its critical operation is to select dominant individuals based on individuals’ fitness values. Technically, it requires the cloud server to compare individuals’ fitness values under unknowing them. Intuitively, secure comparison protocols \[14\]–\[16\] seems to provide a potential solution for this. However, the work \[14\] requires two-party holding private data to perform a comparison operation. If the user participates in the comparison operation, it significantly increases the user’s communication overhead as EA needs several hundred or thousands of individuals to generate the approximate optimal solution. The protocols \[15\], \[16\] only generates an encrypted comparison result. Given encrypted comparison results, the cloud server fails to select dominant individuals. In short, selecting dominant individuals has challenges in communications and operations.

To tackle the above challenges, this paper focuses on the implementation of EaaS through GA and carefully designs a privacy-preserving GA, called PEGA\(^2\). Specifically, we exploit the threshold Paillier cryptosystem (THPC) \[17\] and one-way mapping function to protect the user’s privacy. The homomorphism of THPC enables evaluating individuals’ fitness values over encrypted data. Also, we propose a suite of secure computation protocols to support privacy-preserving evolutionary operations of GA, such as selection. Our contributions can be concluded as three-folds.

- We propose a novel computing paradigm, EaaS, a privacy-preserving evolutionary computation paradigm that outsources evolutionary computation to a cloud server. EaaS does not require users to have expertise of EAs and programming skills for EAs implementation but can output the approximate optimal solution for users. Furthermore, EaaS does not leak users’ privacy to the cloud server.
- We carefully design PEGA, a privacy-preserving genetic algorithm based on the computing paradigm EaaS. Particularly, a secure division protocol (SecDiv) and a secure comparison protocol (SecCmp) are presented to support privacy-preserving fitness proportionate selection. SecDiv and SecCmp enable the cloud server computing the probability of each individual being selected and select potentially dominant individuals without disclosing possible solutions, respectively.
- We take four TSP (a widely known COP) datasets (i.e., gr48, kroA100, eil101, and kroB200) to evaluate the effectiveness and efficiency of PEGA. Results of experiments and analyses on four TSP datasets demonstrate that PEGA is as effective as the conventional GA \[18\] in approximating the optimal solution.

The rest of this paper is organized as follows. In Section II, the related work is briefly described. In Section III, we formulate EaaS and PEGA. The design of PEGA is elaborated in Section IV. In Section V, PEGA for TSP is given. Results of privacy analysis and experimental evaluation are shown in Section VI. Finally, we conclude the paper in Section VII.

II. RELATED WORK

In this section, we briefly review privacy-preserving evolutionary algorithms (EAs). In contrast to privacy-preserving neural networks (NNs) inference \[5\], \[8\], privacy-preserving evolutionary algorithms have received little attention. One possible reason is EAs require the server to perform random operations, such as population initialization, mutation,

---

\(^1\)GDPR: General Data Protection Regulation (EU)

\(^2\)PEGA comes from Privacy-preserving Genetic Algorithm
while the operations of NNs are generally deterministic. Also, privacy-preserving NNs inference does not need the server to obtain intermediate results. On the contrary, privacy-preserving EAs require the server to learn intermediate results to perform subsequent operations. For example, the server requires to learn the plaintext comparison result to select dominant individuals.

Sakuma et al. [9] proposed a privacy-preserving GA by means of the idea of secure multi-party computation and the Paillier cryptosystem to solve TSP. The work [9] considers a scenario where multiple servers hold traveling costs while a user wants to choose the server that provides the optimal route; Servers and the user are unwilling to disclose their own private data. Thus, the work [9] requires interaction between the user and servers. Han et al. [10] presented a privacy-preserving GA for rule discovery, where two parties holding datasets jointly perform a privacy-preserving GA to discover a better set of rules in a privacy-preserving manner. The scheme [10] also needs two parties to interact to generate an optimal solution. Funke et al. [11] designed a privacy-preserving multi-object EA based on Yao’s secure two-party protocol [13]. The authors in [11] claim that their solution improves security and efficiency, but their solution still requires two parties to interact. Jiang et al. [19] put forward a cloud-based privacy-preserving GA by means of somewhat homomorphic encryption, where a user outsources operations of GA to the cloud server. However, the work [19] fails to support privacy-preserving selection operations, and no practical problem is involved to evaluate its effectiveness and efficiency. Zhan et al. [20] proposed a rank-based cryptographic function (RCF) to construct privacy-preserving EAs including particle swarm optimization and differential evolution. However, the authors do not construct the RCF and their scheme suffers from some privacy concerns. Although a designer in [20] fails to obtain the fitness function, he holds possible solutions. Thus, as long as the designer learns which solution is dominant, he can obtain the approximate optimal solution, which discloses a user’s privacy.

From the view of existing privacy-preserving EAs, there is no effective solution that provides a privacy-preserving evolution service for users that does not require the user to interact. Motivated by this, we formulate EaaS and give its implementation through GA.

III. Formulation

In this section, we give formal definitions of evolutionary as a service (EaaS) and privacy-preserving genetic algorithm (PEGA), where PEGA is a concrete implementation of EaaS.

A. Formulation for EaaS

Definition 1 (EaaS). EaaS consists of users and a cloud server, where users have a requirement of solving a COP (denoted by \( \mathcal{P} \)) through evolutionary algorithms (EAs), whilst the cloud server holds competitive EAs and sufficient resources to perform EAs. The cloud server encapsulates EAs as a server and renders convenient and flexible evolutionary computing service for users. To avoid exposing privacy to the cloud server, users encrypt the content of the COP denoted by \( \text{Enc}(\mathcal{P}) \) and outsource it to the cloud server. Taking as input \( \text{Enc}(\mathcal{P}) \) and an EA, the cloud server performs evolutionary operations (e.g., evaluation, selection, crossover, mutation) denoted by \( \text{EO}(\text{Enc}(x), \text{Enc}(\mathcal{P})) \) and returns an encrypted optimal solution \( \text{Enc}(\min f(x/\mathcal{P})) \) to the user, where \( \text{Enc}(x) \) indicates an encrypted solution, and \( f(\cdot) \) is the objective function of \( \mathcal{P} \). Formally, EaaS can be formulated as the following pattern

\[
\langle \text{Enc}(\mathcal{P}), f \rangle \rightarrow \text{Enc}(x_i) \rightarrow \text{EO}(\text{Enc}(x_i), \text{Enc}(\mathcal{P})) \rightarrow \text{Enc}(\min f(x/\mathcal{P})),
\]

\[
s.t. x, x_i \in P, i \in [1, n],
\]

where \( n \) is the population size. Fig. 1 shows the structure of EaaS.

