Hindawi Publishing Corporation
EURASIP Journal on Information Security
Volume 2007, Article ID 37343, 11 pages
doi:10.1155/2007/37343

Research Article

Oblivious Neural Network Computing via Homomorphic Encryption

C. Orlandi,[1] A. Piva,[1] and M. Barni[2]

[1] Department of Electronics and Telecommunications, University of Florence, Via S.Marta 3, 50139 Firenze, Italy
[2] Department of Information Engineering, University of Siena, Via Roma 56, , 53100 Siena, Italy

Correspondence should be addressed to C. Orlandi, orlandi@lci.det.unifi.it

Received 27 March 2007; Accepted 1 June 2007

Recommended by Stefan Katzenbeisser

The problem of secure data processing by means of a neural network (NN) is addressed. Secure processing refers to the possibility that the NN owner does not get any knowledge about the processed data since they are provided to him in encrypted format. At the same time, the NN itself is protected, given that its owner may not be willing to disclose the knowledge embedded within it. The considered level of protection ensures that the data provided to the network and the network weights and activation functions are kept secret. Particular attention is given to prevent any disclosure of information that could bring a malevolent user to get access to the NN secrets by properly inputting fake data to any point of the proposed protocol. With respect to previous works in this field, the interaction between the user and the NN owner is kept to a minimum with no resort to multiparty computation protocols.

1. INTRODUCTION

Recent advances in signal and information processing together with the possibility of exchanging and transmitting data through flexible and ubiquitous transmission media such as Internet and wireless networks have opened the way towards a new kind of services whereby a provider sells its ability to process and interpret data remotely, for example, through a web service. Examples in this sense include interpretation of medical data for remote diagnosis, access to remote databases, processing of personal data, processing of multimedia documents. In addition to technological developments in artificial intelligence, multimedia processing and data interpretation, and to an easy and cheap access to the communication channel, the above services call for the adoption of security measures that ensure that the information provided by the users and the knowledge made available by the service providers are adequately protected.

Most of the currently available solutions for *secure manipulation of signals* apply some cryptographic primitives on top of the signal processing modules, so to prevent the leakage of critical information. In most cases, however, it is assumed that the involved parties trust each other, and thus the cryptographic layer is used only to protect the data against third parties. In the new application scenarios outlined above, however, this is only rarely the case, since the data owner usually does not trust the processing devices, or those actors required to manipulate the data. It is clear that the availability of signal processing algorithms that work directly on the encrypted data, would represent a powerful solution to the security problems described above.

A fundamental brick of modern artificial intelligence theory is represented by neural networks (NNs), which thanks to their approximation and generalization capabilities [1] are a universal tool enabling a great variety of applications. For this reason, in this paper we introduce a protocol whereby a user may ask a service provider to run a neural network on an input provided in encrypted format. The twofold goal is on one side to ensure that the data provided by the user, representing the input of the neural network, are adequately protected, on the other side to protect the knowledge (expertise) of the service provider embedded within the NN. it is worth pointing out that the scope of our protocol is not to preserve user anonymity. Specifically, the latter goal is achieved by protecting the weights of the network arcs, together with the parameters defining the neuron activation functions. The proposed protocol relies on homomorphic encryption principles (first introduced in [2]) whereby a few elementary operations can be performed directly in the encrypted domain. For those tasks that cannot be handled by means of homomorphic encryption, a limited amount of interaction between the NN owner and the user is introduced; however, in contrast to previous works in the general area of privacy preserving data mining [3], the interaction is kept to a minimum and no resort to sophisticated multiparty computation protocols [4, 5] is made. Great attention is paid to avoid any unnecessary disclosure of information, so that at the end of the protocol the user only knows the final NN output, whereas all internal computations are kept secret. In this way, the possibility for a malevolent user to provide a set of fake inputs properly selected to disclose the network secrets is prevented. A solution is also sketched that permits to obfuscate the network topology, however, a deeper investigation in this direction is left for future research.

The rest of this paper is organized as follows. In Section 2, the prior art in the general field of privacy preserving and oblivious computing is reviewed, and the peculiarities of our novel protocol are discussed. In Section 3 the cryptographic primitives our scheme relies on are presented. The details of the protocol we propose for oblivious NN computing are described in Section 4, where a single perceptron is studied, and in Section 5, where the whole multilayer feedforward network is analyzed. Section 6 is devoted to the discussion raised by the necessity of approximating real numbers by integer values (given that the adopted cryptosystem works only with integer values while NN computations are usually carried out by considering real numbers). Section 7 is devoted to the experimental results obtained developing a distributed application that runs the protocol. Some concluding remarks are given in Section 8.

2. PRIOR ART

In modern society great amount of data are collected and stored by different entities. Some of these entities may take an advantage cooperating with each other. For example, two medical institutions may want to perform a joint research on their data; another example is a patient that needs a diagnosis from a medical institute that has the knowledge needed to perform the diagnosis. Of course those entities want to get the maximum advantage from the cooperation, but they cannot (or do not want to) let the other party know the data they own. Usually they cannot disclose personal data due to privacy related law, and at the same time they like to keep their knowledge for business reasons.

A trivial solution to protect the data owned by the participants to the computation consists in resorting to a trusted third party (TTP) that actually carries out the computation on the inputs received by the two parties, and sends to them the corresponding output. A privacy preserving protocol allows to achieve the same goal without the participation of a TTP, in such a way that each player can only learn from the protocol execution the same information he/she could get by his/her own inputs and the output received by the TTP.

In 2000 two different papers proposed the notions of privacy preserving data mining, meaning the possibility to perform data analysis on a distributed database, under some privacy constraints. Lindell and Pinkas [6] presented a way to securely and efficiently compute a decision tree using cryptographic protocols; at the same time, Agrawal and Srikant [7] presented another solution to the same problem using data randomization, that is by adding noise to customer's data.

After the publication of these papers, the interest in privacy preserving cooperative computation has grown up. In particular several techniques from machine learning were converted to the multiparty scenario where several parties contribute to some kind of computation while preserving the security of the data provided by each of them. Solutions for the following algorithms were proposed: decision trees [6], neural networks [8], SVM [9], naive bayes classifiers [10], belief networks [11, 12], clustering [13]. In all these works, we can identify two major scenarios: in the first one Alice and Bob share a dataset and want to extract knowledge from it without revealing their own data (privacy preserving data mining). In the other scenario, the one considered in this paper, Alice owns her private data x, while Bob owns an evaluation function C (in most cases C is a classifier); Alice would like to have her data processed by Bob, but she does not want that Bob learns either her input or the output of the computation. At the same time Bob does not want to reveal the exact form of C, representing his knowledge, since, for instance, he sells a classification service through the web (as in the remote medical diagnosis example). This second scenario is usually referred to as oblivious computing.

