A Privacy-Preserving Reinforcement Learning Approach for Dynamic Treatment Regimes on Health Data

Xiaoqiang Sun,1,2 Zhiwei Sun,3 Ting Wang,3 Jie Feng,4,5 Jiakai Wei,6 and Guangwu Hu1

1School of Computer, Shenzhen Institute of Information Technology, Shenzhen 518172, China
2Guangdong Key Laboratory of Intelligent Information Processing, College of Electronics and Information Engineering, Shenzhen University, Shenzhen 518060, China
3School of Artificial Intelligence, Shenzhen Polytechnic, Shenzhen 518055, China
4Guangzhou Institute of Technology, Xidian University, Guangzhou 510555, China
5Shaanxi Key Laboratory of Information Communication Network and Security, Xi’an University of Posts & Telecommunications, Xi’an, Shaanxi 710121, China
6Department of Neonatology, Xi’an Children’s Hospital, Xi’an Jiaotong University, Xi’an 710003, China

Correspondence should be addressed to Zhiwei Sun; smeker@szpt.edu.cn

Received 13 September 2021; Accepted 22 October 2021; Published 23 November 2021

1. Introduction

As a recent healthcare tendency, personalized medicine [1] enables the patient to obtain early diagnoses, risk estimation, optimal treatments with low costs by using molecular and cellular analysis technologies, diagnosis results, genetic information, etc. Personalized medicine is usually implemented by the dynamic treatment regime technology [2, 3], which can provide various therapeutic methods according to the time-varying clinical states of the patient. This technology is particularly suitable for coping with complex chronic illnesses, such as diabetes, mental diseases, alcohol dependence, and human immunodeficiency virus infection, which have various stages.

Reinforcement learning [4], which is implemented by trial-and-error and interaction with the dynamic environment, is an important method for developing dynamic treatment regimes, industry automation, vehicular networks [5, 6], and other scenarios [7–12]. Meanwhile, with the developing technologies of internet of things and cloud computing, dynamic treatment regimes that are based on reinforcement learning are becoming increasingly attractive. For example, wearable devices are helpful for monitoring the patient’s health data, which include heart rates and blood sugar levels. Next, collected health data are stored on the cloud. Then, the reinforcement learning algorithm can be implemented on these health data for making treatment decisions.
Unfortunately, because of the patient’s limited computation ability, health data are usually outsourced to the cloud server for implementing the reinforcement learning algorithm. Because the cloud server may be untrusted, it is likely that health data will be illegally accessed, forged, tampered, or discarded in the process of transmission and computation. In addition, it may be harmful for personal privacy, economic interests, and even the security of human life. For example, as a billing service company, American Medical Collection Agency was intruded in 2019 [13]. This attack affects the health data of about 12 million patients. Besides, the parent firm of this company has filed for bankruptcy. Furthermore, unlike financial data or other types of human-generated data [14], health data are permanent biological data. They cannot be modified or wiped to avoid the damage, which is caused by health data disclosure.

In order to protect health data, we can encrypt health data by using a traditional encryption algorithm. Unfortunately, the reinforcement learning algorithm cannot be executed on the encrypted health data easily and flexibly. Homomorphic encryption [15] supports the operations on the ciphertext. Hence, the cloud server can run the reinforcement learning algorithm on the encrypted health data perfectly by using homomorphic encryption without leaking patient privacy. Finally, the encrypted computation result is returned to the patient. The computation result can be obtained by using the patient’s secret key.

In this paper, we endeavor to study the security of health data in the above realistic scenario and focus on the secure implementation of the asynchronous advantage actor-critic (A3C) reinforcement learning algorithm. Taking into account the privacy and computation of health data on the untrusted cloud servers, we adopt homomorphic encryption as the main encryption primitive to carry out our research. Eventually, we make the following three contributions:

1. Because the efficiency of Cheon et al.’s approximate homomorphic encryption scheme [16] is better than that of fully homomorphic encryption (FHE), we use it to design secure computation protocols, namely, homomorphic comparison protocol, homomorphic maximum protocol, homomorphic exponential protocol, and homomorphic division protocol. Based on these protocols, we first design the homomorphic reciprocal of square root protocol, which needs only one approximate computation.

2. Based on the proposed secure computation protocols, we design the secure A3C reinforcement learning algorithm for the first time. Then, we use it to implement a secure treatment decision-making algorithm.

3. Finally, we simulate the proposed secure computation protocols and algorithms on the personal computer’s virtual machine. Then, we demonstrate the efficiency of our secure computation algorithms according to the thorough analysis.

The layout of this paper is as follows. Section 2 analyzes related work about homomorphic encryption and secure computation of encrypted health data. Preliminaries are presented in Section 3. Section 4 shows related work about secure dynamic treatment regimes on health data. Building blocks are discussed in Section 5. Section 6 describes the proposed privacy-preserving A3C reinforcement learning algorithm and treatment decision-making algorithm. Performance results are shown and analyzed in Section 7. Finally, this paper is concluded in Section 8.

2. Related Work

In this section, we introduce related work about reinforcement learning, homomorphic encryption, and the computation of encrypted health data, which are described as follows.

Reinforcement learning can be mainly classified as value-based algorithms, policy-based algorithms, and actor-critic algorithms. Value-based algorithms usually compute the optimum cumulative reward and give a suggested policy. As a typical value-based algorithm, Q-learning is used for estimating the utility of the individual pair that consists of a state and an action. Q-learning has been applied for path planning [17, 18] in vehicular networks. Policy-based algorithms can evaluate the optimum policy directly. Williams [19] proposed a policy-based algorithm REINFORCE. Actor-critic algorithms combine the advantages of value-based algorithms and policy-based algorithms. The A3C reinforcement learning algorithm [20] is an actor-critic algorithm. It can work in discrete action spaces as well as continuous action spaces [21].

The concept of homomorphic encryption begins from privacy homomorphism [15]. According to the types of supported homomorphic operations, homomorphic encryption can be divided into partial homomorphic encryption (PHE), somewhat homomorphic encryption (SWHE), and FHE. PHE only supports homomorphic addition or homomorphic multiplication. SWHE is the basis of FHE. SWHE supports finite homomorphic addition and homomorphic multiplication. FHE supports arbitrary homomorphic addition and homomorphic multiplication.

In 2009, Gentry [22] designed the first FHE scheme, which is based on ideal lattices. Since then, homomorphic encryption has become a research hotspot. Next, in order to improve the efficiency of homomorphic operations, Gentry et al. [23] first constructed the FHE scheme, which is based on the approximate eigenvector method. In this scheme, the ciphertext noise increases linearly after each homomorphic multiplication. Although homomorphic multiplication of this scheme is efficient, it does not support the technique of single instruction multiple data (SIMD) [24]. Then, based on the learning with errors over rings (RLWE) [25] assumption and relinearization technique [24], Brakerski et al. [24] designed a FHE scheme, which supports the SIMD technique. However, this scheme does not support approximate homomorphic operations. Hence, based on Brakerski et al.'s scheme [24], Cheon et al. [16] proposed an improved homomorphic encryption scheme.

In terms of the computation of encrypted health data by using homomorphic encryption, there exist following several schemes. Khedr and Gulak [26] first proposed an optimized
homomorphic encryption scheme, which is based on Gen-
try’s scheme [23]. Then, the proposed scheme is applied
for secure medical computations, which include comparison,
Pearson goodness-of-fit test, and logistic regression. Sun
et al. [27] implemented secure average heart rate, long QT
syndrome detection, and chi-square tests by using Dowlin
et al.’s FHE scheme [28]. Based on Boneh et al.’s homomo-

Definition 1 (RLWE). The RLWE assumption is to dis-
tinguish two distributions, namely, \((a, a \cdot s + c) \in R_q \times R_q\)
and \((a, c) \in \text{Unif}(R_q \times R_q)\), where \(a \in R_q\) and \(s \in R_q^*\). \(e\) is an
error term, and \(\text{Unif}\) represents uniform random. Lyuba-
shesvky et al. [25] proved that the security of RLWE
assumption relies on ideal lattices.