From Definition 1 and Fig. 1, we see that EaaS does not ask the user to have the expertise and resources to solve a COP through EAs. To approximate the optimal solution of the COP, the user outsources operations to the cloud server. The cloud server is given encrypted data, so it fails to learn contents of the COP. In other words, EaaS enables the cloud server perform evolutionary operations over encrypted data and generates encrypted optimization solutions to protect the user’s privacy. The key of EaaS is to support evolutionary operations on encrypted data.

B. Formulation for PEGA

To validate the computing paradigm of EaaS, we take GA, a widely known EA, as an example to concrete EaaS, called PEGA. GA usually comprises 5 polynomial time operations: population initialization, evaluation, selection, crossover, and mutation, where the later four are regarded as evolutionary operators [13]. A formal definition of PEGA can be as follow.

Definition 2 (PEGA). A privacy-preserving genetic algorithm (PEGA) takes as input an encrypted COP \( \text{Enc}(\mathcal{P}) \) and its optimization function \( f \), and outputs an encrypted optimiza-
Encrypt content \(\langle \text{Enc}(\mathcal{P}), f \rangle \rightarrow \mathsf{I} \rightarrow \mathsf{E} \rightarrow [\mathcal{S} \rightarrow \mathcal{C} \rightarrow \mathcal{M} \rightarrow \mathsf{E}] \times t \rightarrow \text{Enc}(\min f(x/\mathcal{P})),\)

where \(\times\) indicates the repetition, and \(t\) denotes the iteration times. \(\mathsf{I}, \mathsf{E}, \mathcal{S}, \mathcal{C}, \mathcal{M}\) indicate operations of population initialization, evaluation, selection, crossover, and mutation, respectively. Note that \(\mathsf{I}, \mathsf{E}, \mathcal{S}, \mathcal{C}, \mathcal{M}\) take as input encrypted data and output encrypted data.

From Definition 11 and Definition 2, we see that PEGA is to concrete \(\mathcal{EO}\) as \(\mathcal{E}, \mathcal{S}, \mathcal{C}, \mathcal{M}\). In the next section, we elaborate on the design of PEGA, specially for how to execute evolutionary operations on encrypted data.

IV. PEGA DESIGN

To self-contained, we first list threshold Paillier cryptosystem (THPC) used to encrypt the COP, and then give system model and threat model of PEGA. Next, details of PEGA design are illustrated.

A. Primitive

The detailed algorithms of THPC with \((2, 2)\)-threshold decryption are listed as follows.

Key Generation (KeyGen): Let \(p = 2p' + 1\) and \(q = 2q' + 1\) be two big prime numbers with \(\kappa\) bits (e.g., \(\kappa = 512\)), where \(p', q'\) are also prime numbers. The public key is denoted by \(pk = (p, \mathcal{N})\), where \(N = pq\) and \(g = N + 1\). The private key is denoted by \(sk = (\lambda, \mu)\), where \(\lambda = \text{lcm}(p - 1, q - 1)\) and \(\mu = \lambda^{-1} \mod \mathcal{N}\). Particularly, the private key is split into \(\lambda_1\) and \(\lambda_2\) two partially private keys, s.t., \(\lambda_1 + \lambda_2 = 0 \mod \lambda\) and \(\lambda_1 + \lambda_2 = 1 \mod \mathcal{N}\). As \(\mu = \lambda^{-1} \mod \mathcal{N}\), \(\lambda_1 \mod \lambda\) and \(\lambda_1 \mod \mathcal{N}\). Let \(\lambda_1\) be a random integer in the interval \((0, \lambda\mathcal{N})\) and \(\lambda_2 = \lambda_1 - \lambda_1 \mod \mathcal{N}\).

Encryption (Enc): Take as input a message \(m \in \mathbb{Z}_N\) and \(pk\), and output \([m] = \text{Enc}(pk, m) = (1 + m\mathcal{N}) \cdot r^N \mod \mathcal{N}^2\), where \([m] = [m] \mod \mathcal{N}\) and \(r\) is a random number in \(\mathbb{Z}_N\).

Decryption (Dec): Take as input a ciphertext \([m]\) and \(sk\), and output \(m = \text{Dec}(sk, [m]) = L([m] \lambda \mod \mathcal{N}^2) \cdot \mu \mod \mathcal{N}\), where \(L(x) = \frac{x^N}{N}\).

Partial Decryption (\(P\text{Dec}\)): Take as input a ciphertext \([m]\) and a partially private key \(\lambda_i\) \((i = 1 \text{ or } 2)\), and output \(M_i = P\text{Dec}(\lambda_i, [m]) = [m]^{\lambda_i} \mod \mathcal{N}^2\).

Threshold Decryption (\(T\text{Dec}\)): Take as input partially decrypted ciphertexts \(M_1\) and \(M_2\), and output \(m = T\text{Dec}(M_1, M_2) = L(M_1 \cdot M_2 \mod \mathcal{N}^2)\).

The homomorphic operations on ciphertexts supported by THPC are described as follows.

- **Additive homomorphism:** \(\text{Dec}(sk, [m_1 + m_2 \mod \mathcal{N}]) = \text{Dec}(sk, [m_1]) \cdot \text{Dec}(sk, [m_2])\);
- **Scalar-multiplication homomorphism:** \(\text{Dec}(sk, [c \cdot m \mod \mathcal{N}]) = \text{Dec}(sk, [m])^c\) for \(c \in \mathbb{Z}_N\).

On the basis of additive homomorphism and scalar-multiplication homomorphism, THPC enables subtraction over encrypted data. Specifically, \(\text{Dec}(sk, [m_1 - m_2]) = \text{Dec}(sk, [m_1]) \cdot \text{Dec}(sk, [m_2])^{-1}\).

Note that any single partially private key fails to decrypt any ciphertexts. Also, as operations over ciphertexts encrypted by \(\text{Enc}\) require to perform a \(\mathcal{N}^2\) operation, for brevity, we will omit \(\mathcal{N}^2\) in the rest of this paper. Just like the Paillier cryptosystem (21), THPC only works on integer. To effectively handle floating-point numbers, a given floating-point number \(x\) be encoded as \(\frac{x}{2^\ell} \cdot 2^\ell\), where \(\ell\) is a constant, for example, \(\ell = 53\) is used to encode a 64-bit floating-point number. In this paper, if a message \(m\) to be encrypted is a floating-point number, it is encrypted as \([m] = \left[\frac{m}{2^\ell}\right] \cdot 2^\ell\], i.e., \([m] = \left[\frac{m}{2^\ell}\right] \cdot 2^\ell\]. To simplify notation, we use \([m]\) to denote \([m] \mod \mathcal{N}^2\] in the rest of paper.