Cooperative privacy preserving computing is closely related to secure multiparty computation (SMC), that is a scenario where Alice owns x, Bob owns y, and they want to compute a public function $f(\cdot)$ of their inputs without revealing them to each other. At the end of the protocol, Alice and Bob will learn nothing except $f(x, y)$. The roots of SMC lie in a work by Yao [14] proposing a solution to the millionaire problem, in which two millionaires want to find out which of them is richer without revealing the amount of their wealth. Later on Yao [15] presented a constant-round protocol for privately computing any probabilistic polynomial-time function. The main idea underling this protocol is to express the function f as a circuit of logical gates, and then perform a secure computation for each gate. It is clear that this general solution is unfeasible for situations where the parties own huge quantities of data or the functions to be evaluated are complex.

After these early papers extensively relying on SMC, more efficient primitives for privacy preserving computing were developed, based on homomorphic encryption schemes [16], which permit to carry out a limited set of elementary operations like additions or multiplications in the encrypted domain. In this way, a typical scheme for privacy preserving computing consists in a first phase where each party performs the part of the computation that he can do by himself (possibly by relying on a suitable homomorphic cryptosystem). Then the interactive part of the protocol starts, with protocol designers trying to perform as much as they can in an efficient way. At the end, the operations for which an efficient protocol is not known (like division, maximum finding, etc.) are carried out by resorting to the general solution by Yao.

Previous works on privacy preserving NN computing are limited to the systems presented in [8, 17]. However, first study resort extensively to SMC for the computation of the nonlinear activation functions implemented in the neurons, and hence is rather cumbersome. On the other hand, the protocol proposed in [17] may leak some information at the intermediate states of the computation, in fact the output of all the intermediate neurons is made available to the data owner, hence making it rather easy for a malevolent user to disclose the NN weights by feeding each neuron with properly chosen inputs. This is not the case with our new protocol which conceals all the intermediate NN computations and does not resort to SMC for the evaluation of the activation functions. In a nutshell, the owner of the NN (say Bob) performs all the linear computations in the encrypted domain and delegates the user (say Alice) to compute the nonlinear functions (threshold, sigmoid, etc.). Before doing so, however, Bob obfuscates the input of the activation functions so that Alice does not learn anything about what she is computing.

When designing an SMC protocol it is necessary to take into account the possible behavior of the participants to the protocol. Cryptographic design usually considers two possible behaviors: a participant is defined *semihonest* or *passive* if he follows the protocol correctly, but tries to learn additional information by analyzing the messages exchanged during the protocol execution; he is defined *malicious* or *active* if he arbitrarily deviates from the protocol specifications. In this work, like most of the protocols mentioned above, the semi-honest model is adopted. Let us note, however, that a protocol secure for semi-honest users can always be transformed into a protocol secure against malicious participants by requiring each party to use zero-knowledge protocols to grant that they are correctly following the specifications of the scheme.

3. CRYPTOGRAPHIC PRIMITIVES

In this section the cryptographic primitives used to build the proposed protocol are described.

3.1. Homomorphic and probabilistic encryption

To implement our protocol we need an efficient homomorphic and probabilistic, public key, encryption scheme.

Given a set of possible plain texts M, a set of cipher texts C, and a set of key pairs $K = \mathrm{PK} \times \mathrm{SK}$ (public keys and secret keys), a public key encryption scheme is a couple of functions $E_{\mathrm{pk}} : M \to C$, $D_{\mathrm{sk}} : C \to M$ such that $D_{\mathrm{sk}}(E_{\mathrm{pk}}(m)) = m$ (where $m \in M$) and such that, given a cipher text $c \in C$, it is computationally unfeasible to determine m such that $E_{\mathrm{pk}}(m) = c$, without knowing the secret key sk.

To perform linear computation (i.e., scalar product), we need an encryption scheme that satisfies the additive homomorphic property according to which, given two plaintexts m_1 and m_2 and a constant value a, the following equalities are satisfied:

$$D_{\mathrm{sk}}(E_{\mathrm{pk}}(m_1) \cdot E_{\mathrm{pk}}(m_2)) = m_1 + m_2, \quad D_{\mathrm{sk}}(E_{\mathrm{pk}}(m_1)^a) = am_1. \tag{1}$$

Another feature that we need is that the encryption scheme does not encrypt two equal plain texts into the same cipher text, since we have to encrypt a lot of 0s and 1s, given that the output of the thresholding and sigmoid activation functions is likely to be zero or one in most of the cases. For this purpose, we can define a scheme where the encryption function E_{pk} is a function of both the secret message x and a random parameter r such that if $r_1 \neq r_2$ we have $E_{\mathrm{pk}}(x, r_1) \neq E_{\mathrm{pk}}(x, r_2)$ for every secret message x. Let $c_1 = E_{\mathrm{pk}}(x, r_1)$ and $c_2 = E_{\mathrm{pk}}(x, r_2)$, for a correct behavior we also need that $D_{\mathrm{sk}}(c_1) = D_{\mathrm{sk}}(c_2) = x$, that is, the decryption phase does not depend on the random parameter r. We will refer to a scheme that satisfies the above property as a probabilistic scheme. This idea was first introduced in [18]. Luckily, homomorphic and probabilistic encryption schemes do exist. Specifically, in our implementation we adopted the homomorphic and probabilistic scheme presented by Paillier in [16], and later modified by Damgård and Jurik in [19].

3.2. Paillier cryptosystem

The cryptosystem described in [16], usually referred to as Paillier cryptosystem, is based on the problem to decide whether a number is an nth residue modulo n^2. This problem is believed to be computationally hard in the cryptography community, and is related to the hardness to factorize n, if n is the product of two large primes.

Let us now explain what an nth residue is and how it can be used to encrypt data. The notation we use is the classic one, with $n = pq$ indicating the product of two large primes, $\mathbb{Z}_n$ the set of the integer numbers modulo n, and $\mathbb{Z}_n^*$ the set of invertible elements modulo n, that is, all the integer numbers that are relatively prime with n. As usual, the cardinality of the latter set is indicated by $|\mathbb{Z}_n^*|$ and it is equal to the Euler's totient function $\phi(n)$.

Definition 1. $z \in Z_{n^2}^*$ is said to be a nth residue modulo n^2 if there exists a number $y \in Z_{n^2}^*$ such that $z = y^n \mod n^2$.

Conjecture 1. *The problem of deciding nth residuosity, that is, distinguishing nth residues from non nth residues is computationally hard.*

Paillier cryptosystem works on the following facts from number theory.

(1) The application

$$\varepsilon_g : \mathbb{Z}_n \times \mathbb{Z}_n^* \longrightarrow \mathbb{Z}_{n^2}^*, \quad m, y \longrightarrow g^m y^n \mod n^2 \tag{2}$$

with $g \in \mathbb{Z}_{n^2}^*$ an element with order multiple of n is a bijection.

(2) We define the class of $c \in \mathbb{Z}_{n^2}^*$ as the unique $m \in \mathbb{Z}_n$ for which $y \in \mathbb{Z}_n^*$ exists such that $c = g^m y^n \mod n^2$.