3.3. Cheon et al.’s Homomorphic Encryption Scheme. In
this subsection, we introduce Cheon et al.’s approximate homo-

3. Preliminaries

In this section, we begin with basic notations and definition
of Cheon et al.’s approximate homomorphic encryption
scheme. Then, we give the introduction of the A3C
algorithm.

3.1. Basic Notations. Let \(|z| = (z - 1/2, z + 1/2)\), where \(z\) is a
real number. Let \(|z|_p = z - |z/p|, p \in \{-p/2, p/2\}\), where \(p\)
denotes an integer.

Let \(R = \mathbb{Z}/\langle \Phi_m(x) \rangle\) denote the ring modulo \(\Phi_m(x)\),
where \(\lambda\) is the security parameter, \(m\) is a positive integer,
and \(\Phi_m(x)\) is the \(m\)th cyclotomic polynomial. \(R_q = \mathbb{Z}/\langle \Phi_m(x) \rangle/\langle \Phi_m(x) \rangle\) represents the ring modulo \(q\) and \(\Phi_m(x)\),
where \(q\) is the prime modulus, \(q \geq 2\).

As for an integer \(h > 0\), the distribution \(\mathcal{H} \mathcal{W} (h)\) is
selected from \(\{0, \pm 1\}\) randomly with the Hamming weight
\(h\). As for a rational number \(\sigma > 0\), the distribution \(\mathcal{D}\sigma(x^2)\)
outputs a vector, which coefficients are selected from the dis-
crete Gaussian distribution with the variance \(\sigma^2\). As for a
rational number \(0 \leq \rho \leq 1\), the distribution \(\mathcal{L}\sigma(\rho)\) is chosen
from \(\{0, \pm 1\}\) randomly, where \(\rho/2\) is the probability that \(\pm 1\)
is selected and \(1 - \rho\) is the probability that 0 is selected.

3.2. Learning with Errors over Rings. In 2010, Lyubashevsky
et al. [25] first proposed the RLWE assumption, which is
described as follows.

(i) AHE.KeyGen(\(1^{\lambda}, p, L\)): given the security parameter
\(\lambda\), an integer \(p\), and a level \(L\), this algorithm first
sets \(q_i = p' \cdot q_0\), where \(q_0\) is a fixed integer, \(l = L, \ldots, 1\). It selects a power-of-two integer \(M = M(\lambda, q_0)\), an integer \(P = P(\lambda, q_0)\), and a rational number
\(\sigma = \sigma(\lambda, q_0)\). Next, it chooses a vector \(s\) from \(\mathcal{R} \mathcal{W} (h)\). The secret key \(sk\) is set as \((1, s)\). A ring element \(a\) is
sampled from \(R_{q_i}\). An error term \(e\) is sampled from \(\mathcal{D}\sigma(\sigma^2)\). The public key \(pk\) is set as \((b, a) \in R_{p \cdot q_i}\),
where \(b = -a \cdot s + e\) (mod \(q_i\)). Then, a ring element \(a'\) is
sampled from \(R_{p \cdot q_i}\). An error term \(e'\) is sampled
from \(\mathcal{D}\sigma(\sigma^2)\). The evaluation key \(evk\) is set as \((b', a') \in R_{p \cdot q_i}\),
where \(b' = -a' \cdot s + e'\) (mod \(p \cdot q_i\)).

(ii) AHE.Enc(pk, m): in order to encrypt a plaintext \(m\),
this algorithm samples an integer \(v\) from \(\mathcal{L}(0.5)\). In addition, it chooses two error terms \(e_0\) and \(e_1\) from \(\mathcal{D}\sigma(\sigma^2)\), \(m\) is encrypted as the cipher-
text \(c = v \cdot pk + (m + e_0, e_1) \bmod (q_i)\).

(iii) AHE.Dec(sk, c): in this algorithm, \(c = (b, a)\) is
decrypted as \(b + a \cdot s \bmod (q_i)\).

(iv) AHE.Add(\(c, c'\)): in this algorithm, input parameters
include two ciphertexts \(c' = ([c_0]_{q_i}, [c_1]_{q_i})\)
and \(c'' = ([c_0']_{q_i}, [c_1']_{q_i})\), which are under the same
secret key. Then, the additive ciphertext \(c_{add} = ([c_0]_{q_i} + [c_0']_{q_i}, [c_1]_{q_i} + [c_1']_{q_i})\).

(v) AHE.Mul(evk, \(c, c'\)): in this algorithm, input parameters
include evk, two ciphertexts \(c' = ([c_0']_{q_i}, [c_1']_{q_i})\)
and \(c'' = ([c_0]_{q_i}, [c_1]_{q_i})\), where \(c'\) and \(c''\) are under the same
secret key. Then, the ciphertext
c_{temp} = (c_0, c_1, c_2) = ([c_0' \cdot c_0],
[c_0' \cdot c_1 + c_1' \cdot c_0'], [c_1' \cdot c_0], [c_1']_{q_i}).\) The multiplicative
ciphertext \(c_{mul} = (c_0, c_1) + [P \cdot c_2 \cdot evk] \bmod (q_i)\).

(vi) AHE.ReScale\(_{-l}^l\)(\(c\)): for a ciphertext \(c \in R_{q_i}\) at
the level \(l\), the new ciphertext \(c' = ([q_i']^{l/q_i})c \bmod (q_i')\).
(vii) AHE.Ecd($z, \Delta$): as for a vector $z = (z_0, z_1, \cdots, z_{Nz/2-1}) \in \mathbb{C}^{Nz/2}$ and a scaled factor $\Delta > 0$, this algorithm outputs $z' = [\sigma^{-1}(\Delta \cdot z)] \in \mathbb{R}$, where $\sigma^{-1}$ is the inverse operation of a canonical embedding map $\sigma(\cdot)$.

(viii) AHE.Dcz($z'$, $\Delta$): as for $z' \in \mathbb{R}$, this algorithm outputs $z = \Delta^{-1} \cdot \sigma(m) \in \mathbb{C}^{Nz/2}$.

In Cheon et al.’s scheme, the decryption noise should be bounded by $8\sqrt{2} \cdot \sigma \cdot N + 6\sigma \cdot \sqrt{N} + 16\sigma \cdot \sqrt{h} \cdot N$ for the correctness of decryption. In addition, the noise of the rescaling ciphertext is at most $\sqrt{N/3} \cdot (3 + 8\sqrt{h})$. Furthermore, the noise of the multiplicative ciphertext should be less than $P^{-1} \cdot q_{1} \cdot 8\sigma \cdot N/\sqrt{3} + \sqrt{N/3} \cdot (3 + 8\sqrt{h})$. The details about the analysis of Cheon et al.’s scheme can be found in [16].

3.4. Asynchronous Advantage Actor-Critic Reinforcement Learning Algorithm. In 2016, Mnih et al. [20] proposed the asynchronous advantage actor-critic reinforcement learning algorithm, which is based on combining the value-based method and the policy-based method. One advantage of the A3C algorithm is that it can work in discrete action spaces as well as continuous action spaces. In addition, in order to improve the learning efficiency of the A3C algorithm, multiple asynchronous actor-learners, which can interact with the environment and acquire various independent exploration policies, are running in parallel. The details of the A3C algorithm are described as follows.

In the A3C algorithm, there is a policy function $\pi(a_t | s_t ; \theta)$ and a value function $V(s_t ; \theta_v)$, where $a_t$ denotes an action at the time step $t$, $s_t$ denotes a state at the time step $t$, and $\theta$ and $\theta_v$ are two parameters. In addition, $V(s_t ; \theta_v)$ and $\pi(a_t | s_t ; \theta)$ will be updated $t_{\max}$ times, where $t_{\max}$ denotes the maximum step. $V(s_t ; \theta_v)$ and $\pi(a_t | s_t ; \theta)$ are usually approximated by a single convolutional neural network. Specifically, $V(s_t ; \theta_v)$ is based on a linear layer. $\pi(a_t | s_t ; \theta)$ is relied on a softmax layer. Namely, $V(s_t ; \theta_v) = x(s_t ; \theta_v)$, where $x(s_t)$ is a function which is related to $s_t$. $\pi(a_t | s_t ; \theta) = \delta'(\sigma(a_t ; \theta) \sum_{j=0}^{t_{\max}} \sigma(a_t | s_t ; \theta)) / \sigma(a_t | s_t ; \theta)$, $a_t$ is an action at the time step $t$, and $f(a_t | s_t)$ is a function which is related to $a_t$ and $s_t$.