B. System Model and Threat Model

In our system, we consider a user outsources an encrypted COP to twin-cloud servers (i.e., \(S_1\) and \(S_2\)). Twin-cloud servers jointly provide a privacy-preserving GA service to solve the encrypted COP through performing secure two-party computations. The user obtains an encrypted optimization solution from \(S_1\). As depicted in Fig. 3 PEGA comprises a user and twin cloud servers.

- **User:** The user has a COP \(\mathcal{P}\) to be solved and outsources the problem to cloud servers with powerful computation and sufficient resources. To protect privacy, the user initializes a public/private pair \((pk, sk)\) of THPC, and then encrypts the problem with the public key \(pk\) as...
Also, in order to enable cloud servers performing evolutionary operators over encrypted data, the user splits the private key $sk$ into two partially private keys $sk_1$ and $sk_2$ and sends them into $S_1$ and $S_2$, respectively.

- **Cloud server 1 ($S_1$):** $S_1$ takes charge of storing $[P]$ sent from the user. $S_1$ and $S_2$ jointly perform secure two-party computation protocols over encrypted data to support the operations of GA. Note that $S_1$ can directly execute certain homomorphic operations (e.g., additive homomorphism and scalar-multiplication homomorphism) over encrypted data supported by THPC.

- **Cloud server 2 ($S_2$):** $S_2$ is responsible for assisting $S_1$ to perform the operations of GA in a privacy-preserving manner.

In the system of PEGA, the computation is outsourced to cloud servers. According to the outsourced computation situation [12], there is one type of adversary that attempts to obtain the user’s private information, i.e., contents of the COP. The adversary involves either obtain the user’s private information, i.e., contents of the COP. or intermediate computation results with the other one, which is unwilling to provide data leakage evidence to other.

**Overview of PEGA**

In this section, we give a high-level description of our proposed PEGA. The goal of PEGA is to perform operations of GA over encrypted data and output an encrypted optimization solution. As shown in Fig. 4, PEGA consists of 5 polynomial-time operations, i.e., GEN, Evaluation, Selection, Crossover, and Mutation, and their briefly description is given as follows.

- **GEN:** Given an encrypted COP $[P]$, GEN generates a population compromising $n$ individuals. Each individual is denoted by a chromosome. Each chromosome consists of $m$ genes. Formally, GEN takes as input $[P]$, and outputs $n$ encrypted chromosomes denoted by $[[x_1],\ldots,[x_n]],$ s.t. $x_i \in P$ and $i \in [1,n].$

- **Evaluation:** Given $n$ encrypted chromosomes, EVALUATION firstly computes the fitness value of each encrypted chromosome. Specifically, EVALUATION utilizes the homomorphism of THPC to obtain encrypted fitness value of each encrypted chromosome according to the optimization function $f$. Formally, EVALUATION takes as input $[[f(x_1)],\ldots,[f(x_n)]]$ and $f$, and outputs $[[f(x_1)],\ldots,[f(x_n)]]$. Also, EVALUATION outputs the optimal chromosome holding minimum fitness value. To this end, we carefully design a secure comparison protocol (SecCmp) that can compare $[f(x_i)]$ and $[f(x_j)]$ ($i,j \in [1,n]$ and $i \neq j$). Formally, given $[f(x_i)]$ and $[f(x_j)]$, SecCmp outputs $[f(x_i)]$, where $f(x_i) \leq f(x_j)$. Thus, given $[[f(x_1)],\ldots,[f(x_n)]]$, EVALUATION can output $[[f(x_k)]]$ via SecCmp, where $[f(x_k)]$, s.t. $f(x_k) = \min\{f(x_1),\ldots,f(x_n)\}$ and $k \in [1,n]$.

- **Selection:** For $n$ encrypted chromosomes, SELECTION explores the well-studied fitness proportionate selection operator [23] to select dominant individuals. Specifically, SELECTION firstly computes an encrypted probability for each individual. After that, SELECTION performs operations of fitness proportionate selection over encrypted probabilities. The critical operations of SELECTION include addition, division, and comparison on encrypted data. To enable division on encrypted data, we propose a secure division protocol (SecDiv). Formally, given $[a]$ and $[b]$, SecDiv outputs $[a/b]$.

- **Crossover:** Choosing two chromosomes as parents, CROSSOVER performs a crossover operator to enable two parents crossover generating children. Roughly speaking, CROSSOVER exchanges some genes of the chromosome to generate new chromosomes.

- **Mutation:** For each encrypted chromosome, MUTATION exchanges some genes of the chromosome to generate a new chromosome.

When iterations is used as the termination condition, except for GEN, Initial Pop, Evaluation, Selection, Crossover, and Mutation require to repeat. PEGA takes encrypted data as input, generates encrypted intermediate results, and outputs encrypted optimization solution to protect privacy.

**D. Privacy-preserving Protocols for PEGA**

In this section, we first elaborate on the secure division protocol (SecDiv) and the secure comparison protocol (SecCmp) that are used to construct PEGA. Next, through SecDiv and SecCmp, we design a secure probability algorithm (SecPro) and a secure fitness proportionate selection algorithm (SecFPS) to support SELECTION on encrypted data. Also, SecCmp enables EVALUATION over encrypted data.

1) **Secure Division Protocol (SecDiv):** Given $[x]$ and $[y]$, where $y \neq 0$, SecDiv outputs $[x/y]$. The key idea of SecDiv is to convert division to scalar multiplication. Specifically, for any integers $x,y$, we have $\frac{x}{y} = \frac{x}{2^i}$, where $y = 2^i \cdot 2^k$ and $y$ is an integer. Formally, SecDiv($[x],[y]) \rightarrow [\frac{x}{y} \cdot 2^k]$. SecDiv consists of three steps.

1. $S_1$ calls PDec to partially decrypt $[y]$ to get $[y']$. Next, $S_1$ sends $([y],[y'])$ to $S_2$.
2. $S_2$ calls PDec to partially decrypt $[y']$ to get $[y'']$ and then calls TDec to obtain $y$ with $[y'']$ and $[y']$. After that, $S_2$ computes $\text{den} = \frac{1}{y'}$. Next, $S_2$ encodes $\text{den}$ as $y = \frac{\text{den} \cdot 2^k}{y'}$. Finally, $S_2$ returns $y$ to $S_1$. Clearly, $y$ is an integer.
3. $S_1$ computes $[\frac{x}{y}] \leftarrow [x'y']$. 
2) Secure Comparison Protocol (SecCmp): Given \([x]\) and \([y]\), SecCmp outputs 0 when \(x \geq y\), 1 otherwise (\(x < y\)). Formally, \(\text{SecCmp}([x], [y]) \rightarrow \{0, 1\}\). SecCmp consists of three steps.