This is the ciphering function, where (g, n) represents the public key, m the plaintext, and c the ciphertext. Note that y can be randomly selected to have different values of c that belong to the same class. This ensures the probabilistic nature of the Paillier cryptosystem.

Let us describe now the deciphering phase, that is, how we can decide the class of c from the knowledge of the factorization of n,

(1) It is known that $|\mathbb{Z}_n^*| = \phi(n) = (p-1)(q-1)$ and $|\mathbb{Z}_{n^2}^*| = \phi(n^2) = n\phi(n)$.
Define $\lambda(n) = \mathrm{lcm}(p-1, q-1)$ (least common multiple).

(2) This leads to, for all $x \in \mathbb{Z}_{n^2}^*$,

$$\begin{aligned} &\text{(i)} \quad x^{\lambda(n)} = 1 \mod n, \\ &\text{(ii)} \quad x^{n\lambda(n)} = 1 \mod n^2. \end{aligned} \tag{3}$$

(3) From (ii) $(x^{\lambda(n)})^n = 1 \mod n^2$, so $x^{\lambda(n)}$ is an nth root of unity, and from (i) we learn that we can write it as $1+an$ for some $a \in \mathbb{Z}_n$. So $g^{\lambda(n)}$ can be written as $1+an \mod n^2$.

(4) Note that for every element of the form $1+an$ it is true that $(1+an)^b \mod n^2 = (1+abn) \mod n^2$. So $(g^{\lambda(n)})^m = (1+amn)$.

(5) Consider $c^{\lambda(n)} = g^{m\lambda(n)} y^{n\lambda(n)}$: we have that $g^{m\lambda(n)} = 1 + amn \mod n^2$ from (4) and $y^{n\lambda(n)} = 1 \mod n^2$ from (ii). We obtain that $c^{\lambda(n)} = 1 + amn \mod n^2$.

(6) So we can compute $c^{\lambda(n)} = 1 + amn \mod n^2$ and $g^{\lambda(n)} = 1 + an \mod n^2$.

(7) With the function $L(x) = (x-1)/n$, by computing $L(c^\lambda \mod n^2)$ and $L(c^\lambda \mod n^2)$ we can simply recover am, a, and obtain $am \cdot a^{-1} = m \mod n^2$.

Note that this is only an effort to make Paillier cryptosystem understandable using simple facts. For a complete treatment we refer to the original paper [16] or to [20].

At the end, the encryption and the decryption procedures are the following

Setup

Select p, q big primes. $\lambda = \mathrm{lcm}(p-1, q-1)$ is the private key. Let $n = pq$ and g in $\mathbb{Z}_{n^2}^*$ an element of order αn for some $\alpha \neq 0$. (n, g) is the public key.

Encryption

Let $m < n$ be the plaintext and $r < n$ a random value. The encryption c of m is

$$c = E_{\mathrm{pk}}(m) = g^m r^n \mod n^2. \tag{4}$$

Decryption

Let $c < n^2$ be the ciphertext. The plaintext m hidden in c is

$$m = D_{\mathrm{sk}}(c) = \frac{L(c^\lambda \mod n^2)}{L(g^\lambda \mod n^2)} \mod n, \tag{5}$$

where $L(x) = (x-1)/n$.

3.3. Generalized Paillier cryptosystem

In [19] the authors present a generalized and simplified version of the Paillier cryptosystem. This version is based on the complexity to decide the nth residuosity modulo n^{s+1}, and includes the original Paillier cryptosystem as a special case when $s = 1$.

The way it works is almost the same of the original version except for

(i) The domain where one can pick up the plaintext is $\mathbb{Z}_{n^s}$ and the ciphertexts are in $\mathbb{Z}_{n^{s+1}}^*$.
(ii) g is always set to $1+n$ (that has order n^s).
(iii) The decryption phase is quite different.

The main advantage of this cryptosystem is that the only parameter to be fixed is n, while s can be adjusted according to the plaintext. In other words, unlike other cryptosystems, where one has to choose the plaintext m to be less than n, here one can choose an m of arbitrary size, and then adjust s to have $n^s > m$ and the only requirement for n is that it must be unfeasible to find its factorization.

The tradeoff between security and arithmetic precision is a crucial issue in secure signal processing applications. As we will describe later a cryptosystem that offers the possibility to work with an arbitrary precision allows us to neglect that the cryptosystem works on integer modular numbers so from now on we will describe our protocol as we have a homomorphic cryptosystem that works on approximated real numbers with arbitrary precision. A detailed discussion of this claim is given in Section 6.

3.4. Private scalar product protocol

A secure protocol for the scalar product allows Bob to compute an encrypted version of the scalar product $\langle \cdot, \cdot \rangle$ between an encrypted vector given by Alice $\mathbf{c} = E_{\mathrm{pk}}(\mathbf{x})$ and one vector $\mathbf{y}$ owned by Bob. The protocol guarantees that Bob gets nothing, while Alice gets an encrypted version of the scalar product that she can decrypt with her private key. At the end Bob learns nothing about Alice's input while Alice learns nothing except for the output of the computation. As described in [21], there are a lot of protocols proposed for this issue.

Here we use a protocol based on an additively homomorphic encryption scheme (see Algorithm 1).

After receiving z, Alice can decrypt this value with her private key to discover the output of the computation. By using the notation $(\text{input}_A; \text{input}_B) \to (\text{output}_A; \text{output}_B)$, the above protocol can be written as $(\mathbf{c} = E_{\mathrm{pk}}(\mathbf{x}); \mathbf{y}) \to (z = E_{\mathrm{pk}}(\langle \mathbf{x}, \mathbf{y} \rangle); \varnothing)$ where $\varnothing$ denotes that Bob gets no output.

```
Input: (c = E_pk(x); y)
Output: (z = E_pk(⟨x, y⟩); ∅)
PSPP(c; y)
(1)    Bob computes $w = \prod_{i=1}^{N} c_i^{y_i}$
(2)    Bob sends z to Alice
```

Algorithm 1

It is worth observing that though the above protocol is a secure one in a cryptographic sense, some knowledge about Bob's secrets is implicitly leaked through the output of the protocol itself. If, for instance, Alice can interact N times with Bob (where $N = |\mathbf{x}| = |\mathbf{w}|$ is the size of the input vectors), she can completely find out Bob's vector, by simply setting the input of the ith iteration as the vector with all 0s and a 1 in the ith position, for $i = 1, \dots, N$. This observation does not contrast with the cryptographic notion of secure multiparty computation, since a protocol is defined secure if what the parties learn during the protocol is only what they learn from the output of the computation. However, if we use the scalar product protocol described above to build more sophisticated protocols, we must be aware of this leakage of information. In the following we will refer to this way of disclosing secret information as a *sensitivity attack* after the name of a similar kind of attack usually encountered in watermarking applications [22, 23]. Note that the problems stemming from sensitivity attacks are often neglected in the privacy preserving computing literature.