Furthermore, the A3C algorithm uses two loss functions, namely, policy loss function and value loss function, which are described as follows. On the one hand, the policy loss function

$$ f_{\pi}(\theta) = \ln \pi(a_t | s_t ; \theta) \cdot (R - V(s_t ; \theta_v)) + \beta \cdot H(\pi(s_t ; \theta)), $$

(1)

where $R$ is the reward and the parameter $k$ depends on the state. In addition, the upper bound of $k$ is $t_{\max} - r_{t+i}$ is the immediate reward. The discount factor $\gamma \in (0, 1]$ The entropy function $H(\pi(s_t ; \theta))$ can be set as $-\sum_{j=0}^{t_{\max}} \sigma(a_t | s_t ; \theta) \cdot \ln \pi(s_t ; \theta)$. The hyperparameter $\beta$ can adjust the intensity of the entropy regulation term. Then, we can conclude that

$$ f_{\pi}(\theta) = \left( f(a_t | s_t) - \sum_{j=0}^{t_{\max}} f(a_j | s_j) \right) \cdot \theta \cdot (R - x(s_t ; \theta_v)) + \beta \cdot \left( -\sum_{j=0}^{t_{\max}} f(a_j | s_j) \cdot f(a_t | s_t) - \sum_{j=0}^{t_{\max}} f(a_j | s_j) \right) \cdot \theta^2. $$

(2)

Hence, the differentiation of $f_{\pi}(\theta)$ with respect to $\theta$ is

$$ \frac{\partial f_{\pi}(\theta)}{\partial \theta} = \left( f(a_t | s_t) - \sum_{j=0}^{t_{\max}} f(a_j | s_j) \right) \cdot (R - x(s_t ; \theta_v)) + 2\beta \cdot \theta \cdot f(a_t | s_t) \cdot \left( f(a_t | s_t) - \sum_{j=0}^{t_{\max}} f(a_j | s_j) \right). $$

(3)

On the other hand, the value loss function

$$ f_v(\theta_v) = (R - V(s_t ; \theta_v))^2 = (R - x(s_t ; \theta_v))^2. $$

(4)

Hence, the differentiation of $f_v(\theta_v)$ with respect to $\theta_v$ is

$$ \frac{\partial f_v(\theta_v)}{\partial \theta_v} = 2(R - x(s_t ; \theta_v)) \left( \frac{\partial R}{\partial \theta_v} - x(s_t ; \theta_v) \right). $$

(5)

Based on the above two loss functions and corresponding differentiation, the A3C reinforcement learning algorithm is defined in Algorithm 1, which is described as follows. Algorithm 1 requires input parameters $\theta$, $\theta_v$, $\theta'$, $\theta''$, $T$, $t$, $t_{\max}$, $T_{\max}$, $t_0$, $\eta$, $W$, and $\sigma$, where the definition of these parameters are shown in Table 1. In order to implement Algorithm 1, we first set $T = 0$, $t = 1$. If $T < T_{\max}$ and $w \in [1, W]$, we implement the iteration, which is shown as follows. Global gradients $\theta$ and $\theta_v$ are set as 0. $\theta'$ and $\theta''$ are synchronized as $\theta$ and $\theta_v$, respectively. We set $t_0 = t$ and observe the system state $s_{t+1} \in S$, where $s$ is a state set, $s = (s_0, \cdots, s_{W-1})$, and $\phi$ is the number of states. Next, we repeat a subalgorithm until $t - t_0 \neq t_{\max}$. In this subalgorithm, the action $a$ is obtained by using $\pi(a \mid s_{t'}, \theta')$, where $s$ is an action set, $s = (s_0, \cdots, s_{W-1})$, and $\chi$ is the number of actions. We execute $a$, get the reward $R$, and observe the next state $s_{t+1}$, where $R$ is set as $\sum_{i=0}^{k+1} r_{t+i} + \gamma^k \cdot V(s_{t+k+1} ; \theta') = \sum_{i=0}^{k} \gamma^i \cdot r_{t+i} + \gamma^k \cdot x(s_{t+k+1} ; \theta'). \theta'$. In addition, we set $t = t + 1$. After the implementation of the above subalgorithm, we observe whether $t \% t_0$ equals to 0. If $s_{t'}$ is terminal, we set $R = 0$. If $s_{t'}$ is nonterminal $R = V(s_{t'} ; \theta'') = x(s_{t'} ; \theta'')$. Then, we repeat a subalgorithm from $i = t - 1$ to $i = t_0$. In this subalgorithm, $R$ is set as $R_{i} + \gamma \cdot R$, namely, $R = \sum_{i=0}^{k-1} \gamma^i \cdot r_{t+i} + \gamma^k \cdot x(s_{t+k} ; \theta') + \gamma \cdot R$. We compute

$$ \frac{\partial f_{\pi}(\theta')} {\partial \theta'} = \left( f(a_i | s_i ; \theta') \right) \cdot (R - x(s_{t} ; \theta_v)) + 2\beta \cdot \theta' \cdot f(a_i | s_i ; \theta') \left( f(a_i | s_i ; \theta') - \sum_{j=0}^{t_{\max}} f(a_j | s_j) \right) \cdot \theta'^2. $$

(6)
\[ A3C(\theta, \theta_v, \theta^v, t, t_{max}, T, t, t_{max}, \theta, \theta_v, W, \alpha): \]

**Input:** \( \theta, \theta_v, \theta^v, t, t_{max}, T, t, t_{max}, \theta_v, \theta, W, \alpha \)

**Output:** \( \theta, \theta_v \)

Set \( T = 0, t = 1 \).

While \( T < T_{max} \) do.

For \( \omega = 1 \) to \( W \) do.

Set \( d\theta = 0, d\theta_v = 0 \).

Synchronize \( \theta' = \theta, \theta_v' = \theta_v \).

Set \( t_0 = t \) and get \( \delta_t \).

Repeat.

Get \( \delta_t \), according to \( \pi(\delta_t | \delta_t ; \theta') \).

Execute \( \delta_t \), get \( R_t \) and observe \( \delta_{t+1} \).

\[ t = t + 1 \]

Until \( t - t_0 = t_{max} \)

If \( \delta_t \) is terminal, \( R = 0 \).

If \( \delta_t \) is non-terminal, \( R = x(\delta_t) \cdot \theta_v' \).

For \( i = t - 1 \) to \( t_0 \) do

\[ R = R_t + \gamma \cdot R \]

Compute \( \partial f_v(\theta')/\partial \theta_v = (f(\delta_t | \delta_t ; \theta_v) - \sum_{i=0}^{\max} f(\delta_t | \delta_t) + 2\beta \cdot \theta' f(\delta_t | \delta_t) - f(\delta_t | \delta_t) - \sum_{j=0}^{\max} f(\delta_t | \delta_t)) \]

\[ \partial R/\partial \theta_v \]

End for.

\[ \partial f_v(\theta^v) = 2(R - x(\delta_t) \cdot \theta_v^v) \cdot (\partial R/\partial \theta_v) \]

End for.

End while.

---

**Algorithm 1:** A3C reinforcement learning algorithm.

---

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \theta, \theta_v )</td>
<td>Shared parameter vectors in the global network</td>
</tr>
<tr>
<td>( \theta', \theta^v )</td>
<td>Thread-specific parameter vectors in the local network</td>
</tr>
<tr>
<td>( T )</td>
<td>Global counter</td>
</tr>
<tr>
<td>( t )</td>
<td>Local step counter</td>
</tr>
<tr>
<td>( t_{max}, T_{max} )</td>
<td>Upper bounds</td>
</tr>
<tr>
<td>( t_g )</td>
<td>An integer</td>
</tr>
<tr>
<td>( \eta )</td>
<td>Learning rate</td>
</tr>
<tr>
<td>( W )</td>
<td>Number of agents</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>Momentum</td>
</tr>
</tbody>
</table>

where \( \partial f_v(\theta^v)/\partial \theta_v \) is the differentiation of \( f_v(\theta^v) \) with respect to \( \theta_v \), and \( \partial R/\partial \theta_v \) is the differentiation of \( R \) with respect to \( \theta^v \). Finally, \( \theta \) and \( \theta_v \) can be updated by using equations

\[ \theta = \theta - \eta (d\theta) + \gamma (\theta - \eta (d\theta)) \]

and

\[ \theta_v = \theta_v - \eta (d\theta_v) + \gamma (\theta_v - \eta (d\theta_v)) \]

respectively, where \( g = \alpha \cdot g + (1 - \alpha)(d\theta) \) and \( g_v = \alpha \cdot g_v + (1 - \alpha)(d\theta_v) \).