(1) \(S_1\) generates a random number \(\pi \in \{0, 1\}\) through tossing a coin. \(S_1\) computes

\[
\Delta = \begin{cases} \left(\left(\frac{[x]}{[y]}\right)^{-1}\right)^{r_1} \cdot [r_1 + r_2], & \text{for } \pi = 0 \\
\left(\left(\frac{[x]}{[y]}\right)^{-1}\right)^{r_2}, & \text{for } \pi = 1 
\end{cases}
\]

where \(r_1, r_2\) are two randomly integers, s.t., \(r_1 \leftarrow \{0, 1\}^\sigma \setminus \{0\}\), \(r_1 + r_2 > \frac{N}{3}\), and \(r_2 \leq \frac{N}{3}\). \(\sigma\) is secure parameter, e.g., \(\sigma = 128\). Next, \(S_1\) calls \(\text{PDec}\) to get \([\Delta_1]\), and sends \([\Delta_1], [\Delta_2]\) to \(S_2\).

(2) \(S_2\) first calls \(\text{PDec}\) to get \([\Delta_2]\) and then calls \(\text{TDec}\) to obtain \(\Delta\) with \([\Delta_1]\) and \([\Delta_2]\). If \(\Delta > \frac{N}{3}\), \(S_2\) sets \(u = 0\), otherwise, \(u = 1\). Finally, \(S_2\) sends \(u\) to \(S_1\).

(3) \(S_1\) obtains the comparison result by computing \(\pi \oplus u\).

When \(\pi = 0\), we have \(\pi \oplus u = 0\) when \(u = 0\), otherwise, \(\pi \oplus u = 1\). When \(\pi = 1\), we have \(\pi \oplus u = 1\) when \(u = 0\), otherwise, \(\pi \oplus u = 0\).

Clearly, given \([\{f(x_1), \cdots, f(x_n)\}]\), it is easy to implement EVALUATION by calling \(\text{SecCmp}\). Specifically, EVALUATION performs \(n\) comparison operations on encrypted data to obtain the optimal chromosome holding minimum fitness values.

For brevity, we utilize \(v_i\) to denote \(f(x_i)\). According to the fitness proportionate selection operator [2], it requires to compute each individual’s probability. Thus, the individual’s probability is denoted by

\[
p_i \leftarrow \frac{v_i}{\sum_{j=1}^{n} v_j},
\]

However, to protect users’ privacy, the cloud server only obtains \([v_i]\). Given \([\{v_1, \cdots, v_n\}]\), it is not trivial for the cloud server to compute \(\sum_{j=1}^{n} v_j\). Fortunately, the proposed SecDiv offers a potential solution. Specifically, through the proposed SecDiv, we design a secure probability algorithm (SecPro) to compute each individual’s probability on encrypted data. Given \(n\) encrypted fitness values \([\{v_1, \cdots, v_n\}]\), SecPro outputs \(n\) encrypted probabilities \([\{p_1, \cdots, p_n\}]\). Formally, \(\text{SecPro}([\{v_1, \cdots, v_n\}]) \rightarrow [\{p_1, \cdots, p_n\}]\).

![Image](image.png)

**Algorithm 1:** SecPro\(([\{D_1, \cdots, D_n\}], [\text{sum}]) \rightarrow [\{p_1, \cdots, p_n\}]\).

1. **Input:** \(S_1\) has \([\{v_1, \cdots, v_n\}]\).
2. **Output:** \(S_1\) obtains \([\{p_1, \cdots, p_n\}]\).
3. **Step 1.** \(S_1\) computes
   - \([\text{sum}] \leftarrow \prod_{i=1}^{n} [v_i]\);
   - \([\{\text{sum}\}]_1 \leftarrow \text{PDec}(sk_1, [\text{sum}]_1);
   - and sends \([\{\text{sum}, [\text{sum}]_1\}]\) to \(S_2\).
4. **Step 2.** \(S_2\) computes
   - \([\{\text{sum}\}_2 \leftarrow \text{PDec}(sk_2, [\text{sum}]_1)]\) and \([\text{sum}] \leftarrow \text{TDec}([\{\text{sum}\}_1], [\{\text{sum}\}_2])\);
   - \([\text{den}] = \frac{\text{sum}}{\text{sum}}, [\text{den}] = \frac{1}{[\text{den}]}, \text{and } D = \frac{\text{den} \cdot 2^\ell};\)
   - and sends \(D\) to \(S_1\).
5. **Step 3.** \(S_1\) computes
   - \([p_i] \leftarrow [v_i]^D\) for \(i \in [1, n]\).

From Algorithm [1] we see that \(\text{Dec}(sk, [p_i]) = \frac{v_i}{\sum_{j=1}^{n} v_j} \cdot 2^{\ell}\). If \(\sum_{j=1}^{n} v_j > \sum_{i=1}^{n} v_i\), we have \(\text{Dec}(sk, [p_i]) > \text{Dec}(sk, [p_i])\). In other words, Algorithm [1] does not change the numerical relationship among probabilities of individuals.

Also, to enable fitness proportionate selection on encrypted data, we construct a secure fitness proportionate selection algorithm (SecFPS) via SecCmp. Given \(n\) encrypted probabilities \([\{p_1, \cdots, p_n\}]\), SecFPS outputs \(n\) individuals. The key idea of SecFPS is to perform \(n\) comparison operations over encrypted data. Formally, \(\text{SecFPS}([\{p_1, \cdots, p_n\}] \rightarrow \text{Pop}\), where \(\text{Pop}\) represents a population consisting of \(n\) individuals. As shown in Algorithm [2], SecFPS consists of three steps.

1. \(S_2\) generates \(n\) encrypted random numbers
Algorithm 2: SecFPS(\{\{p_1\}, \ldots, \{p_n\}\}) \rightarrow Pop.

1 Input: \(S_1\) has \(\{\{p_1\}, \ldots, \{p_n\}\}\).
2 Output: \(S_1\) obtains \(Pop\).
3 \(S_2\) computes \(\{r_i\} \leftarrow Enc(pk, r_i \cdot 2^{2\ell})\) for \(i \in [1, n]\),
   where \(r_i\) is a random number in \((0, 1)\) and then sends \(\{\{r_i\}, \ldots, \{r_n\}\}\) to \(S_1\);
4 for \(i = 2\) to \(n\) do
5  \(S_1\) computes \(\{p_i\} \leftarrow \{p_{i-1}\} \cdot \{p_i\}\);
6 end
7 for \(j = 1\) to \(n\) do
8  \(S_1\) and \(S_2\) jointly perform
9    \(i \leftarrow \text{FINDINDIVIDUAL}(\{p_j\}, 1, n)\);
10   \(S_1\) adds \(i\) to \(Pop\);
11 end
12
13 \text{FINDINDIVIDUAL}\((\{p\}, \text{low}, \text{high})\) begin
14  \(i \leftarrow \left[ \frac{\text{low} + \text{high}}{2} \right] ;\)
15  if \(\text{SecCmp}(\{p_i\}, \{r\})\) returns 1 then
16    return \(\text{FINDINDIVIDUAL}(\{p\}, i + 1, \text{high})\);
17 end
18 if \(i < 2\) then
19    return 1;
20 end
21 else if \(\text{SecCmp}(\{p_{i-1}\}, \{r\})\) returns 1 then
22    return \(i\);
23 end
24 else
25    return \(\text{FINDINDIVIDUAL}(\{p\}, \text{low}, i - 1)\);
26 end
27 end
28
29

Note that the proposed SecCmp can be used to construct secure selection operators, such as secure tournament selection, secure elitism selection. The critical operation for tournament selection and elitism selection is to compare fitness values of individuals [23], which is supported by SecCmp.