3.5. Malleability

The homomorphic property that allows us to produce meaningful transformation on the plaintext modifying the ciphertext also allows an attacker to exploit it for a malicious purpose.

In our application one can imagine a competitor of Bob that wants to discredit Bob's ability to process data, and thus adds random noise to all data exchanged between Alice and Bob, making the output of the computation meaningless. Alice and Bob have no way to discover that such an attack was done, because if the attacker knows the public key of Alice, he can transform the ciphertext in the same way that Bob can, so Alice cannot distinguish between honest homomorphic computation made by Bob and malicious manipulation of the ciphertext performed by the attacker. This is a well known drawback of every protocol that uses homomorphic encryption to realize secure computation. Such a kind of attacks is called a malleability attack [24]. To prevent attackers from maliciously manipulating the content of the messages exchanged between Alice and Bob, the protocol, such as any other protocol based on homomorphic encryption, should be run on a secure channel.

4. PERCEPTRON

We are now ready to describe how to use the Paillier cryptosystem and the private scalar product protocol to build a protocol for oblivious neural network computation. We start by describing the instance of a single neuron, in order to clarify how the weighted sum followed by the activation function shaping the neuron can be securely computed.

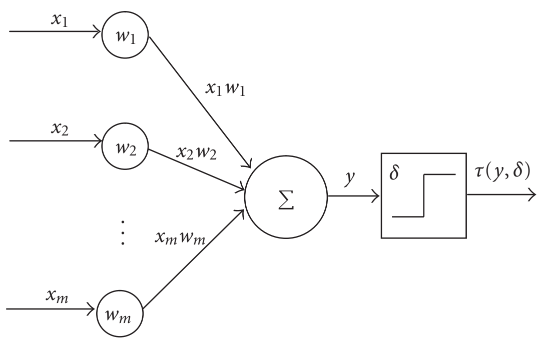

Figure 1: A perceptron is a binary classifier that performs a weighted sum of the inputs $x_1, \dots, x_m$ by means of the weights $w_1, \dots, w_m$ followed by an activation function usually implemented by a threshold operation.

A single neuron in a NN is usually referred to as perceptron. A perceptron (see Figure 1) is a binary classifier that performs a weighted sum of the input $x_1, \dots, x_m$ by means of the weights $w_1, \dots, w_m$ followed by an activation function (usually a threshold operation). So if $y = \sum_{i=1}^{m} x_i w_i$, the output of the perceptron will be

$$\tau(y, \delta) = \begin{cases} 1 & \text{if } y \geq \delta, \\ 0 & \text{if } y < \delta. \end{cases} \tag{6}$$

We also address the case where the activation function is a sigmoid function, in this case the output of the perceptron is

$$\sigma(y, \alpha) = \frac{1}{1 + e^{-\alpha y}}. \tag{7}$$

This function is widely used in feedforward multilayer NNs because of the following relation:

$$\frac{d\sigma(x, \alpha)}{dx} = \alpha\sigma(x, \alpha)(1 - \sigma(x, \alpha)) \tag{8}$$

that is easily computable and simplifies the backpropagation training algorithm execution [25].

In the proposed scenario the data are distributed as follows: Alice owns her private input $\mathbf{x}$, Bob owns the weights $\mathbf{w}$, and at the end only Alice obtains the output. Alice will provide her vector in encrypted format ($\mathbf{c} = E_{\text{pk}}(\mathbf{x})$) and will receive the output in an encrypted form. We already showed how to compute an encrypted version of y, the scalar product between $\mathbf{x}$ and $\mathbf{w}$. Let us describe now how this computation can be linked to the activation function in order to obtain a secure protocol $(\mathbf{c}; \mathbf{w}, \delta) \rightarrow (E_{\text{pk}}(\tau(\langle \mathbf{x}, \mathbf{w} \rangle, \delta)); \varnothing)$ in the case of a threshold activation function, or $(\mathbf{x}; \mathbf{w}, \alpha) \rightarrow (E_{\text{pk}}(\sigma(\langle \mathbf{x}, \mathbf{w} \rangle, \alpha)); \varnothing)$ in the case of the sigmoid activation function. In order to avoid any leakage of information, an obfuscation step is introduced to cover the scalar product and the parameters of the activation function.

```
Input: $(\mathbf{c}; \mathbf{w}, \delta)$
Output: $(E_{\text{pk}}(\tau(\langle \mathbf{x}, \mathbf{y} \rangle, \delta)); \emptyset)$
PerceptronThreshold$(\mathbf{c}; \mathbf{w}, \delta)$
(1) Bob computes $y = \prod_{i=1}^{N} c_i^{w_i}$
(2) Bob computes $y = (y \cdot E_{\text{pk}}(-\delta))^a$ with random $a$ and
    $a > 0$
(3) Bob sends $y$ to Alice
(4) Alice's output is 1 if $D_{\text{sk}}(y) \geq 0$; else it is 0
```

Algorithm 2

```
Input: $(\mathbf{c}; \mathbf{w}, \alpha)$
Output: $(E_{\text{pk}}(\sigma(\langle \mathbf{x}, \mathbf{y} \rangle, \alpha)); \emptyset)$
PerceptronSigmoid$(\mathbf{x}; \mathbf{w}, \alpha)$
(1) Bob computes $y = \prod_{i=1}^{N} c_i^{w_i}$
(2) Bob computes $\eta = y^{\alpha}$ and
(3) Bob sends $\eta$
(4) Alice decrypts $\eta$ and computes her output
    $\sigma(D_{\text{sk}}(\eta), 1)$
```

Algorithm 3

4.1. *Secure threshold evaluation*

What we want here is that Alice discovers the output of the comparison without knowing the terms that are compared. Moreover, Bob cannot perform such a kind of computation by himself, as thresholding is a highly non-linear function, thus homomorphic encryption cannot help here. The solution we propose is to obfuscate the terms of the comparison and give them to Alice in such a way that Alice can compute the correct output without knowing the real values of the input. To be specific, let us note that $\tau(y, \delta) = \tau(f(y - \delta), 0)$ for every function such that $\text{sign}(f(x)) = \text{sign}(x)$. So Bob needs only to find a randomly chosen function that he can compute in the encrypted domain, that transforms $y - \delta$ into a value indistinguishable from purely random values and keeps the sign unaltered. In our protocol, the adopted function is $f(x) = ax$ with $a > 0$. Due to the homomorphic property of the cryptosystem, Bob can efficiently compute

$$E_{\text{pk}}(\langle \mathbf{x}, \mathbf{w} \rangle - \delta)^a \sim E_{\text{pk}}(a(\langle \mathbf{x}, \mathbf{w} \rangle - \delta)), \tag{9}$$

where $\sim$ means that they contain the same plaintext. Next, Bob sends this encrypted value to Alice that can decrypt the message and check if $a(\langle \mathbf{x}, \mathbf{w} \rangle - \delta) > 0$. Obviously, this gives no information to Alice on the true values of $\langle \mathbf{x}, \mathbf{w} \rangle$ and δ. In summary, the protocol for the secure evaluation of the perceptron is shown in Algorithm 2.

4.2. *Secure sigmoid evaluation*

The main idea underlying the secure evaluation of the sigmoid function is similar to that used for thresholding. Even in this case we note that $\sigma(y, \alpha)$ depends only on the product of the two inputs, say if $y\alpha = y'\alpha'$, then $\sigma(y, \alpha) = \sigma(y', \alpha')$. So what Bob can do to prevent Alice to discover the output of scalar product and the parameter of the sigmoid α is to give Alice the product of those values, that Bob can compute in the encrypted domain and that contains the same information of the output of the sigmoid function. In fact, as the sigmoid function could be easily inverted, the amount of information provided by $\sigma(y, \alpha)$ is the same of αy. The solution we propose, then, is the following: by exploiting again the homomorphic property of the cryptosystem Bob computes $E_{\text{pk}}(y)^{\alpha} \sim E_{\text{pk}}(\alpha y)$. Alice can decrypt the received value and compute the output of the sigmoid function. The protocol for the sigmoid-shaped perceptron is shown in Algorithm 3.

4.3. *Security against sensitivity attacks*

Before passing to the description of the protocol for the computation of a whole NN, it is instructive to discuss the sensitivity attack at the perceptron level. Let us consider first the case of a threshold activation function: in this case the perceptron is nothing but a classifier whose decision regions are separated by a hyperplane with coefficients given by the vector $\mathbf{w}$. Even if Alice does not have access to the intermediate value $\langle \mathbf{x}, \mathbf{w} \rangle$, she can still infer some useful information about $\mathbf{w}$ by proceeding as follows. She feeds the perceptron with a set of random sequences until she finds two sequences lying in different decision regions, that is, for one sequence the output of the perceptron is one, while for the other is zero. Then Alice applies a bisection algorithm to obtain a vector that lies on the border of the decision regions. By iterating the above procedure, Alice can easily find m points belonging to the hyperplane separating the two decision regions of the perceptron, hence she can infer the values of the m unknowns contained in $\mathbf{w}$. In the case of a sigmoid activation function, the situation is even worse, since Alice only needs to observe $m + 1$ values of the product αy to determine the $m + 1$ unknowns $(w_1, w_2 \cdots w_m; \alpha)$.

Note that it is impossible to prevent the sensitivity attacks described above by working at the perceptron level, since at the end of the protocol the output of the perceptron is the minimum amount of disclosed information. As it will be outlined in the next section, this is not the case when we are interested in using the perceptron as an intermediate step of a larger neural network.

5. MULTILAYER FEEDFORWARD NETWORK

A multilayer feedforward network is composed by n layers, each having m_i neurons ($i = 1 \cdots n$). The network is then composed by $N = \sum_{i=1}^{n} m_i$ neurons. Every neuron is identified by two indexes, the superscript refers to the layer the neuron belongs to, the subscript refers to its position in the layer (e.g., $\mathbf{w}_3^2$ indicates the weights vector for the third neuron in the second layer, while its components will be referred to as $w_{3,j}^2$). An example of such a network is given in Figure 2. The input of each neuron in the ith layer is the weighted sum of the outputs of the neurons of the $(i-1)$th layer. The input of the first layer of the NN is Alice's vector, while the output of the last layer is the desired output of the computation. Each neuron that is not an output neuron is called *hidden neuron*.

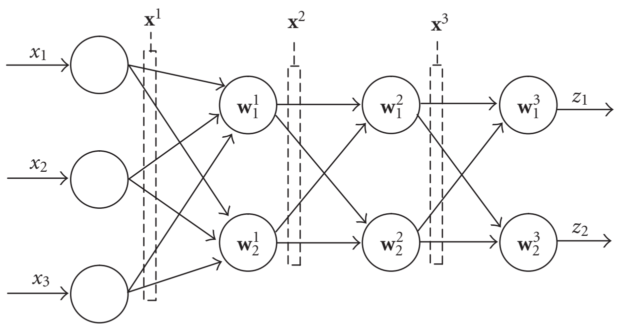

FIGURE 2: This network has $n = 3$ layers. The network has three inputs, and all layers are composed of two neurons ($m_1 = m_2 = m_3 = 2$). The network is so composed of six neurons ($N = 6$). Let us note that the input neurons are not counted as they do not perform computation. For the sake of simplicity, the weights vector of every neuron is represented into the neuron, and not on the edge.

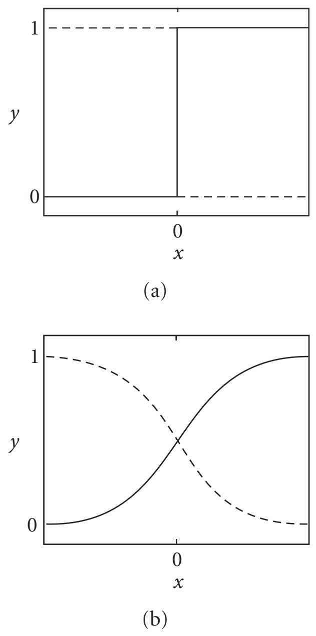

FIGURE 3: Both threshold and sigmoid functions are antisymmetric with respect to $(0, 1/2)$ as shown, that is, $\tau(-y, -\delta) = 1 - \tau(y, \delta)$ and $\sigma(-y, \alpha) = 1 - \sigma(y, \alpha)$.

In addition to protecting the weights of the NN, as described in the previous section, the protocol is designed to protect also the output of each of those neurons. In fact the simple composition of N privacy preserving perceptrons would disclose some side information (the output of the hidden neurons nodes) that could be used by Alice to run a sensitivity attack at each NN node.

The solution adopted to solve this problem is that Bob does not delegate Alice to compute the real output of the hidden perceptrons, but an apparently random output, so that, as it will be clarified later, the input of each neuron of the ith layer will not be directly the weighted sum of the outputs of the neurons of the $(i - 1)$th layer, but an obfuscation of them. To be specific, let us focus on the threshold activation function, in this case every neuron will output a 0 or a 1. The threshold function is antisymmetric with respect to $(0, 1/2)$ as shown in Figure 3. That is, we have that $y \geq \delta \Rightarrow -y \leq -\delta$ or equivalently:

$$\tau(-y, -\delta) = 1 - \tau(y, \delta). \tag{10}$$

Then, if Bob changes the sign of the inputs of the threshold with 0.5 probability, he changes the output of the computation with the same probability, and Alice computes an apparently random output according to her view. Then she encrypts this value, sends it to Bob that can flip it again in the encrypted domain, so that the input to the next layer will be correct.

Also the sigmoid is antisymmetric with respect to $(0, 1/2)$, since we have that $1/(1 + e^{-\alpha y}) = 1 - 1/(1 + e^{\alpha y})$ or equivalently:

$$\sigma(-y, \alpha) = 1 - \sigma(y, \alpha), \tag{11}$$

then if Bob flips the product inputs with 0.5 probability, the sign of the value that Bob sends to Alice will be again apparently random. Alice will still be able to use this value to compute the output of the activation function that will appear random to her. However, Bob can retrieve the correct output, since he knows whether he changed the sign of the inputs of the activation function or not. Note that Bob can flip the sign of one or both the inputs of τ or σ in the encrypted domain, and he can also retrieve the real output by still working in the encrypted domain since he can do this by means of a simple linear operation (multiplication by 1 or -1 and subtractions).

5.1. Multilayer network with threshold

We are now ready to describe the final protocol for a multilayer feedforward neural network whose neurons use the threshold as activation function. The privacy preserving perceptron protocol presented before is extended adding an input for Bob using the following notation: given $\xi \in \{+, -\}$, we define

PERCEPTRONTHRESHOLD $(\mathbf{c}; \mathbf{w}, \delta, \xi) \rightarrow (E_{\text{pk}}(\tau(\xi\langle \mathbf{x}, \mathbf{w}\rangle, \xi\delta)); \varnothing)$.

The Alice's encrypted input vector will be the input for the first layer, that is $\mathbf{c}^1 = \mathbf{c}$. With this new definition, we obtain the protocol shown in Algorithm 4.

To understand the security of the protocol, let us note that if Bob flips the sign of the input in the threshold with probability 1/2, Alice does not learn anything from the computation of the threshold function hence achieving perfect security according to Shannon's definition. In fact, it is like performing a one time pad on the neuron output bit. This is not true in the case of the sigmoid, for which an additional step must be added.

5.2. Multilayer network with sigmoid

Even in this case, we need to extend the perceptron protocol presented before by adding an input to allow Bob to flip the

```
Input: $(\mathbf{c}; \mathbf{w}_j^i, \delta_j^i)$ with $i = 1 \cdots n$, $j = 1 \cdots m_i$
Output: $(E_{pk}(\mathbf{z}); \emptyset)$ where $\mathbf{z}$ is the output of the last layer of the network
PPNNThreshold$(\mathbf{c}; \mathbf{w}_j^i, \delta_j^i)$
(1)   $\mathbf{c}^1 = \mathbf{c}$
(2)   for $i = 1$ to $n - 1$
(3)       for $j = 1$ to $m_i$
(4)           Bob picks $\xi \in \{+, -\}$ at random
(5)           $(E_{pk}(\tau(\xi\langle \mathbf{x}^i, \mathbf{w}_j^i\rangle, \xi\delta_j^i)); \emptyset) = $ PerceptronThreshold$(\mathbf{c}^i; \mathbf{w}_j^i, \delta_j^i, \xi)$
(6)           Alice decrypts the encrypted output and computes the new input $\mathbf{x}^{i+1}$, sending
              back to Bob $c_j^{i+1} = E_{pk}(x_j^{i+1})$
(7)           if $\xi = $ "$-$"
(8)               Bob sets $c_j^{i+1} = E_{pk}(1) \cdot (c_j^{i+1})^{-1}$
(9)   // last layer does not obfuscate the output
(10)  for $j = 1$ to $m_n$
(11)      $(z_j; \emptyset) = $ PerceptronThreshold$(\mathbf{x}^{\mathbf{n}}; \mathbf{w}_j^n, \delta_j^n, +)$
```

Algorithm 4

sigmoid input:

PerceptronSigmoid $(\mathbf{c}; \mathbf{w}, \alpha, \xi) \rightarrow (E_{pk}(\sigma(\xi\langle \mathbf{x}, \mathbf{w}\rangle, \alpha)); \emptyset)$.

At this point we must consider that, while the threshold function gives only one bit of information, and the flipping operation carried out by Bob completely obfuscates Alice's view, the case of the sigmoid is quite different: if Bob flips the inputs with probability 0.5, Alice will not learn if the input of the sigmoid was originally positive or negative, but she will learn the product $\pm\alpha y$. This was not a problem for the perceptron case, as knowing z or this product is the same (due to the invertibility of sigmoid function). For the multilayer case, instead, it gives to Alice more information than what she needs, and this surplus of information could be used to perform a sensitivity attack.

Our idea to cope with this attack at the node level is to randomly scramble the order of the neurons in the layer for every execution of the protocol except for the last one. If the layer i has m_i neurons we can scramble them in m_i! different ways. We will call π_{r_i} the random permutation used for the layer i, depending on some random seed r_i (where $i = 1, \ldots, n-1$) so that the protocol will have a further input $\mathbf{r}$. Evidently, the presence of the scrambling operator prevents Alice from performing a successful sensitivity attack. In summary, the protocol for the evaluation of a multilayer network with sigmoid activation function, using the same notation of the threshold case, is shown in Algorithm 5.

5.3. Sensitivity attack

Before concluding this section, let us go back to the sensitivity attack. Given that the intermediate values of the computation are not revealed, a sensitivity attack is possible only at the whole network level. In other words, Alice could consider the NN as a parametric function with the parameters corresponding to the NN weights, and apply a sensitivity attack to it. Very often, however, a multilayer feedforward NN implements a complicated, hard-to-invert function, so that discovering all the parameters of the network by considering it as a black box requires a very large number of interactions. To avoid this kind of attack, then, we can simply assume that Bob limits the number of queries that Alice can ask, or require that Alice pays an amount of money for each query.

5.4. Protecting the network topology

As a last requirement Bob may desire that Alice does not learn anything about the NN topology. Though strictly speaking this is a very ambitious goal, Bob may distort Alice's perception of the NN by randomly adding some fake neurons to the hidden layers of the network, as shown in Figure 4. As the weights are kept secret, Bob should randomly set the inbound weight of each neuron. At the same time Bob has to reset the outbound weights, so that the fake neurons will not change the final result of the computation. The algorithms that we obtain by considering this last modification are equal to those described so far, the only difference being in the topology of Bob's NN. Note that for networks with sigmoid activation functions, adding fake neurons will also increase the number of random permutations that can be applied to avoid sensitivity attacks.

6. HANDLING NONINTEGER VALUES

At the end of Section 3.3 we made the assumption that the Paillier encryption scheme, noticeably the Damgård-Jurik extension, works properly on noninteger values and satisfies the additive homomorphic properties on such kind of data to simplify the analysis reported in the subsequent sections. Indeed, rigorously speaking, this is not true. We now analyze more formally every step of the proposed protocols showing how the assumption we made in Section 3.3 is a reasonable one.

To start with, let us remember that the Damgård-Jurik cryptosystem allows to work on integers in the range $\{0, \ldots, n^s - 1\}$. First of all we map, in a classic way, the positive numbers in $\{0, \ldots, (n^s - 1)/2\}$, and the negative ones in $\{(n^s - 1)/2+1, \ldots, n^s-1\}$, with $-1 = n^s-1$. Then, given a real value $x \in \mathbb{R}$, we can quantize it with a quantization factor Q,

```
Input: $(\mathbf{c}; \mathbf{w}_j^i, \alpha_j^i, \mathbf{r})$ with $i = 1 \cdots n, j = 1 \cdots m_i$
Output: $(E_{pk}(\mathbf{z}); \varnothing)$ where $\mathbf{z}$ is the output of the last layer of the network
PPNNSIGMOID$(\mathbf{c}; \mathbf{w}_j^i, \alpha_j^i, \mathbf{r})$
(1)  for $i = 1$ to $n - 1$
(2)      Bob permutes neurons position in layer $i$ using random permutation $\pi_{r_i}$
(3)  // now the network is scrambled, and the protocol follows as before
(4)  $\mathbf{x}^1 = \mathbf{x}$
(5)  for $i = 1$ to $n - 1$
(6)      for $j = 1$ to $m_i$
(7)          Bob picks $\xi \in \{+, -\}$ at random
(8)          $(E_{pk}(\sigma(\xi\langle \mathbf{x}^i, \mathbf{w}_j^i \rangle, \alpha_j^i)); \varnothing)$ = PerceptronSigmoid$(\mathbf{c}^i; \mathbf{w}_j^i, \alpha_j^i, \xi)$
(9)          Alice decrypts the encrypted output and computes the new input $\mathbf{x}^{i+1}$, sending
             back to Bob $c_j^{i+1} = E_{pk}(x_j^{i+1})$
(10)         if $\xi$ = "−"
(11)             Bob sets $c_j^{i+1} = E_{pk}(1) \cdot (c_j^{i+1})^{-1}$
(12) // last layer does not obfuscate the output
(13) for $j = 1$ to $m_n$
(14)     $(z_j; \varnothing)$ = PerceptronSigmoid$(\mathbf{c}^n; \mathbf{w}_j^n, \alpha_j^n, +)$
```

Algorithm 5

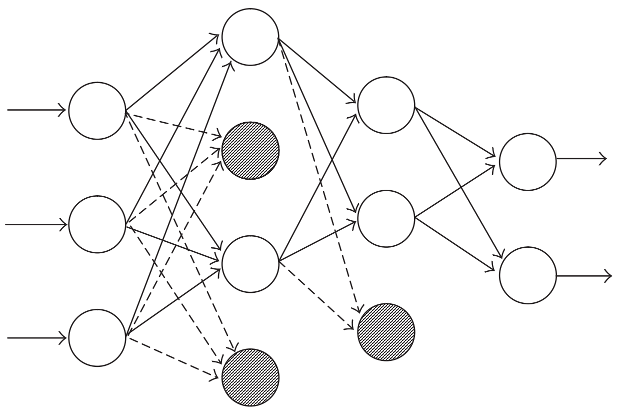

Figure 4: To obfuscate the number and position of the hidden neurons Bob randomly adds fake neurons to the NN. Fake neurons do not affect the output of the computation as their outbound weights are set to 0. Inbound weights are dotted as they are meaningless.

and approximate it as $\overline{x} = \lfloor x/Q \rfloor \simeq x/Q$ for a sufficiently thin quantization factor. Clearly the first homomorphic property still stands, that is,

$$D_{sk}(E_{pk}(\overline{x}_1) \cdot E_{pk}(\overline{x}_2)) = \overline{x}_1 + \overline{x}_2 \simeq \frac{x_1 + x_2}{Q}. \qquad (12)$$

This allows Bob to perform an arbitrarily number of sums among cipher texts. Also the second property holds, but with a drawback. In fact:

$$D_{sk}(E_{pk}(\overline{x})^{\overline{a}}) = \overline{a} \cdot \overline{x} \simeq \frac{a \cdot x}{Q^2}. \qquad (13)$$

The presence of the Q^2 factor has two important consequences:

(1) the size of the encrypted numbers grows exponentially with the number of multiplications;

(2) Bob must disclose to Alice the number of multiplications, so that Alice can compensate for the presence of the Q^2 factor.

The first drawback is addressed with the availability of Damgård-Jurik cryptosystem that allows us, by increasing s, to cipher bigger numbers. The second one imposes a limit on the kind of secure computation that we can perform using the techniques proposed here.

We give here an upper bound for the bigger integer that can be encrypted, that forces us to select an appropriate parameter s for the Damgård-Jurik cryptosystem.

In the neural network protocol, the maximum number of multiplications done on a quantized number is equal to two: the first in the scalar product protocol and the second with a random selected number in the secure thresholding evaluation or with the α parameter in the secure sigmoid evaluation. Assume that the random values and the α parameters are bounded by R.

Let X be the upper bound for the norm of Alice's input vector, and W an upper bound for the weight vectors norm. We have that every scalar product computed in the protocol is bounded by $|\mathbf{x}| \cdot |\mathbf{w}| \cos(\widehat{\mathbf{xw}}) \leq XW$. Given a modulo n sufficiently high for security purposes, we have to select s such that

$$s \geq \left\lceil \log_n \frac{2XWR}{Q^2} \right\rceil, \qquad (14)$$

where the factor 2 is due to the presence of both positive and negative values.

Other solutions for working with noninteger values can be found in [8] where a protocol to evaluate a polynomial on floating-point numbers is defined (but the exponent must be chosen in advance), and [26], where a sophisticated cryptosystem based on lattice properties allowing computation with rational values is presented (even in this case, however, a bound exists on the number of multiplications that can be carried out to allow a correct decryption).

7. IMPLEMENTATION OF THE PROTOCOL

In this section a practical implementation of the proposed protocol is described, and a case study execution that will give us some numerical results in terms of computational and bandwidth resource needed is analyzed.

7.1. *Client-server application*

We developed a client-server application based on the Java remote method invocation technology.[1] The application, based on the implementation of the Damgård-Jurik cryptosystem available on Jurik's homepage,[2] is composed of two methods, one for the initialization of the protocol (where public key and public parameters are chosen) and one for the evaluation of every layer of neurons.

7.2. *Experimental data*

From the UCI machine learning repository,[3] the data set chosen by Gorman and Sejnowski in their study about the classification of sonar signals by means of a neural network [27] has been selected. The task is to obtain a network able to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. Following the author's results, we have trained a NN with 12 hidden neurons and sigmoid activation function with the standard backpropagation algorithm, obtaining an accuracy of 99.8% on the training set and 84.7% on the test set.

7.3. *Experimental setup*

To protect our network we have embedded it in a network made of 5 layers of 15 neurons each, obtaining a high level of security as the ratio of real neurons on fake one is really low, in fact it is $12/75 = 0.16$. The public key n is 1024 bit long, and the s parameter has been set to 1, without any problem for a very thin quantization factor $Q = 10^{-6}$. We have then initialized every fake neuron with connections from every input neuron in a way that they will look the same of the real ones, setting the weights of the connection at random. Then we have deployed the application on two mid-level notebooks, connected on a LAN network.