---

**4. Secure Dynamic Treatment Regimes on Health Data**

**4.1. System Model.** As shown in Figure 1, the system model of secure dynamic treatment regimes on health data consists of four parts, namely, undiagnosed patient, key generation center, cloud servers, and historical data owners, which are described as follows:

(i) The undiagnosed patient’s current state is collected by using wearable devices, which integrate modules of physiological sensors, weak computation, and communication. Wearable devices include smart bracelet, smart glasses, sleep monitoring sensors, and smart watch. They can collect a variety of health data, such as body temperature, heart rate, blood sugar, and blood volume index. Then, these health...
data are transmitted to cloud servers for computation. Based on the returned computation result, the patient can obtain the diagnosis.

(ii) Key generation center is an indispensable and independent entity, which is trusted by the other entities in this system model. It is responsible for distributing and managing all the public keys and private keys of Cheon et al.’s homomorphic encryption scheme for wearable devices and undiagnosed patients via a secure channel.

(iii) Cloud servers have the powerful data storage space. Hence, they store and manage ciphertexts, which come from undiagnosed patients and wearable devices. Additionally, they can perform some computations on these ciphertexts.

(iv) Historical data owners have a sequence of medical data and their corresponding decision results. These encrypted data are transmitted and stored on cloud servers. Cloud servers can compute these ciphertexts for training the reinforcement learning model.

4.2. Attack Model. In this paper, we suppose that the entities in the system model are honest-but-curious. Namely, the entities strictly follow the designed protocols. But they are interested in acquiring medical data of other entities. We suppose that there is an adversary $A^*_1$ in the attack model.

The goal of $A^*_1$ is to guess the plaintexts of the challenge historical data owners’ ciphertexts or the challenge wearable devices’ ciphertexts.

In order to acquire the ciphertexts of historical data owners and wearable devices, middle ciphertext results during the execution of privacy-preserving A3C reinforcement learning algorithm and treatment decision-making algorithm (Section 6), $A^*_1$ eavesdrops on the communication links among the entities in the system model. However, these ciphertexts are based on Cheon et al.’s approximate homomorphic encryption scheme [16]. Hence, $A^*_1$ cannot decrypt these ciphertexts without knowing their secret keys. It can be guaranteed by using the semantic security of Cheon et al.’s scheme. In addition, the key generation center distributes key pairs to historical data owners and wearable devices in a secure way. Furthermore, due to the lack of private keys of these ciphertexts, $A^*_1$ cannot generate evaluation keys. Hence, $A^*_1$ cannot transform these ciphertexts into some domains that $A^*_1$ can decrypt. Besides, $A^*_1$ cannot get useful information by adding or multiplying a plaintext with these ciphertexts. In a conclusion, the proposed model is secure.

4.3. System Setup and Overview. Our secure model of dynamic treatment regimes consists of two phases, which are described as follows.

(i) Training dataset outsourcing and initialization: historical data owners initialize input parameters $\theta, \theta_v$,......
learning rate $\eta$, and discount factor $\gamma$. The state set $s = (s_0, \ldots, s_{t+1})$ and action set $a = (a_0, \ldots, a_{t-1})$ are encrypted as $c_s = (c_{s_0}, \ldots, c_{s_{t+1}})$ and $c_a = (c_{a_0}, \ldots, c_{a_{t-1}})$. Then, the cloud server transmits the ciphertexts $c_s$ to the device users. The cloud server uses $c_s$ to encrypt the converted result.

5.2. Homomorphic Comparison Protocol. In order to implement the privacy-preserving A3C reinforcement learning algorithm, we need to encrypt health data. Health data are usually rational numbers. However, most of the homomorphic encryption schemes only support homomorphic operations over integers. They cannot cope with rational numbers. Hence, in this paper, we take Cheon et al.’s encoding technique [16], which can encode a rational number. Then, the rational number can be converted to a ring element just like using the integer encoding technique. We can use Cheon et al.’s scheme [16] to encrypt the converted result.

5.3. Homomorphic Maximum Protocol. In order to compute the encrypted index of the largest plaintext, we design a homomorphic maximum protocol by using the above protocol and Sun et al.’s method [35]. We suppose that the user owns plaintexts $m_0$, $m_1$, $m_2$, $m_3$, and $m_4$. The user uses $c_0$ and $c_1$ to encrypt these plaintexts. The ciphertexts are $c_{0}$, $c_{1}$, respectively. The user owns the secret key $sk$. As shown in Algorithm 3, the cloud server computes $c_0 = \text{comp}(c_{\max}, c_1)$, where $c_{\max}$ is initialized as $c_0$. If the decryption result $b < t$, $c_{\max} = c_b$, $i = i + 1$. The cloud server continues to compare $c_{\max}$ and $c_1$ until $i > k - 1$. Finally, the user can obtain the index of the largest plaintext by decrypting $c_{\max}$. For example, there exist ciphertexts $c_2$, $c_3$, and $c_4$, whose plaintexts are 2, 3, and 4, respectively. We set $c_{\max} = c_3$. Next, we first compare $c_{\max}$ and $c_3$. $c_{\max}$ is updated as $c_3$. Then, we compare $c_{\max}$ and $c_4$. $c_{\max}$ is updated as $c_4$. After the decryption of $c_{\max}$, the user gets the maximum result 4.

5.4. Homomorphic Exponential Protocol. In this section, based on the Taylor series, we begin to describe the homomorphic exponential protocol. We suppose that the user owns the plaintext $m$. Next, it is encrypted as $c_m$ by using Cheon et al.’s homomorphic encryption scheme. The user has the secret key $sk$. Then, $c_m$ is stored on the cloud server. In the homomorphic exponential protocol (Algorithm 4), the cloud server first computes the ciphertext $c_{e^m} = 1 + c_m + c_m^2/2! + \cdots + c_m^n/n!$ without decryption, where $n$ denotes an integer. The precision of $e^m$ increases with the increasing of $n$. Then, $c_{e^m}$ is returned to the user. The user decrypts the ciphertext $c_{e^m}$ by using his secret key. For example, for $m = 1$, we can set $m = 4$, $n = 3$, and then $c_{e^3} = 1 + c_1 + c_1^2/2! + c_1^3/3!$, where $c_1$ and $c_2$ are ciphertexts of $e^2$ and 4, respectively. After the decryption of $c_{e^3}$, the user gets the exponential result $e^3$.

5.5. Homomorphic Division Protocol. In this section, we begin to describe the homomorphic division protocol. We suppose that the user owns plaintexts $m_0$, $m_1$, and $m_2$. Then, they are encrypted as $c_{m_0}$, $c_{m_1}$, and $c_{m_2}$ by using Cheon et al.’s
homomorphic encryption scheme. Only the user has the secret key. Then, $c_{\text{m}1}$, $c_{\text{m}2}$, and $c_{\text{m}3}$ are transmitted to the cloud server.