V. PEGA FOR TSP

This section takes TSP, a widely known COP, as an example to demonstrate the idea of EaaS through the proposed PEGA.
Fig. 5 gives the storage structure of encrypted TSP. Specifically, the first row and the first column of $\begin{bmatrix} M \end{bmatrix}$ denotes the city index. $d_{i,j}$ represents the traveling cost between city $i$ and city $j$. "0" indicates that the two cities are unreachable, whereas it indicates two cities are reachable. Assume the size of cities be $m$, the objective function $f$ can be denoted by

$$f(x) = d_{i,j} + d_{j,k} + \cdots + d_{k,i},$$

where $x$ is a possible route. Finally, a user outsources $\langle \begin{bmatrix} M \end{bmatrix}, f(\cdot) \rangle$ to $S_1$.

### B. Problem Solving via PEGA

In this section, we elaborate on how to solve an encrypted TSP through PEGA.

1) Initialization: Given $\begin{bmatrix} M \end{bmatrix}$, $S_1$ initializes $n$ encrypted chromosomes denoted by $\langle [x_1], \cdots, [x_n] \rangle$, where $[x_i]$ is denoted by an array of index of $\begin{bmatrix} M \end{bmatrix}$, such as "9-5-7-4". As the one-way function $\phi(\cdot)$ is adopted, the index of $\begin{bmatrix} M \end{bmatrix}$ does not disclose the city. Thus, given $\begin{bmatrix} M \end{bmatrix}$, $S_1$ is able to generate $n$ encrypted chromosomes to initialize a population.

2) Evaluation: Given $\begin{bmatrix} M \end{bmatrix}$ and $\langle [x_1], \cdots, [x_n] \rangle$, $S_1$ can compute $\langle [f(x_1)], \cdots, [f(x_n)] \rangle$. Specifically, without loss of generality, let $[x_i]$ be denoted by "i-j-k", $S_1$ computes an encrypted fitness value $[v_i]$ as

$$[v_i] \leftarrow [d_{i,j}] \cdots [d_{k,i}].$$

As the additive homomorphism of THPC, we see that $v_i = d_{i,j} + \cdots + d_{k,i}$. Thus, $S_1$ can calculate and generate $\langle [v_1], \cdots, [v_n] \rangle$. Next, $S_1$ and $S_2$ jointly compute and find out the encrypted chromosome holding minimum fitness value by calling SecCmp. Specifically, without loss of generality, assume $v_i = \min\{v_1, \cdots, v_n\}$, i.e., $v_i = \sum_{j=1}^{n} v_j$. $S_1$ outputs $[v_i]$ denoted by "i-j-k" and sets it as the optimal chromosome.

3) Selection: $S_1$ can choose different selection operators, such as fitness proportionate selection, tournament selection, elitism selection, to perform a selection operator. In here, we consider $S_1$ utilizes the fitness proportionate selection operator as the selection operator. Specifically, $S_1$ firstly cooperates with $S_2$ to obtain $\langle [p_1], \cdots, [p_n] \rangle$ by calling SecPro, where $p_i$ is the probability of the individual $i$ ($i \in [1, n]$). After that, $S_1$ teams with $S_2$ to generate a new population by calling SecFPS.

4) Crossover: Given encrypted chromosomes $\langle [x_1], \cdots, [x_n] \rangle$, $S_1$ can adopt the conventional crossover operator (such as edge recombination crossover operator, ERX [18]) to generate children. Assume $S_1$ chooses $[x_i]$ denoted by "i-j-k" and $[x_j]$ denoted by "j-i-k" as two parent chromosomes, it is easy for $S_1$ to generate two children by calling ERX [18].

5) Mutation: Given encrypted chromosomes, $S_1$ is able to perform mutation operations on $[x_i]$. Specifically, $S_1$ can change the element of $[x_i]$.

### VI. RESULTS OF PRIVACY ANALYSIS AND EXPERIMENT

#### A. Privacy Analysis

THPC [17] have been proved to be semantically secure. Thus, the homomorphic operations performed by $S_1$ do not disclose the user’s private data. In this paper, we carefully design SecDiv and SecCmp based on a non-colluding twin-server architecture to select dominant individuals in a privacy-preserving manner. In this section, we demonstrate SecDiv and SecCmp are secure to perform division and comparison over encrypted data.

**Theorem 1.** Given $[x]$ and $[y]$, where $y \neq 0$, SecDiv does not disclose $\frac{x}{y}$.

**Proof.** Given $[x]$ and $y (y \neq 0)$, SecDiv computes $[x]^\frac{1}{y} \cdot 2^t$ to produce $[\frac{x}{y}]$. When $t$ is large enough, $\frac{1}{y} \cdot 2^t$ must be an integer. Without loss of the generality, let $c = \frac{1}{y} \cdot 2^t$, we have $c \in \mathbb{Z}_N$. Thus, SecDiv essentially is to perform one scalar multiplication operation. As THPC is semantically secure, $[y]^c$ does not leak $cm$. Therefore, SecDiv does not disclose $\frac{x}{y}$.

**Lemma 1.** SecPro can produce each individual’s encrypted fitness value and encrypted probability without leaking the individual’s city list and route cost.

**Proof.** According to Algorithm [7] we see $[v_i] \leftarrow \text{SecDiv}(\sum_i [D_i]^{-1}, \sum_i)$, where $\sum_i$ is the sum of all individuals’ route costs, and $D_i$ is the individual $i$ route cost. As Theorem [7] holds, SecPro can securely compute encrypted fitness values. Also, we have $[p_i] \leftarrow \text{SecDiv}([v_i], \sum_{i=1}^{n} [v_i])$. Thus, we say that SecPro can securely compute encrypted probabilities when Theorem [7] holds.