The execution of the whole process took 11.7 seconds, of which 9.3 on server side, with a communication overhead of 76 kb. Let us note that no attempt to optimize the execution time was done, and as seen the client computation is negligible. These results confirm the practical possibility to run a neural network on an input provided in encrypted format.

[1] http://java.sun.com/javase/technologies/core/basic/rmi
[2] http://www.daimi.au.dk/~jurik/research.html
[3] http://www.ics.uci.edu/~mlearn/MLRepository.html

8. CONCLUSIONS

In artificial intelligence applications, the possibility that the owner of a specific expertise is asked to apply its knowledge to process some data without that the privacy of the data owner is violated is of crucial importance. In this framework, the possibility of processing data and signals directly in the encrypted domain is an invaluable tool, upon which secure and privacy preserving protocols can be built. Given the central role that neural network computing plays in modern artificial intelligence applications, we devoted our attention to NN-based privacy-preserving computation, where the knowledge embedded in the NN as well as the data the NN operates on are protected. The proposed protocol relies on homomorphic encryption; for those tasks that cannot be handled by means of homomorphic encryption, a limited amount of interaction between the NN owner and the user is introduced; however, in contrast to previous works, the interaction is kept to a minimum, without resorting to multiparty computation protocols. Any unnecessary disclosure of information has been avoided, keeping all the internal computations secret such that at the end of the protocol the user only knows the final output of the NN. Future research will be focused on investigating the security of the network topology obfuscation proposed here, and on the design of more efficient obfuscation strategies. Moreover, the possibility of training the network in its encrypted form will also be studied.

ACKNOWLEDGMENTS

The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract no 034238-SPEED. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

REFERENCES

[1] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[2] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," in *Foundations of Secure Computation*, pp. 169–178, Academic Press, New York, NY, USA, 1978.

[3] B. Pinkas, "Cryptographic techniques for privacy-preserving data mining," *ACM SIGKDD Explorations Newsletter*, vol. 4, no. 2, pp. 12–19, 2002, ACM special interest group on knowledge discovery and data mining.

[4] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or a completeness theorem for protocols with honest majority," in *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC '87)*, pp. 218–229, ACM Press, New York, NY, USA, May 1987.

[5] D. Chaum, C. Crépeau, and I. Damgård, "Multiparty unconditionally secure protocols," in *Proceedings of the 20th Annual*

ACM Symposium on Theory of Computing (STOC '88), pp. 11–19, ACM Press, Chicago, Ill, USA, May 1988.

[6] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '00)*, vol. 1880 of *Lecture Notes in Computer Science*, pp. 36–54, Santa Barbara, Calif, USA, August 2000.

[7] R. Agrawal and R. Srikant, "Privacy-preserving data mining," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 439–450, ACM Press, Dallas, Tex, USA, May 2000.

[8] Y.-C. Chang and C.-J. Lu, "Oblivious polynomial evaluation and oblivious neural learning," *Theoretical Computer Science*, vol. 341, no. 1–3, pp. 39–54, 2005.

[9] S. Laur, H. Lipmaa, and T. Mielikäihen, "Cryptographically private support vector machines," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '06)*, pp. 618–624, ACM Press, Philadelphia, Pa, USA, August 2006.

[10] M. Kantarcioglu and J. Vaidya, "Privacy preserving naive bayes classifier for horizontally partitioned data," in *Proceedings of the Workshop on Privacy Preserving Data Mining*, Melbourne, Fla, USA, November 2003.

[11] Z. Yang and R. N. Wright, "Improved privacy-preserving Bayesian network parameter learning on vertically partitioned data," in *Proceedings of the 21st International Conference on Data Engineering Workshops (ICDEW '05)*, p. 1196, IEEE Computer Society, Tokyo, Japan, April 2005.

[12] R. Wright and Z. Yang, "Privacy-preserving Bayesian network structure computation on distributed heterogeneous data," in *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '04)*, pp. 713–718, ACM Press, Seattle, Wash, USA, August 2004.

[13] G. Jagannathan and R. N. Wright, "Privacy-preserving distributed k-means clustering over arbitrarily partitioned data," in *Proceeding of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD '05)*, pp. 593–599, ACM Press, Chicago, Ill, USA, August 2005.

[14] A. C. Yao, "Protocols for secure computations," in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pp. 160–164, Chicago, Ill, USA, November 1982.

[15] A. Yao, "How to generate and exchange secrets," in *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS '86)*, pp. 162–167, Toronto, Ontario, Canada, October 1986.

[16] P. Pailler, "Public-key cryptosystems based on composite degree residuosity classes," in *Proceedings of International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '99)*, vol. 1592 of *Lecture Notes is Computer Science*, pp. 223–238, Springer, Prague, Czech Republic, May 1999.

[17] M. Barni, C. Orlandi, and A. Piva, "A privacy-preserving protocol for neural-network-based computation," in *Proceedings of the 8th Multimedia and Security Workshop (MM & Sec '06)*, pp. 146–151, ACM Press, Geneva, Switzerland, September 2006.

[18] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[19] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of Paillier's probabilistic public-key system," in *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography (PKC '01)*, pp. 119–136, Cheju Island, Korea, February 2001.

[20] D. Catalano, *The bit security of Paillier's encryption scheme and a new, efficient, public key cryptosystem*, Ph.D. thesis, Università di Catania, Catania, Italy, 2002.

[21] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen, "On private scalar product computation for privacy-preserving data mining," in *Proceedings of the 7th Annual International Conference in Information Security and Cryptology (ICISC '04)*, pp. 104–120, Seoul, Korea, December 2004.

[22] I. J. Cox and J.-P. M. G. Linnartz, "Public watermarks and resistance to tampering," in *Proceedings the 4th IEEE International Conference on Image Processing (ICIP '97)*, vol. 3, pp. 3–6, Santa Barbara, Calif, USA, October 1997.

[23] T. Kalker, J.-P. M. G. Linnartz, and M. van Dijk, "Watermark estimation through detector analysis," in *Proceedings of IEEE International Conference on Image Processing (ICIP '98)*, vol. 1, pp. 425–429, Chicago, Ill, USA, October 1998.

[24] D. Dolev, C. Dwork, and M. Naor, "Nonmalleable cryptography," *SIAM Journal on Computing*, vol. 30, no. 2, pp. 391–437, 2000.

[25] T. M. Mitchell, *Machine Learning*, McGraw-Hill, New York, NY, USA, 1997.

[26] P.-A. Fouque, J. Stern, and J.-G. Wackers, "CryptoComputing with rationals," in *Proceedings of the 6th International Conference on Financial-Cryptography (FC '02)*, vol. 2357 of *Lecture Notes in Computer Science*, pp. 136–146, Southampton, Bermuda, March 2002.

[27] R. P. Gorman and T. J. Sejnowski, "Analysis of hidden units in a layered network trained to classify sonar targets," *Neural Networks*, vol. 1, no. 1, pp. 75–89, 1988.