In order to output the ciphertext $c_{\text{div}}$ of the plaintext $m_2/(m_0 + m_1)$, we design the homomorphic division protocol (Algorithm 5), which is described as follows. The cloud server first computes the ciphertext $c_{\text{add}} = c_{\text{m}0} + c_{\text{m}1}$ without decryption, where the plaintext of $c_{\text{add}}$ is $m_0 + m_1$. Then, $c_{\text{add}}$ is returned to the user. The user gets the plaintext $m_0 + m_1$ by using his secret key. The user calculates $\text{rev} = 1/\text{add}$. $\text{rev}$ is encrypted as $c_{\text{rev}}$ by using Cheon et al.’s scheme. $c_{\text{rev}}$ is transmitted to the cloud server. The cloud server calculates the ciphertext $c_{\text{div}} = c_{\text{m}0} \cdot c_{\text{rev}}$. Finally, $c_{\text{div}}$ is returned to the user. After the decryption of $c_{\text{div}}$, the user gets the division result $\text{div} = m_2/(m_0 + m_1)$. For example, we set $m_0 = 6$, $m_1 = 1$, and $m_2 = 2$. Then, the cloud server calculates $c_{\text{add}} = c_{\text{m}0} + c_{\text{m}1} = c_3$, where the plaintext of $c_3$ is 3. $c_3$ is returned to the user. After the decryption of $c_3$, the user calculates $\text{rev} = 1/3 = 0.33$. The ciphertext $c_{\text{rev}}$ is sent to the cloud server. The cloud server calculates $c_{\text{div}} = c_{\text{m}0} \cdot c_{\text{rev}} = 0.33 \times 6 = c_{1.98}$, where the plaintext of $c_{1.98}$ is 1.98.

5.6. Homomorphic Reciprocal of Square Root Protocol. In this section, we begin to describe the homomorphic reciprocal of square root protocol. The traditional method is to compute the ciphertext of the approximate square root firstly. Then, it computes the approximate reciprocal of square root homomorphically. However, two approximate computations will affect the precision of the final result. Hence, based on Lomont’s fast inverse square root algorithm [36], we design a new homomorphic reciprocal of square root protocol (Algorithm 6), which only needs one approximate computation. In our protocol, we suppose that the user owns the floating number $m = (1 + m_0)2^{m_1-127}$, where $0 < m_0 < 1$ and $0 < m_1 < 255$. It is encrypted as $c_m$ by using Cheon et al.’s homomorphic encryption scheme. Only the user has the secret key. $c_m$ is stored on the cloud server. $c_m$ is first transmitted to the user. The user decrypts $c_m$ by his secret key.

The decryption result $m$ is converted to an integer $m' = m_1 \cdot 2^{23} + m_0 \cdot 2^{23}$. $m'$ is encrypted as $c_m'$ by using Cheon et al.’s scheme. Next, $c_m'$ is transmitted to the cloud server. The cloud server computes the intermediate ciphertext

$$c_{\text{temp}} = \frac{3}{2} \left( 127 - 0.045 \right) 2^{23} - 0.5 c_m', \quad (8)$$

where the plaintext of $c_{\text{temp}}$ is the floating number temp. Then, the cloud server sends $c_{\text{temp}}$ to the user. The user decrypts $c_{\text{temp}}$ to obtain temp = temp$_1 \cdot 2^{23} + \text{temp}_0 \cdot 2^{23}$, where $0 < \text{temp}_0 < 1$ and $0 < \text{temp}_1 < 255$. temp is converted to an integer $m'' = 1 + \text{temp}_0 \cdot 2^{\text{temp}_1-127}$. $m''$ is encrypted as $c_m''$. $c_{\text{temp}}'$ is transmitted to the cloud server. The cloud server computes the ciphertext

$$c_{1/\sqrt{m}}' = \frac{3}{2} c_{\text{temp}}' - \frac{1}{2} c_m' \cdot c_{\text{temp}}'. \quad (9)$$

Then, $c_{1/\sqrt{m}}'$ is returned to the user. The user gets the reciprocal of square root $1/\sqrt{m}$ by using his secret key. For example, we set $m = (1 + 0.25)^{124-127} = 0.156$. Then, $m$ is converted to $m'' = 62 \cdot 2^{23} + 0.125 \cdot 2^{23}$. The ciphertext $c_m'$ of $m''$ is sent to the cloud server. The cloud server computes

$$c_{\text{temp}}' = \frac{3}{2} \left( 127 - 0.045 \right) 2^{23} - 0.5 c_m'. \quad (10)$$

The user decrypts $c_{\text{temp}}'$ to obtain temp = $128 \cdot 2^{23} + 0.3075 \cdot 2^{23}$. temp is converted to temp$' = (1 + 0.3075) 2^{128-127} = 2.615$. The ciphertext $c_{\text{temp}}'$ of temp$'$ is sent to
the cloud server. The cloud server computes the ciphertext

\[
\text{c}_{\sqrt{0.156}} = \frac{3}{2} \text{c}_{\text{temp}} - \frac{1}{2} \text{c}_{m} \cdot t_{\text{temp}}^3.
\]  

The user decrypts \( \text{c}_{\sqrt{0.156}} \) to obtain \( 1/\sqrt{0.156} = 3/2 \cdot 2.615 - 1/2 \cdot 0.156 \cdot 2.615^3 \approx 2.528 \).

### 6. Privacy-Preserving Computation Algorithms

#### 6.1. Privacy-Preserving A3C Reinforcement Learning Algorithm

In this section, we begin to describe how to implement a privacy-preserving A3C reinforcement learning algorithm by using Cheon et al.’s approximate homomorphic encryption scheme [16]. As shown in Algorithm 7, we describe the privacy-preserving A3C reinforcement learning algorithm as follows. Algorithm 7 requires input parameters \( \theta \), \( \theta', \theta'' \), \( T \), \( t \), \( t_{\max} \), \( t_{g} \), \( \eta \), \( W \), \( \alpha \), \( c_{d} \), and \( c_{st} \). Set \( T = 0 \), \( t = 1 \). If \( T < t_{\max} \) and \( w \in [1, W] \), we implement the iteration, which is shown as follows. \( d_{\theta} \) and \( d_{\theta'} \) are set as 0. \( \theta' \) and \( \theta'' \) are synchronized as \( \theta' \) and \( \theta'' \) respectively. We set \( t_{0} = t \) and obtain the encrypted system state \( c_{s_j} \in c_{s} \). Next, we repeat a subalgorithm until \( t - t_{0} \neq t_{\max} \). In this subalgorithm, the encrypted action \( c_{th} \in c_{st} \) is obtained based on argmax(\( c_{s_j} \), \( c_{s_j} \), \( \cdots \), \( c_{s_{max}} \)), where \( c_{s_j} = \varphi(\text{c}_{\sqrt{0.156}}) \cdot \varphi' \), \( j = 0, 1, \ldots, t_{\max} \). We execute \( c_{st} \) and get the encrypted reward

\[
\text{c}_{\text{r}} = \sum_{i=1}^{t_{\max}} \varphi' \cdot t_{t+i} + \varphi' \cdot x(\text{c}_{\text{st}_{t+i}}) \cdot \theta_{v}.
\]  

We observe the next encrypted state \( c_{s_j} \in c_{s} \). In addition,
where $t = t + 1$. After the implementation of the above subalgorithm, we observe whether $c_{\delta_i}$ is terminal. Then, we repeat a subalgorithm from $i = t - 1$ to $i = t_0$. In this subalgorithm, if $c_{\delta_i}$ is terminal, $c_R$ is set as

$$c_R = \sum_{i=0}^{k-1} r^i \cdot r_{t+i} + y^k \cdot c_{\delta_i} \cdot \theta'_v. \quad (13)$$

The cloud server computes

$$c_{\delta_i}(v') = \left( f(c_{\delta_i}, c_{\delta_i}) - \sum_{j=0}^{\text{max}} f(c_{\delta_j}, c_{\delta_i}) \right) \cdot \left( y^k \cdot x(c_{\delta_i}) - x(c_{\delta_i}) \right). \quad (14)$$

The cloud server computes

$$c_{\delta_i}(v') = 2 \left( \sum_{i=0}^{k-1} r^i \cdot r_{t+i} + y^k \cdot x(c_{\delta_i}) \cdot \theta'_v - x(c_{\delta_i}) \cdot c_{\delta_i} \right) \cdot \left( y^k \cdot x(c_{\delta_i}) - x(c_{\delta_i}) \right). \quad (15)$$

If $c_{\delta_i}$ is nonterminal, $c_R$ is set as

$$c_R = \sum_{i=0}^{k-1} r^i \cdot r_{t+i} + y^k \cdot x(c_{\delta_i}) \cdot \theta'_v + y \cdot x(c_{\delta_i}) \cdot \theta_v. \quad (16)$$