**Theorem 2.** Given $[x]$ and $[y]$, SecCmp does not disclose $x$ and $y$.

**Proof.** In the view of $S_1$, he only learns encrypted data, so SecCmp does not disclose $x$ and $y$ to $S_1$ as THPC is semantically secure. In the view of $S_2$, he can learn $r_1 \cdot (x-y+1)+r_2 (\pi = 0)$ or $r_1 \cdot (y-x)+r_2 (\pi = 1)$. However, as $r_1$ and $r_2$ are unkown for $S_2$, given either $r_1 \cdot (x-y+1)+r_2$ or $r_1 \cdot (y-x)+r_2$, $S_2$ fails to get $x, y, x-y+1, y-x$. Thus, SecCmp does not leak $x$ and $y$ to $S_2$. In short, SecCmp does not disclose $x$ and $y$. Furthermore, even though $S_2$ knows $y$, he cannot get $x$ as fails to know $r_1$ and $r_2$.

**Lemma 2.** SecFPS can select dominant individuals over encrypted data without leaking the individual’s probability.

**Proof.** From Algorithm [2] we see although $S_1$ can obtain $r_1 \leq p_i$ or $r_i > p_i$ ($i \in [1, n]$), he fails to know $r_1$. Thus, we say that $S_1$ fails to learn $p_i$. Also, $S_2$ can get $r_1 \cdot (r_i - p_i + 1) + r_2$ or $r_1 \cdot (p_i - r_i) + r_2$, but he fails to obtain $p_i$ as Theorem [2] holds. In short, SecFPS does not disclose the individual’s probability.

#### B. Experimental Evaluation

In this section, we evaluate the effectiveness of PEGA by comparing it with two conventional GA variants [18] and
Experimental results are shown in Fig. 6. The first experiment compares the performance between GA1 and GA2. Note that GA1 and GA2 utilize the ERX operator as the crossover operator. Roughly speaking, SecFPS based on SecCmp and SecDiv enables the fitness proportionate selection. Also, SecCmp naturally supports the k-tournament selection.

The experiments are executed on four most widely used TSP datasets1(i.e., gr48, kroA100, eil101, kroB200), where gr48 and kroA100 are small scale, while eil101 and kroB are medium scale [24]. We implement PEGA and conventional GA variants in Java. The experiment is performed on a personal computer running windows 10-64bit with an Intel Core i7-4790 CPU 3.6 GHz processor and 16 GB memory, which acts as the user. Also, the server running windows 10 64 bit with an Intel Core i7-10700 CPU 2.9 GHz processor, and 32 GB memory, which simulates two cloud servers. Since GA is a stochastic approach, 30 independent runs are executed for each algorithm to generate an average. Experimental settings are listed in Table I where \(|x|\) denotes the length of \(x\) in bits. The crossover rate and the mutation rate use settings in [23], [24].

### C. Effectiveness Evaluation

Given four TSPs, i.e., gr48, kroA100, eil101, and kroB200, we firstly compare the performance between GA1 and GA2. Experimental results are shown in Fig. 6. The x-axis is the number of generations and the y-axis is the path length of routing. The red solid line and blue dashed line represent GA1 and GA2, respectively. As depicted in Fig. 6, we see that GA2 is remarkably superior to GA1 in terms of convergence. Specifically, in contrast to GA1, GA2 always converges to a smaller path length of routing in four given TSPs. In other words, GA2 has a stronger ability in approximating the optimal solution than GA1. Thus, we argue that k-tournament selection outperforms fitness proportionate selection for TSPs. One possible explanation is that k-tournament selection always selects a dominant individual into next generation, whilst poor individuals are possible to be selected by the fitness proportionate selection.

1http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/
random numbers during performing evolution operators.

To visualize the above conclusion, we plot convergence curves of PEGA1, GA1, PEGA2, and GA2 on four TSPs shown in Fig. 7. The x–axis is the number of generations and the y–axis is the path length of routing. The red solid line and blue solid line represent GA1 and GA2, respectively. Cyan dashed line and black dashed line represent PEGA1 and PEGA2, respectively. Fig. 7 visually shows that PEGA1 and GA1 have the same convergence trend, and PEGA2 and GA2 has the same convergence trend. As shown in Table II and Fig. 7, we argue that PEGA is as effective as GA for TSPs in approximating the optimal solution.

To further demonstrate the effectiveness of PEGA, we make PEGA1 and PEGA2 use the same random numbers with GA1 and GA2 to perform evolution operations, respectively. The experimental results are given in Fig. 8. Magenta circle and cyan circle represent GA1 and GA2, respectively. Blue solid line and black solid line represent PEGA1 and PEGA2, respectively. From Fig. 8, we see that PEGA1 and GA1 has the same convergence, and PEGA2 and GA2 has the same convergence, when the same random numbers are adopted. This is because our proposed secure computing protocols support exactly computations on encrypted data. In fact, Fig. 8 illustrates that PEGA is as effective as GA. In other words, given encrypted TSPs, PEGA can effectively approximate the optimal solution as GA.

D. Efficiency Evaluation

In this section, we evaluate the efficiency of PEGA in terms of communication cost and computation cost. Table III shows the comparison results of communication cost between PEGA and GA. From Table III we see that PEGA has a larger communication cost than GA. In PEGA, a user submits an encrypted TSP matrix, i.e., [M]. On the contrary, the user in GA submits the TSP matrix M directly. As the ciphertext of THPC is significantly larger than its plaintext, the communication cost of PEGA is larger than that of GA. Also, when a small N is set, it can significantly reduce the communication cost of PEGA. One possible explanation is that the smaller N, the smaller the ciphertext size of THPC is. As shown in Table III we see that even a large TSP (e.g., kroB200) and a large N are set, the communication cost of PEGA is less than 6 MB.

Assume there be n individuals and m cities (n > m). A GA consists of \textsc{Gen\_Initial\_Pop}, \textsc{Evaluation}, \textsc{Selection}, \textsc{Crossover}, and \textsc{Mutation}. \textsc{Gen\_Initial\_Pop} initializes n m–dimension individuals, so its computational complexity is \(O(mn)\). \textsc{Evaluation} is to compute each individual’s route cost, so its computational complexity is also \(O(mn)\). \textsc{Selection} generally selects n new individuals via a proportionate selection operator. The computational complexity of the conventional fitness proportionate selection operator is \(O(n^2)\). In this paper, PEGA adopts the idea of binary search to select new individuals, and its computational complexity is \(O(n \log n)\). Thus, PEGA improves the performance of \textsc{Selection} comparing to conventional GA. In this paper, we adopt ERX to perform \textsc{Crossover}. The computational complexity of ERX is \(O(m)\). The computational complexity of \textsc{Mutation} is \(O(n)\) for a population with n individuals. Thus, we see that the computational complexity of GA is \(O(n^2t + \sum_{i=1}^{t} O(mn_i))\), while that of PEGA is \(O(n \log nt + \sum_{i=1}^{t} O(mn_i))\), where t is the number of generations.