The cloud server computes

$$c_{\delta_{ji}}(v') = \left( f(c_{\delta_i}, c_{\delta_i}) - \sum_{j=0}^{\text{max}} f(c_{\delta_j}, c_{\delta_i}) \right) \cdot \left( y^k \cdot x(c_{\delta_i}) - x(c_{\delta_i}) \right). \quad (17)$$

where $c_{\delta_{ji}}(v')$ is the ciphertext of $\partial f_{\delta_i}(\theta')/\partial \theta'$. The cloud server computes

$$c_{\delta_{ji}}(v') = 2 \left( \sum_{i=0}^{k-1} r^i \cdot r_{t+i} + y^k \cdot x(c_{\delta_i}) \cdot \theta'_v + y \cdot x(c_{\delta_i}) \right) \cdot \left( y^k \cdot x(c_{\delta_i}) - x(c_{\delta_i}) \right). \quad (18)$$

where $c_{\delta_{ji}}(v')$ is the ciphertext of $\partial f_{\delta_i}(\theta')/\partial \theta'$. Then, $c_{\delta_{ji}}$ is set as $c_{\delta_{ji}} + c_{\delta_{ji}}(v') \cdot \theta'_v$. The cloud server computes $c_{\delta_{ji}} = \alpha c_{\delta_{ji}} + (1 - \alpha)(c_{\delta_{ji}})^2$ and $c_{\delta_{ji}} = \alpha c_{\delta_{ji}} + (1 - \alpha)(c_{\delta_{ji}})^2$. The cloud server sends $c_{\delta_{ji}}$ to the user. The user decrypts $c_{\delta_{ji}}$ and $c_{\delta_{ji}}$ to obtain $\theta'$. In order to further reduce the depth of homomorphic multiplication for the calculation of $c_{\delta_{ji}}$ and $c_{\delta_{ji}}$, the cloud server sends $c_{\delta_{ji}}$ to the user. The user decrypts $c_{\delta_{ji}}$ and $c_{\delta_{ji}}$ to obtain $\theta'$. Finally, based on the above homomorphic reciprocity of square root protocol, $\theta$ and $\theta_v$ can be updated by using equations $\theta = \theta - \eta(c_{\delta_{ji}}/\sqrt{c_{\delta_{ji}} + \epsilon})$ and $\theta_v = \theta_v - \eta(c_{\delta_{ji}}/\sqrt{c_{\delta_{ji}} + \epsilon})$, respectively. After the execution of Algorithm 7, we can get the encrypted optimized parameters $c_{\delta_{ji}}$ and $c_{\delta_{ji}}$, which are returned to the user. The user decrypts $c_{\delta_{ji}}$ and $c_{\delta_{ji}}$ to obtain $\theta$ and $\theta_v$, which can be used to implement the secure decision-making algorithm.

In order to better understand Algorithm 7, we give an example, which is described as follows. In this example, as shown in Table 2, we set the initial values of related parameters. We suppose that $(c_{\delta_{ji}}, \cdots, c_{\delta_{ji}})$ are ciphertexts of $\delta = (\delta_{ji}, \cdots, \delta_{ji}) = (0.1, 0.1, 0.15, 0.3, 0.35)$, respectively. $(c_{\delta_{ji}}, \cdots, c_{\delta_{ji}})$ are ciphertexts of $\delta_{ji} = (\delta_{ji}, \cdots, \delta_{ji}) = (0.1, 0.1, 0.15, 0.3, 0.35)$, respectively. For the convenience of computation, we let $(c_{\delta_{ji}}) = c_{\delta_{ji}}, f(c_{\delta_{ji}}, c_{\delta_{ji}}) = c_{\delta_{ji}} c_{\delta_{ji}}, \pi(c_{\delta_{ji}} | \delta_{ji}) = d_{aji}(\delta_{ji}), i = 0, 1, 2, 3, 4, j = 0, 1, 2, 3, 4. The cloud server first computes $c_{\delta_{ji}} = d_{aji}(\delta_{ji}) \cdot \theta'$ and $c_{\delta_{ji}} = d_{aji}(\delta_{ji}) \cdot \theta'$. Then, the ciphertext of $c_{\delta_{ji}}$ is $e_{1,0}^{3.0} = 0.005$ and the ciphertext of $c_{\delta_{ji}}$ is $e_{1,0}^{3.0} = 0.005$. Based on the implementation of the protocol argmax$(c_{\delta_{ji}}, c_{\delta_{ji}}) \delta_{ji}$, $e_{1,0}^{3.0}$ is executed. The cloud server computes $c_{\delta_{ji}} = r_t + yx(c_{\delta_{ji}}) \pi_{\delta_{ji}}$, where

$$R_t = r_t + yx(S_i) \pi_{\delta_{ji}} = 0.5 + 0.6 \times 0.1 \times 0.5 = 0.505. \quad (19)$$

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta, \theta_v$</td>
<td>0.5</td>
</tr>
<tr>
<td>$\theta', \theta_v'$</td>
<td>0.5</td>
</tr>
<tr>
<td>$T_{\text{max}}, t_{\text{max}}$</td>
<td>1</td>
</tr>
<tr>
<td>$t_0$</td>
<td>1</td>
</tr>
<tr>
<td>$k$</td>
<td>1</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>0.6</td>
</tr>
<tr>
<td>$r_1, r_2$</td>
<td>0.5</td>
</tr>
<tr>
<td>$\beta$</td>
<td>0.1</td>
</tr>
<tr>
<td>$W$</td>
<td>1</td>
</tr>
<tr>
<td>$g, g_v$</td>
<td>0</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>0.5</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>0.01</td>
</tr>
<tr>
<td>$\eta$</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Set $t = 2$. Because $\delta_2$ is nonterminal, the cloud server computes $c_R = x(c_{\delta_2}) \cdot \theta'_v$, where

$$R = x(c_{\delta_2}) \cdot \theta'_v = 0.15 \times 0.5 = 0.075.$$ \hfill (20)

Then, the cloud server computes

$$c_R = r_2 + y \cdot x(c_{\delta_2}) \cdot \theta'_v + y \cdot x(c_{\delta_2}) \cdot \theta'_v,$$ \hfill (21)

where

$$R = r_2 + y \cdot x(c_{\delta_2}) \cdot \theta'_v + y \cdot x(c_{\delta_2}) \cdot \theta'_v = 0.5 + 0.6 \times 0.15 \times 0.5 = 0.59.$$ \hfill (22)

The cloud server computes

$$\frac{\partial f_{\pi}(\theta'_v)}{\partial \theta'_v} = \left(\frac{f(c_{\delta_2} | c_{\delta_2}) - \Sigma_{j=0}^2 f(c_{\delta_2} | c_{\delta_2})}{r_2 + y \cdot x(c_{\delta_2}) \cdot \theta'_v + y \cdot x(c_{\delta_2}) \cdot \theta'_v}\right) \cdot \left(\theta'_v + y \cdot x(c_{\delta_2}) \cdot \theta'_v - x(c_{\delta_2}) \cdot \theta'_v + 2\beta \cdot \theta' \cdot f(c_{\delta_2} | c_{\delta_2}) \cdot \left(f(c_{\delta_2} | c_{\delta_2}) = \Sigma_{j=0}^2 f(c_{\delta_2} | c_{\delta_2})\right)\right),$$ \hfill (23)

where

$$\frac{\partial f_{\pi}(\theta'_v)}{\partial \theta'_v} = \left(\frac{f(c_{\delta_2} | c_{\delta_2}) - \Sigma_{j=0}^2 f(c_{\delta_2} | c_{\delta_2})}{r_2 + y \cdot x(c_{\delta_2}) \cdot \theta'_v + y \cdot x(c_{\delta_2}) \cdot \theta'_v + 2\beta \cdot \theta' \cdot f(c_{\delta_2} | c_{\delta_2}) \cdot \left(f(c_{\delta_2} | c_{\delta_2}) = \Sigma_{j=0}^2 f(c_{\delta_2} | c_{\delta_2})\right)\right) = (0.15 \times 0.15 - (0.1 \times (0.1 + 0.1 \times 0.1))) \cdot (0.5 + 0.6 \times 0.15 \times 0.5 + 0.6 \times 0.15 \times 0.5 - 0.15 \times 0.5) + 2 \times 0.1 \times 0.15 \times 0.15(0.15 \times 0.15 - 0.1 \times (0.1 + 0.1 \times 0.1)) = 0.0013.$$ \hfill (24)

The cloud server computes

$$c_{\delta_2, \theta'_v} = 2 \left(r_2 + y \cdot x(c_{\delta_2}) \cdot \theta'_v + y \cdot x(c_{\delta_2}) \cdot \theta'_v - x(c_{\delta_2}) \cdot \theta'_v\right) \cdot \left(\theta'_v + y \cdot x(c_{\delta_2}) \cdot \theta'_v + y \cdot x(c_{\delta_2}) \cdot \theta'_v\right) + y \cdot x(c_{\delta_2}) - x(c_{\delta_2}),$$ \hfill (25)

where

$$\frac{\partial f_{\pi}(\theta'_v)}{\partial \theta'_v} = 2 \left(r_2 + y \cdot x(c_{\delta_2}) \cdot \theta'_v + y \cdot x(c_{\delta_2}) \cdot \theta'_v - x(c_{\delta_2}) \cdot \theta'_v\right) \cdot \left(\theta'_v + y \cdot x(c_{\delta_2}) \cdot \theta'_v + y \cdot x(c_{\delta_2}) \cdot \theta'_v\right) = 2(0.5 + 0.6 \times 0.15 \times 0.5 + 0.6 \times 0.15 \times 0.5 - 0.15 \times 0.5)(0.6 \times 0.15 + 0.6 \times 0.15 - 0.15) = 0.0309.$$ \hfill (26)

Set $c_{\delta_2} = 0$, $c_{\delta_2} = 0$. The cloud server computes $c_{\delta_2} = c_{\delta_2} + c_{\delta_2} \cdot \theta'_v$, where

$$d\theta = d\theta + \frac{\partial f_{\pi}(\theta'_v)}{\partial \theta'_v} = 0.0013.$$ \hfill (27)

We compute $c_{\delta_2} = c_{\delta_2} + c_{\delta_2} \cdot \theta'_v$, where

$$d\theta' = d\theta + \frac{\partial f_{\pi}(\theta'_v)}{\partial \theta'_v} = 0.0309.$$ \hfill (28)

With the help of the user, the cloud server gets refreshed ciphertexts $c_{\delta_2}$ and $c_{\delta_2}$. The cloud server computes $c_{g} = ac_{g} + (1 - \alpha)(c_{\delta_2})^2$, where

$$g = ac_{g} + (1 - \alpha)(c_{\delta_2})^2 = (1 - 0.5) \times 0.00000845.$$ \hfill (29)

The cloud server computes $c_{g} = ac_{g} + (1 - \alpha)(c_{\delta_2})^2$, where

$$g = ac_{g} + (1 - \alpha)(c_{\delta_2})^2 = (1 - 0.5) \times 0.0000477405.$$ \hfill (30)

After the cloud server obtains ciphertexts $c'_g$ and $c'_g$, we set $c_{g} = c'_g$ and $c_{g} = c'_g$. The cloud server computes $c_{g} = \theta - \eta(c_{\delta_2} \cdot \sqrt{c_{g} + \epsilon})$, where

$$\theta = \theta - \eta \frac{d\theta}{\sqrt{g + \epsilon}} = 0.5 - 0.1 = 0.0013 \sqrt{0.0000000845 + 0.01} = 0.4987.$$ \hfill (31)

The cloud server computes $c_{\delta_2} = \theta - \eta(c_{\delta_2} \cdot \sqrt{c_{g} + \epsilon})$, where

$$\theta = \theta - \eta \frac{d\theta}{\sqrt{g_{\delta_2} + \epsilon}} = 0.5 - 0.1 = 0.0013 \sqrt{0.0000477405 + 0.01} = 0.4698.$$ \hfill (32)

The user can obtain refreshed $\theta = 0.4987$ and $\theta = 0.4698$ by using his secret key.

6.2. Secure Treatment Decision-Making Algorithm. In this subsection, based on the above privacy-preserving A3C reinforcement learning algorithm, secure treatment decision-making algorithm TDM$(\theta, \theta_{\pi}, c_{\pi}, c_{\pi}, c_{\phi})$ is implemented in Algorithm 8, which is described as follows. In this algorithm, input parameters include $\theta$, $\theta_{\pi}$, and the undiagnosed patient’s encrypted current state $c_{\pi}$, $c_{\phi}$, and $c_{\phi}$. Set the index $col = 0$. The ciphertext $c_{\phi}$ is initiated, where $i = 0, \cdots, \chi - 1$ and $j = 0, \cdots, \phi - 1$. Firstly, $c_{\phi}$'s element $c_{\phi}$ is compared with $c_{\phi}$ by using the above homomorphic comparison protocol.
Algorithm 8: Secure treatment decision-making algorithm.

Table 3: The distribution of the encrypted probability.

<table>
<thead>
<tr>
<th>Encrypted probability</th>
<th>Encrypted actions</th>
<th>Encrypted states</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$c_{d_1}$</td>
<td>$c_{d_2}$</td>
</tr>
<tr>
<td>$c_{s_k}$</td>
<td>$c_{0,0}$</td>
<td>$c_{1,0}$</td>
</tr>
<tr>
<td>$c_{s_1}$</td>
<td>$c_{0,1}$</td>
<td>$c_{1,1}$</td>
</tr>
<tr>
<td>$c_{s_2}$</td>
<td>$c_{0,2}$</td>
<td>$c_{1,2}$</td>
</tr>
<tr>
<td>$c_{s_3}$</td>
<td>$c_{0,3}$</td>
<td>$c_{1,3}$</td>
</tr>
</tbody>
</table>

Figure 2: The efficiency of our homomorphic comparison protocol.

Figure 3: The efficiency of our homomorphic maximum protocol $(k = 5)$.

Figure 4: The efficiency of our homomorphic maximum protocol $(k = 6)$.

comp($c_{s_j}, c_{s_i}$). $c_b$ is the encrypted comparison result. After the decryption of $c_b$, if the comparison result $b = t$, it means that $s_j = x$, set col = $j$, where $j = 0, \cdots, \varphi - 1$. Next, the cloud server computes the value function's encrypted value $c_v = V(c_{s_{col}}; \theta) = x(c_{s_j}) \cdot \theta_v$, where the plaintext of $c_v$ is $V(s_{col}; \theta)$. Set the ciphertext $c_{hor} = c_b$. The cloud server computes the policy function's encrypted value

$$c_{\pi,j} = \pi(c_{d_i} | s_{col}) = \frac{e^{f(c_{d_i} | s_{col})}}{\sum_{j=0}^{\varphi - 1} e^{f(c_{d_j} | s_{col})}}$$

where the plaintext of $c_{\pi,j}$ is $\pi(s_{col} | s_{col})$. The cloud server computes homomorphic multiplication between $c_v$ and $c_{\pi,j}$, namely, $c_{i, col} = c_v \cdot c_{\pi,i}$, where $c_{i, col}$ is the

230

Running time (ms)

Log q

oc = 1

oc = 2

oc = 3

oc = 4

690

460

230

300

2000

3000

4000

5000

6000

7000

400

500

600

700

Log q

oc = 1

oc = 3

oc = 2

oc = 4

6000

1600

2400
respectively. For the convenience of computation, we let \( x_{ci} = 0 \). Finally, the cloud server computes the ciphertext of \( S_{\text{om}} = S_{\text{om min}} \). Then, \( c_{dr} \) will be transmitted to the undiagnosed patient. In order to obtain the treatment decision \( dr \), \( c_{dr} \) can be decrypted by using his own secret key.

In order to better understand Algorithm 8, we give an example, which is described as follows. In this example, we set \( \chi = 4 \), \( \varphi = 4 \), \( \theta = 0.5 \), \( \theta_c = 0.5 \), and \( t_{\text{max}} = 3 \). We suppose that \( (c_{\delta_0}, \cdots, c_{\delta_{\chi-1}}) \) are ciphertexts of \((0,1,0,2,0,3,0,4)\), respectively. \( (c_{\delta_{be_{col}}}, \cdots, c_{\delta_{be_{col}}}) \) are ciphertexts of \((0,1,0,2,0,3,0,4)\), respectively. For the convenience of computation, we let \( x(c_{\delta_i}) = c_{\delta_i} \), \( f(c_{\delta_i} | c_{\delta_j}) = c_{\delta_i} \), \( \pi(\delta_i | \delta_j) = e^{\ell(\delta_i | \delta_j)} \), \( i = 0, 1, 2, 3 \), \( j = 0, 1, 2, 3 \). The calculation of \( c_{ij} = V(c_{\delta_i} ; \theta_c) \pi(c_{\delta_i} | c_{\delta_j}) \) is the key component for the execution of Algorithm 8, where the corresponding plaintext of \( c_{ij} \) is \( V(\delta_i ; \theta_c) \pi(\delta_j | \delta_j) \). In this example, the distribution of \( c_{ij} \) can be shown in Table 3.

If the patient requires a diagnostic service, the encrypted current state \( c_s \) is input for the implementation of Algorithm 8. Then, \( c_{\delta_{be_{col}}} \) is compared with \( c_s \) by the execution of the protocol \( c_b = \text{comp}(c_{\delta_{be_{col}}}, c_s) \), where \( be = 0 \). If the comparison result \( b = t \), set \( col = be \). If the comparison result \( b \neq t \), the next element \( c_{\delta_{be_{col}+1}} \) will be compared with \( c_s \) until \( be + 1 > 3 \). We suppose that \( x = \delta'_1 \), namely, \( col = 1 \). Hence, \( c_s = x(c_{\delta'_1}) \theta_c = x(c_{\delta'_1}) \theta_c = 0.5 \), where the plaintext of \( c_s \) is \( 0.2 \times 0.5 = 0.1 \). We compute ciphertexts \( c_{\delta_{be_{col}}}, c_{\delta_{be_{col}}}, c_{\delta_{be_{col}}}, \) and \( c_{\delta_{be_{col}}} \), which corresponding plaintexts are \( e^{0.01} / \text{temp} \), \( e^{0.02} / \text{temp} \), \( e^{0.03} / \text{temp} \), and \( e^{0.04} / \text{temp} \), respectively, where temp = \( e^{0.01} + e^{0.02} + e^{0.03} + e^{0.04} \). Then, we compute ciphertexts \( c_{\delta_0}, c_{\delta_1}, c_{\delta_2}, \) and \( c_{\delta_3}, \) respectively.

Figure 5: The efficiency of our homomorphic maximum protocol (\( n = 3 \)).

Figure 6: The efficiency of our homomorphic exponential protocol (\( n = 2 \)).

Figure 7: The efficiency of our homomorphic exponential protocol (\( n = 3 \)).

Figure 8: The efficiency of our homomorphic exponential protocol (\( n = 4 \)).
and $c_3$, which corresponding plaintexts are $0.1e^{0.01}/\text{temp}$, $0.1e^{0.02}/\text{temp}$, $0.1e^{0.03}/\text{temp}$, and $0.1e^{0.04}/\text{temp}$, respectively. Based on the execution of the protocol $\text{argmax}(c_{0,1}, c_{1,1}, c_{2,1}, c_{3,1})$, we can obtain the output ciphertext is $c_{3,1}$. The encrypted treatment decision is $c_A^3$, which corresponding plaintext is $A^3$.

7. Performance Results

In this section, based on Cheon et al.’s homomorphic encryption scheme, we analyze the efficiency of our secure computation protocols, secure A3C reinforcement learning algorithm, and secure treatment decision-making algorithm. We use the virtual machine to implement experiments without the GPU hardware platform. In our experimental environment, the operating system is macOS 10.14.6. Our personal computer has two Intel (R) Core (TM) i5 CPU processors, which runs at 2.3 GHz with 8.00 GB RAM. The operation system of a virtual machine is ubuntu 16.04. The virtual machine is allocated single Intel (R) Core (TM) i5 CPU processor with 1.0 GB RAM. In order to implement high-level numeric algorithms, we choose the NTL library. We use the GCC platform to compile our C++ codes. We adopt the UC Irvine Machine Learning Repository (http://archive.ics.uci.edu/ml/index.php) for implementing the experiments. For convenience, we set $\log q$ ranging from 400 to 700, the scaling factor $\log p = 30$.

Figure 2 shows the efficiency of our homomorphic comparison protocol, where the number of comparison $oc$ ranges from 1 to 4. As shown in Figure 2, the running time of our homomorphic comparison protocol increases significantly with the increasing of $oc$. Figures 3–5 show the efficiency of our homomorphic maximum protocol, where the number
of maximum om ranges from 1 to 4 and the number of plaintexts k ranges from 5 to 7. It can be easily observed that the running time of our homomorphic maximum protocol increases significantly with the increasing of k and om. Figures 6–8 show the efficiency of our homomorphic exponential protocol, where the number of exponential operation oe ranges from 1 to 4 and the integer n ranges from 2 to 4. We can observe that the running time of our homomorphic exponential protocol increases rapidly with the increasing of oe and n.

Then, Figure 9 shows the efficiency of our homomorphic division protocol, where the number of division od ranges from 1 to 4. We can observe the changing trend of the running time of our homomorphic division protocol. This protocol has an obvious growth of running time with the increasing of od and log q. Figure 10 shows the efficiency of our homomorphic reciprocal of square root protocol, where os ranges from 1 to 4; os denotes the number of operations of reciprocal of square root. With the increasing of os and log q, more running time is needed for implementing our homomorphic reciprocal of square root protocol. It can be observed that its running time is longer than the above homomorphic comparison, maximum, exponential, and division protocols. Figure 11 shows the efficiency of our secure A3C reinforcement learning algorithm, where ot ranges from 1 to 4; ot denotes the number of operations of A3C training algorithm. With the increasing of ot and log q, our A3C reinforcement learning algorithm requires more running time. This algorithm is responsible for training the parameters θ and θr. Hence, this algorithm is complicated. We can observe too much running time is needed for this algorithm, which can demonstrate the above viewpoint. Figure 12 shows the efficiency of our secure treatment decision-making algorithm, where dm ranges from 1 to 4; dm denotes the number of operations of treatment decision-making algorithm. The running time of this algorithm grows with the increasing of dm. This algorithm uses the optimized θ and θr. Hence, this algorithm is less complicated than the secure A3C algorithm. The running time of this algorithm is shorter than the secure A3C algorithm, which can verify the above viewpoint. In conclusion, the above efficiency analysis shows the feasibility of our secure computation protocols and algorithms.

### 8. Conclusion

Reinforcement learning is helpful for implementing dynamic treatment regimes on health data. However, private health data may be illegally leaked, falsified, or deleted in the execution of the reinforcement learning algorithm. Hence, we study secure dynamic treatment regimes on health data. In this paper, we have designed homomorphic comparison protocol, homomorphic maximum protocol, homomorphic exponential protocol, homomorphic division protocol, and homomorphic reciprocal of square root protocol. Based on these secure computation protocols, we have proposed a privacy-preserving A3C reinforcement learning algorithm for the first time. Then, it is used for implementing the secure treatment decision-making algorithm. Finally, we simulate the proposed secure computation protocols and algorithms. Simulation results show that our secure computation protocols and algorithms are feasible.

In the future research, we will use homomorphic encryption to implement other machine learning algorithms, such as distributed learning [37] and federated reinforcement learning [38], which can successfully dominate multiple real devices that have the same type and slightly different dynamics. In addition, we plan to evaluate the performance of the secure A3C algorithm in other real-world scenarios, for example, vehicular ad hoc network.

### Data Availability

The data of secure computation protocols and algorithms used to support the findings of this study are available from the corresponding author upon request.

### Conflicts of Interest

The authors declare that they have no conflicts of interest.

### Acknowledgments

This work was supported by the Science and Technology Innovation Projects of Shenzhen (JCYJ20190809152003992), Shenzhen Science and Technology Program (JCYJ20210324100813034), the Guangdong Basic and Applied Basic Research Foundation (2020A1515110496), and the College-Enterprise Collaboration Project of Shenzhen Institute of Information Technology (11400-2021-010201-010199).

### References