In contrary to GA, PEGA requires to encrypt the TSP matrix \([M]\). Fig. 9 shows the runtime of encryption and searching for the optimal route of PEGA, where the runtime of searching is running one generation. From Fig. 9 we can learn that
the more cities, the more runtime of encryption for PEGA is. PEGA requires encrypting reachable routes between two cities. The more cities, the more reachable routes between two cities are. Also, PEGA takes around 3 s to produce a potential solution when k-tournament selection is adopted. Furthermore, for four TSPs, PEGA takes almost the same runtime to produce a possible solution. For PEGA, SELECTION performs most computations on encrypted data, and computations on encrypted data are time-consuming than computations on plaintext data.

As the computational complexity of SELECTION of PEGA is $O(n \log n)$, when $n$ is the same, PEGA takes almost the same runtime to produce a possible solution for four different TSPs. From Fig. 9 we also see that the fitness proportionate selection consumes more runtime to generate a possible solution. One possible explanation is that the fitness proportionate selection operator requires more operations on encrypted data than k-tournament selection operator.

VII. CONCLUSION

In this paper, we proposed the computing paradigm of evolution as a service (EaaS) and designed a privacy-preserving genetic algorithm for COPs based on EaaS, called PEGA. To show the effectiveness and efficiency of PEGA, we use the widely known TSP to evaluate PEGA. In PEGA, a user encrypts her TSP matrix to protect the privacy and outsources the evolutionary computations to cloud servers. The cloud server performs evolutionary computations over encrypted data and produces an effective solution as conventional GA.

To support operations on encrypted TSPs, this paper presented a secure division protocol (SecDiv) and a secure comparison protocol (SecCmp) falling in the twin-server architecture. Experimental evaluations on four TSPs (i.e., gr48, KroA100, eil101, and KroB200) show that there is no significant difference between PEGA and conventional GA. Also, given encrypted TSPs, PEGA with k-tournament selection operator can produce one potential solution around 3 s. For future work, we will extend the idea of EaaS to other algorithms, such as particle swarm optimization (PSO), ant colony optimization (ACO).

| Problems | Scale | Statistical tests | PEGA1 | GA1 | PEGA2 | GA2 |
|----------|-------|-------------------|-------|-----|-------|-----|
| gr48     | small | mean              | 6.2071e+03 | 6.3138e+03 | 5.2949e+03 | 5.3033e+03 |
|          |       | std               | 515.74  | 530.22 | 119.78 | 98.32 |
|          |       | p-value            | 0.3183  | 0.3615 | 1.0    |       |
| kroA100  | small | mean              | 6.8017e+04 | 6.8961e+04 | 2.2819e+04 | 2.3175e+04 |
|          |       | std               | 2.4004e+03 | 1.9935e+03 | 619.31  | 755.14 |
|          |       | p-value            | 0.3615  | 0.5475 | 0.1150 |       |
| eil101   | medium| mean              | 1.4739e+03 | 1.4431e+03 | 868.4667 | 683.8667 |
|          |       | std               | 45.85   | 82.73  | 15.68  | 9.99  |
|          |       | p-value            | 0.5475  | 0.5746 |       |       |
| kroB200  | medium| mean              | 1.7723e+05 | 1.7895e+05 | 3.3878e+04 | 3.3775e+04 |
|          |       | std               | 5.8435e+03 | 3.6344e+03 | 761.45  | 615.0  |
|          |       | p-value            | 0.4679  | 0.8419 |       |       |

Note. †: the length of public key $N$ of Paillier cryptosystem in bits in PEGA is 128; ‡: the length of public key $N$ of Paillier cryptosystem in bits in PEGA is 256;

Table II: Comparison Results between PEGA and GA (The average result of 30 independent runs)

Table III: Comparison of communication cost between PEGA and GA (The average result of 30 independent runs)

| Problems | Scale | GA | PEGA | PEGA1 |
|----------|-------|----|------|-------|
| gr48     | small | 7 KB | 173 KB | 373 KB |
| kroA100  | small | 33 KB | 132 KB | 173 KB |
| eil101   | medium| 25 KB | 5.76 MB | 0.74 MB |
| kroB200  | medium| 132 KB | 2.98 MB | 5.90 MB |

REFERENCES

[1] G. Naseri and M. A. Koffas, “Application of combinatorial optimization strategies in synthetic biology,” Nature Communications, vol. 11, no. 1, pp. 1–14, 2020.
[2] V. T. Paschos, Applications of combinatorial optimization (2nd Edition). John Wiley & Sons, 2017.
[3] R. Wang and Z. Zhang, “Set theory-based operator design in evolutionary algorithms for solving knapsack problems,” IEEE Transactions on Evolutionary Computation, vol. 25, no. 6, pp. 1133–1147, 2021.
[4] A. Radhakrishnan and G. Jeyakumar, “Evolutionary algorithm for solving combinatorial optimization—A review,” Innovations in Computer Science and Engineering, pp. 539–545, 2021.
[5] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, “Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy,” in Proceedings of International Conference on Machine Learning. PMLR, 2016, pp. 201–210.
[6] P. Mohassel and J. Rindal, “ABY3: A mixed protocol framework for machine learning,” in Proceedings of ACM SIGSAC conference on computer and communications security, 2018, pp. 35–52.
[7] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, “Delphi: A cryptographic inference service for neural networks,” in Proceedings of USENIX Security Symposium, 2020, pp. 2505–2522.
[8] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, “CryptTFlow2: Practical 2-party secure inference,” in Proceedings of ACM Conference on Computer and Communications Security, 2020, pp. 325–342.
[9] J. Sakuma and S. Kobayashi, “A genetic algorithm for privacy preserving combinatorial optimization,” in Proceedings of ACM Conference on Genetic and Evolutionary Computation, 2007, pp. 1372–1379.

[10] S. Han and W. K. Ng, “Privacy-preserving genetic algorithms for rule discovery,” in Proceedings of International Conference on Data Warehousing and Knowledge Discovery. Springer, 2007, pp. 407–417.

[11] D. Funke and F. Kerschbaum, “Privacy-preserving multi-objective evolutionary algorithms,” in Proceedings of International Conference on Parallel Problem Solving from Nature. Springer, 2010, pp. 41–50.

[12] Z. Shan, K. Ren, M. Blanton, and C. Wang, “Practical secure computation outsourcing: A survey,” ACM Computing Surveys, vol. 51, no. 2, pp. 1–40, 2018.

[13] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, “Automatically designing cnn architectures using the genetic algorithm for image classification,” IEEE transactions on cybernetics, vol. 50, no. 9, pp. 3840–3854, 2020.

[14] A. C. Yao, “Protocols for secure computations,” in Proceedings of the Annual Symposium on Foundations of Computer Science. IEEE, 1982, pp. 160–164.

[15] T. Veugen, “Encrypted integer division and secure comparison,” International Journal of Applied Cryptography, vol. 3, no. 2, pp. 166–180, 2014.

[16] T. Veugen, F. Blom, S. J. de Hoogh, and Z. Erkin, “Secure comparison protocols in the semi-honest model,” IEEE Journal of Selected Topics in Signal Processing, vol. 9, no. 7, pp. 1217–1228, 2015.

[17] A. Lysyanskaya and C. Peikert, “Adaptive security in the threshold setting: From cryptosystems to signature schemes,” in Proceedings of International Conference on the Theory and Application of Cryptology and Information Security. Springer, 2001, pp. 331–350.

[18] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, “Genetic algorithms for the travelling salesman problem: A review of representations and operators,” Artificial intelligence review, vol. 13, no. 2, pp. 129–170, 1999.

[19] L. Jiang and Z. Fu, “Privacy-preserving genetic algorithm outsourcing in cloud computing,” Journal of Cybersecurity, vol. 2, no. 1, p. 49, 2020.

[20] Z.-H. Zhan, S.-H. Wu, and J. Zhang, “A new evolutionary computation framework for privacy-preserving optimization,” in Proceedings of International Conference on Advanced Computational Intelligence. IEEE, 2021, pp. 220–226.

[21] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 1999, pp. 223–238.

[22] P. Molhasel and Y. Zhang, “Secureml: A system for scalable privacy-preserving machine learning,” in Proceedings of IEEE symposium on security and privacy. IEEE, 2017, pp. 19–38.

[23] J. Zhong, X. Hu, M. Gu, and J. Zhang, “Comparison of performance between different selection strategies on simple genetic algorithms,” in International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, vol. 2. IEEE, 2005, pp. 1115–1121.

[24] F.-F. Wei, W.-N. Chen, X.-M. Hu, and J. Zhang, “An empirical study on evolutionary algorithms for traveling salesman problem,” in Proceedings of International Conference on Information Science and Technology. IEEE, 2019, pp. 273–280.

Wei-Neng Chen (S’07–M’12–SM’17) received the bachelor’s and Ph.D. degrees from Sun Yat-sen University, Guangzhou, China, in 2006 and 2012, respectively. He is currently a Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou. He has co-authored over 90 papers in international journals and conferences, including over 30 papers in IEEE TRANSACTIONS journals. His current research interests include swarm intelligence algorithms and their applications on cloud computing, operations research, and software engineering.

Dr. Chen was a recipient of the National Science Fund for Excellent Young Scholars in 2016 and the IEEE Computational Intelligence Society Outstanding Dissertation Award in 2016 for his doctoral thesis. He is the Vice-Chair of the IEEE Guangzhou Section.

Feng-Feng Wei received her Bachelor’s degree in computer science from South China University of Technology, Guangzhou, China, in 2019, where she is currently pursuing the Ph.D. degree. Her current research interests include evolutionary computation algorithms and their applications on expensive and distributed optimization in real-world problems.

Ximeng Liu (S’13–M’16–SM’21) received the B.Sc. degree in electronic engineering from Xidian University, Xi’an, China, in 2010 and the Ph.D. degree in Cryptography from Xidian University, China, in 2015. Now he is the full professor in the College of Computer Science and Data Science, Fuzhou University. Also, he was a research fellow at Peng Cheng Laboratory, Shenzhen, China. He has published more than 200 papers on the topics of cloud security and big data security including papers in IEEE TOC, IEEE TII, IEEE TDSIC, IEEE TSC, IEEE IoT Journal, and so on. He awards “Minnjiang Scholars” Distinguished Professor, “Qishan Scholars” in Fuzhou University, and ACM SIGSAC China Rising Star Award (2018). His research interests include cloud security, applied cryptography and big data security.

Bowen Zhao (M’22) received his Ph.D. degree in cyberspace security from South China University of Technology, China, in 2020. He was a research scientist in the school of computing and information systems, Singapore Management University, from 2020 to 2021. Now, he is an associate professor at Guangzhou Institute of Technology, Xidian University, Guangzhou, China. His current research interests include privacy-preserving computation and learning and privacy-preserving crowdsensing.

Qingqi Pei (SM’22) received his B.S., M.S. and Ph.D. degrees in Computer Science and Cryptography from Xidian University, in 1998, 2005 and 2008, respectively. He is now a Professor and member of the State Key Laboratory of Integrated Services Networks, also a Professional Member of ACM and Senior Member of IEEE, Senior Member of Chinese Institute of Electronics and China Computer Federation. His research interests focus on digital contents protection and wireless networks and security.
Qing Li (Senior Member, IEEE) received the B.Eng. degree in computer science from Hunan University, Changsha, China, in 1982, and the M.Sc. and Ph.D. degrees in computer science from the University of Southern California, Los Angeles, CA, USA, in 1988. He is a Chair Professor with the Department of Computing, Hong Kong Polytechnic University, Hong Kong. His current research interests include multimodal data management, conceptual data modeling, social media, Web services, and e-learning systems. He has authored/coauthored over 400 publications in the above areas.

Xiao-Nan Luo received the B.S. degree from Jiangxi Normal University, Nanchang, China, in 1983, the M.S. degree from Xidian University, Xi’an, China, in 1985, and the Ph.D. degree from the Dalian University of Technology, Dalian, China, in 1991. He is currently a Professor with the School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin, China. His current research interests include computer graphics, computer-aided design, image processing, and mobile computing.

Hua-Qiang Yuan received the Ph.D. degree from Shanghai Jiao Tong University, Shanghai, China, in 1996. He is currently a Professor with the School of Computer Science and Network Security, Dongguan University of Technology, Dongguan, China. His current research interests include computational intelligence and cyberspace security.

Jun Zhang (M’02–SM’08–F’17) received the Ph.D. degree from the City University of Hong Kong, Hong Kong, in 2002. He is currently a Professor, Hanyang University, South Korea. His current research interests include computational intelligence, cloud computing, high-performance computing, operations research, and power electronic circuits. Dr. Zhang was a recipient of the Changjiang Chair Professor from the Ministry of Education, China, in 2013, the China National Funds for Distinguished Young Scientists from the National Natural Science Foundation of China in 2011, and the First-Grade Award in Natural Science Research from the Ministry of Education, China, in 2009. He is currently an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE TRANSACTIONS ON CYBERNETICS, and the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS.