Machine Learning Classification over Encrypted Data
Raphaël Bost* Raluca Ada Popa† Stephen Tu‡ Shafi Goldwasser‡

Abstract

Machine learning classification is used in numerous settings nowadays, such as medical or genomics predictions, spam detection, face recognition, and financial predictions. Due to privacy concerns, in some of these applications, it is important that the data and the classifier remain confidential.

In this work, we construct three major classification protocols that satisfy this privacy constraint: hyperplane decision, Naïve Bayes, and decision trees. We also enable these protocols to be combined with AdaBoost. At the basis of these constructions is a new library of building blocks for constructing classifiers securely; we demonstrate that this library can be used to construct other classifiers as well, such as a multiplexer and a face detection classifier.

We implemented and evaluated our library and classifiers. Our protocols are efficient, taking milliseconds to a few seconds to perform a classification when running on real medical datasets.

1 Introduction

Classifiers are an invaluable tool for many tasks today, such as medical or genomics predictions, spam detection, face recognition, and finance. Many of these applications handle sensitive data [WGH12, SG11, SG13], so it is important that the data and the classifier remain private.

Consider the typical setup of supervised learning, depicted in Figure 1. Supervised learning algorithms consist of two phases: (i) the training phase during which the algorithm learns a model \( w \) from a data set of labeled examples, and (ii) the classification phase that runs a classifier \( C \) over a previously unseen feature vector \( x \), using the model \( w \) to output a prediction \( C(x, w) \).

In applications that handle sensitive data, it is important that the feature vector \( x \) and the model \( w \) remain secret to one or some of the parties involved. Consider the example of a medical study or a hospital having a model built out of the private medical profiles of some patients; the model is sensitive because it can leak information about the patients, and its usage has to be HIPAA\(^1\) compliant. A client wants to use the model to make a prediction about her health (e.g., if she is likely to contract a certain disease, or if she would be treated successfully at the hospital), but does not want to reveal her sensitive medical profile. Ideally, the hospital and the client run a protocol at the end of which the client learns one bit (“yes/no”), and neither party learns anything else about the other party’s input. A similar setting arises for a financial institution (e.g., an insurance company) holding a sensitive model, and a customer wanting to estimate rates or quality of service based on her personal information.

Throughout this paper, we refer to this goal shortly as privacy-preserving classification. Concretely, a client has a private input represented as a feature vector \( x \), and the server has a private input consisting of a private model \( w \). The way the model \( w \) is obtained is independent of our protocols here. For example, the server could have computed the model \( w \) after running the training phase on plaintext data as usual. Only the classification needs to be privacy-preserving: the client should learn \( C(x, w) \) but nothing else about the model \( w \), while the server should not learn anything about the client’s input or the classification result.

* Direction Générale de l’Armement - Maitrise de l’Information. Work done while visiting MIT CSAIL. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the DGA or the French Government.

† ETH Zürich
‡ MIT CSAIL
1 Health Insurance Portability and Accountability Act of 1996
Figure 1: Model overview. Each shaded box indicates private data that should be accessible to only one party: the dataset and the model to the server, and the input and prediction result to the client. Each straight non-dashed rectangle indicates an algorithm, single arrows indicate inputs to these algorithms, and double arrows indicate outputs.

<table>
<thead>
<tr>
<th>Machine learning algorithm</th>
<th>Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perceptron</td>
<td>Hyperplane decision</td>
</tr>
<tr>
<td>Least squares</td>
<td>Hyperplane decision</td>
</tr>
<tr>
<td>Fischer linear discriminant</td>
<td>Hyperplane decision</td>
</tr>
<tr>
<td>Support vector machine</td>
<td>Hyperplane decision</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>Naive Bayes</td>
</tr>
<tr>
<td>Decision trees (ID3/C4.5)</td>
<td>Decision trees</td>
</tr>
</tbody>
</table>

Table 1: Machine learning algorithms and their classifiers, defined in Section 3.1.

In this work, we construct efficient privacy-preserving protocols for three of the most common classifiers: hyperplane decision, Naive Bayes, and decision trees, as well as a more general classifier combining these using AdaBoost. These classifiers are widely used – even though there are many machine learning algorithms, most of them end up using one of these three classifiers, as described in Table 1.

While generic secure multi-party computation [Yao82, GMW87, HKS10, MNPS04, BDNP08] can implement any classifier in principle, due to their generality, such schemes are not efficient for common classifiers. As described in Section 10.5, on a small classification instance, such tools ([HKS10, BDNP08]) ran out of memory on a powerful machine with 256GB of RAM; also, on an artificially simplified classification instance, these protocols ran \( \approx 500 \) times slower than our protocols ran on the non-simplified instance.

Hence, protocols specialized to the classification problem promise better performance. However, most existing work in machine learning and privacy [LP00, DHC04, WY04, ZW05, ZW05, BDMN05, VKC08, GLN12] focuses on preserving privacy during the training phase, and does not address classification. The few works on privacy-preserving classification either consider a weaker security setting in which the client learns the model [BLN13] or focus on specific classifiers (e.g., face detectors [EFG09, SSW09, AB06, AB07]) that are useful in limited situations.

Designing efficient privacy-preserving classification faces two main challenges. The first is that the computation performed over sensitive data by some classifiers is quite complex (e.g., decision trees), making it hard to support efficiently. The second is providing a solution that is more generic than the three classifiers: constructing a separate solution for each classifier does not provide insight into how to combine these classifiers or how to construct other classifiers. Even though we contribute privacy-preserving protocols for three of the most common classifiers, various settings use other classifiers or use a combination of these three classifiers (e.g., AdaBoost). We address these challenges using two key techniques.

Our main technique is to identify a set of core operations over encrypted data that underlie many classification protocols. We found these operations to be comparison, argmax, and dot product. We use efficient protocols for each one of these, either by improving existing schemes (e.g., for comparison) or by constructing new schemes (e.g., for argmax).

Our second technique is to design these building blocks in a composable way, with regard to both functionality and security. To achieve this goal, we use a set of sub-techniques:

- The input and output of all our building blocks are data encrypted with additively homomorphic encryption. In addition, we provide a mechanism to switch from one encryption scheme to another. Intuitively, this enables a building block’s output to become the input of another building block;
- The API of these building blocks is flexible: even though each building block computes a fixed function, it allows
a choice of which party provides the inputs to the protocol, which party obtains the output of the computation, and whether the output is encrypted or decrypted;

- The security of these protocols composes using modular sequential composition [Can98].

We emphasize that the contribution of our building blocks library goes beyond the classifiers we build in this paper: a user of the library can construct other privacy-preserving classifiers in a modular fashion. To demonstrate this point, we use our building blocks to construct a multiplexer and a classifier for face detection, as well as to combine our classifiers using AdaBoost.

We then use these building blocks to construct novel privacy-preserving protocols for three common classifiers. Some of these classifiers incorporate additional techniques, such as an efficient evaluation of a decision tree with fully homomorphic encryption (FHE) based on a polynomial representation requiring only a small number of multiplications and based on SIMD FHE slots (see Section 7.2). All of our protocols are secure against passive adversaries (see Section 3.2.3).

We also provide an implementation and an evaluation of our building blocks and classifiers. We evaluate our classifiers on real datasets with private data about breast cancer, credit card approval, audiology, and nursery data; our algorithms are efficient, running in milliseconds up to a few seconds, and consume a modest amount of bandwidth.

The rest of the paper is organized as follows. Section 2 describes related work, Section 3 provide the necessary machine learning and cryptographic background, Section 4 presents our building blocks, Sections 5–8 describe our classifiers, and Sections 9–10 present our implementation and evaluation results.

## 2 Related work

Our work is the first to provide efficient privacy-preserving protocols for a broad class of classifiers.

Secure two-party computation protocols for generic functions exist in theory [Yao82, GMW87, LP07, IPS08, LP09] and in practice [HKS10, MNPS04, BDNP08]. However, these rely on heavy cryptographic machinery, and applying them directly to our problem setting would be too inefficient as exemplified in Section 10.5.

Previous work focusing on privacy-preserving machine learning can be broadly divided into two categories: (i) techniques for privacy-preserving training, and (ii) techniques for privacy-preserving classification (recall the distinction from Figure 1). Most existing work falls in the first category, which we discuss in Section 2.1. Our work falls in the second category, where little work has been done, as we discuss in Section 2.2. We also mention work related to the building blocks we use in our protocols in Section 2.3.

It is worth mentioning that our work on privacy-preserving classification is complementary to work on differential privacy in the machine learning community (see e.g. [CMS11]). Our work aims to hide each user’s input data to the classification phase, whereas differential privacy seeks to construct classifiers/models from sensitive user training data that leak a bounded amount of information about each individual in the training data set.

### 2.1 Privacy-preserving training

A set of techniques have been developed for privacy-preserving training algorithms such as Naïve Bayes [VKC08, WY04, ZW05], decision trees [BDMN05, LP00], linear discriminant classifiers [DHC04], and more general kernel methods [LLM06].

Grapel et al. [GLN12] show how to train several machine learning classifiers using a somewhat homomorphic encryption scheme. They focus on a few simple classifiers (e.g. the linear means classifier), and do not elaborate on more complex algorithms such as support vector machines. They also support private classification, but in a weaker security model where the client learns more about the model than just the final sign of the classification. Indeed, performing the final comparison with fully homomorphic encryption (FHE) alone is inefficient, a difficulty we overcome with an interactive setting.
2.2 Privacy-preserving classification

Little work has been done to address the general problem of privacy-preserving classification in practice; previous work focuses on a weaker security setting (in which the client learns the model) and/or only supports specific classifiers.

In Bos et al. [BLN13], a third party can compute medical prediction functions over the encrypted data of a patient using fully homomorphic encryption. In their setting, everyone (including the patient) knows the predictive model, and their algorithm hides only the input of the patient from the cloud. Our protocols, on the other hand, also hide the model from the patient. Their algorithms cannot be applied to our setting because they leak more information than just the bit of the prediction to the patient. Furthermore, our techniques are notably different; using FHE directly for our classifiers would result in significant overheads.

Barni et al. [BFK+09, BFL+09] construct secure evaluation of linear branching programs, which they use to implement a secure classifier of ECG signals. Their technique is based on finely-tuned garbled circuits. By comparison, our construction is not limited to branching programs (or decision trees), and our evaluation shows that our construction is twice as fast on branching programs. In a subsequent work [BFL+11], Barni et al. study secure classifiers based on neural networks, which is a generalization of the perceptron classifiers, and hence also covered by our work.

Other works [EFG+09, SSW09, AB06, AB07] construct specific face recognition or detection classifiers. We focus on providing a set of generic classifiers and building blocks to construct more complex classifiers. In Section 10.1.2, we show how to construct a private face detection classifier using the modularity of our techniques.

2.3 Work related to our building blocks

Two of the basic components we use are private comparison and private computation of dot products. These items have been well-studied previously; see [Yao82, DGK07, DGK09, Veu11, LT05, AB06, KSS09] for comparison techniques and [AD01, GLLM04, Kil05, AB06] for techniques to compute dot products. Section 4.1 discusses how we build on these tools.

3 Background and preliminaries

3.1 Classification in machine learning algorithms

The user’s input $x$ is a vector of $d$ elements $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, called a feature vector. To classify the input $x$ means to evaluate a classification function $C_w : \mathbb{R}^d \to \{c_1, \ldots, c_k\}$ on $x$. The output is $c_{k^*} = C_w(x)$, where $k^* \in \{1 \ldots k\}$; $c_{k^*}$ is the class to which $x$ corresponds, based on the model $w$. For ease of notation, we often write $k^*$ instead of $c_{k^*}$, namely $k^* = C_w(x)$.

We now describe how three popular classifiers work on regular, unencrypted data. These classifiers differ in the model $w$ and the function $C_w$. For more details, we refer the reader to [BN06].

Hyperplane decision-based classifiers. For this classifier, the model $w$ consists of $k$ vectors in $\mathbb{R}^d$ ($w = \{w_i\}_{i=1}^k$). The classifier is (cf. [BN06]):

$$k^* = \arg\max_{i \in [k]} \langle w_i, x \rangle, \quad (1)$$

where $\langle w_i, x \rangle$ denotes inner product between $w_i$ and $x$.

We now explain how Eq. (1) captures many common machine learning algorithms. A hyperplane based classifier typically works with a hypothesis space $\mathcal{H}$ equipped with an inner product $\langle \cdot, \cdot \rangle$. This classifier usually solves a binary classification problem ($k = 2$); given a user input $x$, $x$ is classified in class $c_2$ if $\langle w, \phi(x) \rangle \geq 0$, otherwise it is labeled as part of class $c_1$. Here, $\phi : \mathbb{R}^d \to \mathcal{H}$ denotes the feature mapping from $\mathbb{R}^d$ to $\mathcal{H}$ [BN06]. In this work, we focus on the case when $\mathcal{H} = \mathbb{R}^d$ and note that a large class of infinite dimensional spaces can be approximated with a finite dimensional space (as in [RR07]), including the popular gaussian kernel (RBF). In this case, $\phi(x) = x$ or $\phi(x) = P x$ for a randomized projection matrix $P$ chosen during training. Notice that $P x$ consists solely of inner products; we will show how to support private evaluation of inner products later, so for simplicity we drop $P$ from the discussion. To extend such a classifier from 2 classes to $k$ classes, we use one of the most common approaches, one-versus-all, where $k$ different models $\{w_i\}_{i=1}^k$ are trained to discriminate each class from all the others. The decision rule is then given by
Naïve Bayes classifiers. For this classifier, the model $w$ consists of various probabilities: the probability that each class $c_i$ occurs, namely $\{p(C = c_i)\}_{i=1}^k$, and the probabilities that an element $x_j$ of $x$ occurs in a certain class $c_i$. More concretely, the latter is the probability of the $j$-th component $x_j$ of $x$ to be $v$ when $x$ belongs to category $c_i$; this is denoted by $\{\{p(X_j = v|C = c_i)\}_{v \in D_j}\}_{j=1}^d$ $\{i=1\}_k$, where $D_j$ is $X_j$’s domain\(^2\). The classification function, using a maximum a posteriori decision rule, works by choosing the class with the highest posterior probability:

$$k^* = \arg\max_{i \in [k]} p(C = c_i | X = x)$$

$$= \arg\max_{i \in [k]} p(C = c_i, X = x)$$

$$= \arg\max_{i \in [k]} p(C = c_i, X_1 = x_1, \ldots, X_d = x_d)$$

where the second equality follows from applying Bayes’ rule (we omitted the normalizing factor $p(X = x)$ because it is the same for a fixed $x$).

The Naïve Bayes model assumes that $p(C = c_i, X = x)$ has the following factorization:

$$p(C = c_i, X_1 = x_1, \ldots, X_d = x_d)$$

$$= p(C = c_i) \prod_{j=1}^d p(X_j = x_j|C = c_i),$$

namely, each of the $d$ features are conditionally independent given the class. For simplicity, we assume that the domain of the features values (the $x_j$’s) is discrete and finite, so the $p(X_j = x_j|C = c_i)$’s are probability masses.

Decision trees. A decision tree is a non-parametric classifier which works by partitioning the feature vector space one attribute at a time; interior nodes in the tree correspond to partitioning rules, and leaf nodes correspond to class labels. A feature vector $x$ is classified by walking the tree starting from the root, using the partitioning rule at each node to decide which branch to take until a leaf node is encountered. The class at the leaf node is the result of the classification.

Figure 2 gives an example of a decision tree. The model consists of the structure of the tree and the decision criteria at each node (in this case the thresholds $w_1, \ldots, w_4$).

\(^2\)Be careful to distinguish between $X_j$, the probabilistic random variable representing the values taken by the $j$-th feature of user’s input, and $x_j$, the actual value taken by the specific vector $x$. 
3.2 Cryptographic preliminaries

3.2.1 Cryptosystems

In this work, we use three additively homomorphic cryptosystems. A public-key encryption scheme HE is additively
homomorphic if, given two encrypted messages $HE.\text{Enc}(a)$ and $HE.\text{Enc}(b)$, there exists a public-key operation $\oplus$
such that $HE.\text{Enc}(a) \oplus HE.\text{Enc}(b)$ is an encryption of $a + b$. We emphasize that these are homomorphic only for
addition, which makes them efficient, unlike fully homomorphic encryption [Gen09], which supports any function. The
cryptosystems we use are:

1. the QR (Quadratic Residuosity) cryptosystem of Goldwasser-Micali [GM82],
2. the Paillier cryptosystem [Pai99], and
3. a leveled fully homomorphic encryption (FHE) scheme, HELib [Hal13]

3.2.2 Cryptographic assumptions

We prove that our protocols are secure based on the semantic security [Gol04] of the above cryptosystems. These
cryptosystems rely on standard and well-studied computational assumptions: the Quadratic Residuosity assumption, the
Decisional Composite Residuosity assumption, and the Ring Learning With Error (RLWE) assumption.

3.2.3 Adversarial model

We prove security of our protocols using the secure two-party computation framework for passive adversaries (or honest-
but-curious [Gol04]) defined in Appendix B.1. To explain what a passive adversary is, at a high level, consider that a
party called party A is compromised by such an adversary. This adversary tries to learn as much private information
about the input of the other party by watching all the information party A receives; nevertheless, this adversary cannot
prevent party A from following the prescribed protocol faithfully (hence, it is not an active adversary).

To enable us to compose various protocols into a bigger protocol securely, we invoke modular sequential composition
(see Appendix B.2).

3.3 Notation

All our protocols are between two parties: parties $A$ and $B$ for our building blocks and parties $C$ (client) and $S$ (server)
for our classifiers.

Inputs and outputs of our building blocks are either unencrypted or encrypted with an additively homomorphic
encryption scheme. We use the following notation. The plaintext space of QR is $\mathbb{F}_2$ (bits), and we denote by $[b]$ a bit $b$
encrypted under QR; the plaintext space of Paillier is $\mathbb{Z}_N$ where $N$ is the public modulus of Paillier, and we denote by $\llbracket m \rrbracket$ an integer $m$ encrypted under Paillier. The plaintext space of the FHE scheme is $\mathbb{F}_2$. We denote by $SK_P$ and $PK_P$, a secret and a public key for Paillier, respectively. Also, we denote by $SK_{QR}$ and $PK_{QR}$, a secret and a public key for QR.

For a constant $b$, $a \leftarrow b$ means that $a$ is assigned the value of $b$. For a distribution $D$, $a \leftarrow D$ means that $a$ gets a
sample from $D$.

4 Building blocks

In this section, we develop a library of building blocks, which we later use to build our classifiers. We designed this
library to also enable constructing other classifiers than the ones described in our paper. The building blocks in this
section combine existing techniques with either new techniques or new optimizations.
Table 2: The API of our comparison protocol and its implementation. There are five types of comparisons each having a different setup.

### 4.1 Comparison

We now describe our comparison protocol. In order for this protocol to be used in a wide range of classifiers, its setup needs to be flexible: namely, it has to support a range of choices regarding which party gets the input, which party gets the output, and whether the input or output are encrypted or not. Table 2 shows the various ways our comparison protocol can be used. In each case, each party learns nothing else about the other party’s input other than what Table 2 indicates as the output.

We implemented each row of Table 2 by modifying existing protocols. We explain only the modifications here, and defer full protocol descriptions to Appendix A and proofs of security to Appendix C.1.

There are at least two approaches to performing comparison efficiently: using specialized homomorphic encryption [DGK07, DGK09, EFG09, Veu11], or using garbled circuits [BHKR13]. We compared empirically the performance of these approaches and concluded that the former is more efficient for comparison of encrypted values, and the second is more efficient for comparison of unencrypted values.

#### 4.1.1 Comparison with unencrypted inputs (Row 1)

To compare unencrypted inputs, we use garbled circuits implemented with the state-of-the-art garbling scheme of Bellare et al. [BHKR13], the short circuit for comparison of Kolesnikov et al. [KSS09] and a well-known oblivious transfer (OT) scheme due to Naor and Pinkas [NP01]. Since most of our other building blocks expect inputs encrypted with homomorphic encryption, one also needs to convert from a garbled output to homomorphic encryption to enable composition. We can implement this easily using the random shares technique in [KSS13].

The above techniques combined give us the desired comparison protocol. Actually, we can directly combine them to build an even more efficient protocol: we use an enhanced comparison circuit that also takes as input a masking bit. Using a garbled circuit and oblivious transfer, A will compute $(a < b) \oplus c$ where $c$ is a bit randomly chosen by B. B will also provide an encryption $[c]$ of $c$, enabling A to compute $[a < b]$ using the homomorphic properties of QR.

#### 4.1.2 Comparison with encrypted inputs (Rows 2, 3)

Our classifiers also require the ability to compare two encrypted inputs. More specifically, suppose that party A wants to compare two encrypted integers $a$ and $b$, but party B holds the decryption key. To implement this task, we slightly modify Veugen’s [Veu11] protocol: it uses a comparison with unencrypted inputs protocol as a sub-procedure, and we replaced it with the comparison protocol we just described above. This yields a protocol for the setup in Row 2. To ensure that A receives the plaintext output as in Row 3, B sends the encrypted result to A who decrypts it. Appendix A provides the detailed protocol.

#### 4.1.3 Reversed comparison over encrypted data (Row 4, 5)

In some cases, we want the result of the comparison to be held by the party that does not hold the encrypted data. For this, we modify Veugen’s protocol to reverse the outputs of party A and party B: we achieve this by exchanging the role of party A and party B in the last few steps of the protocol, after invoking the comparison protocol with unencrypted inputs. We do not present the details in the paper body because they are not insightful, and instead include them in Appendix A.
This results in a protocol whose specification is in Row 4. To obtain Row 5, A sends the encrypted result to B who can decrypt it.

### 4.1.4 Negative integers comparison and sign determination

Negative numbers are handled by the protocols above unchanged. Even though the Paillier plaintext size is “positive”, a negative number simply becomes a large number in the plaintext space due to cyclicity of the space. As long as the values encrypted are within a preset interval \((-2^\ell, 2^\ell)\) for some fixed \(\ell\), Veugen’s protocol and the above protocols work correctly.

In some cases, we need to compute the sign of an encrypted integer \([b]\). In this case, we simply compare to the encryption of 0.

### 4.2 argmax over encrypted data

In this scenario, party A has \(k\) values \(a_1, \ldots, a_k\) encrypted under party B’s secret key and wants party B to know the argmax over these values (the index of the largest value), but neither party should learn anything else. For example, if A has values \([1]\), \([100]\) and \([2]\), B should learn that the second is the largest value, but learn nothing else. In particular, B should not learn the order relations between the \(a_i\)'s.

Our protocol for argmax is shown in Protocol 1. We now provide intuition into the protocol and its security.

**Intuition.** Let’s start with a strawman. To prevent B from learning the order of the \(k\) values \(\{a_i\}_{i=1}^k\), A applies a random permutation \(\pi\). The \(i\)-th element becomes \([a_{\pi(i)}^i]\) instead of \([a_i]\).

Now, A and B compare the first two values \([a_1^1]\) and \([a_2^2]\) using the comparison protocol from row 4 of Table 2. B learns the index, \(m\), of the larger value, and tells A to compare \([a_m']^m\) to \([a_1^1]\) next. After iterating in this manner through all the \(k\) values, B determines the index \(m\) of the largest value. A can then compute \(\pi^{-1}(m)\) which represents the argmax in the original, unpermuted order.

Since A applied a random permutation \(\pi\), B does not learn the ordering of the values. The problem, though, is that A learns this ordering because, at every iteration, A knows the value of \(m\) up to that step and \(\pi\). One way to fix this problem is for B to compare every pair of inputs from A, but this would result in a quadratic number of comparisons, which is too slow.

Instead, our protocol preserves the linear number of comparisons from above. The idea is that, at each iteration, once B determines which is the maximum of the two values compared, B should randomize the encryption of this maximum in such a way that A cannot link this value to one of the values compared. B uses the Refresh procedure for the randomization of Paillier ciphertexts. In the case where the “refresher” knows the secret key, this can be seen as a decryption followed by a re-encryption. If not, it can be seen as a multiplication by an encryption of 0.

A difficulty is that, to randomize the encryption of the maximum \([a_m']^m\), B needs to get this encryption – however, B must not receive this encryption because B has the key SK\(\rho^*\) to decrypt it, which violates privacy. Instead, the idea is for A itself to add noise \(r_i\) and \(s_i\) to \([a_m']^m\), so decryption at \(B\) yields random values, then \(B\) refreshes the ciphertext, and then \(A\) removes the randomness \(r_i\) and \(s_i\) it added.

In the end, our protocol performs \(k - 1\) encrypted comparisons of \(l\) bits integers and \(7(k - 1)\) homomorphic operations (refreshes, multiplications and subtractions). In terms of round trips, we add \(k - 1\) roundtrips to the comparison protocol, one roundtrip per loop iteration.

**Proposition 4.1.** Protocol 1 is correct and secure in the honest-but-curious model.

**Proof intuition.** The correctness property is straightforward. Let’s argue security. A does not learn intermediary results in the computation because of the security of the comparison protocol and because she gets a refreshed ciphertext from \(B\) which \(A\) cannot couple to a previously seen ciphertext. \(B\) does learn the result of each comparison – however, since \(A\) applied a random permutation before the comparison, \(B\) learns no useful information. See Appendix C for a complete proof.
Protocol 1 argmax over encrypted data

Input A: $k$ encrypted integers $([a_1], \ldots, [a_k])$, the bit length $l$ of the $a_i$, and public keys $PK_{QR}$ and $PK_P$

Input B: Secret keys $SK_P$ and $SK_{QR}$, the bit length $l$

Output A: argmax $a_i$

1: A: chooses a random permutation $\pi$ over $\{1, \ldots, k\}$
2: A: $[\max] \leftarrow [a_{\pi(1)}]$
3: B: $m \leftarrow 1$
4: for $i = 2$ to $k$
5: Using the comparison protocol (Sec. 4.1.3), B gets the bit $b_i = (\max \leq a_{\pi(i)})$
6: A picks two random integers $r_i, s_i \leftarrow (0, 2^{l+1}) \cap \mathbb{Z}$
7: A: $[m'_i] \leftarrow [\max] \cdot [r_i]$ $\triangleright m'_i = \max + r_i$
8: A: $[a'_i] \leftarrow [a_{\pi(i)}] \cdot [s_i]$ $\triangleright a'_i = a_{\pi(i)} + s_i$
9: A sends $[m'_i]$ and $[a'_i]$ to B
10: if $b_i$ is true then
11: B: $m \leftarrow i$
12: B: $[v_i] \leftarrow \text{Refresh}[a'_i]$ $\triangleright v_i = a'_i$
13: else
14: B: $[v_i] \leftarrow \text{Refresh}[m'_i]$ $\triangleright v_i = m'_i$
15: end if
16: B sends to A $[v_i]$
17: B sends to A $[b_i]$
18: A: $[\max] \leftarrow [v_i] \cdot (g^{-1} \cdot [b_i])^{v_i} \cdot [b_i]^{-s_i}$ $\triangleright \max = v_i + (b_i - 1) \cdot r_i - b_i \cdot t_i$
20: end for
21: B sends $m$ to A
22: A outputs $\pi^{-1}(m)$

4.3 Changing the encryption scheme

To enable us to compose various building blocks, we developed a protocol for converting ciphertexts from one encryption scheme to another while maintaining the underlying plaintexts. We first present a protocol that switches between two encryption schemes with the same plaintext size (such as QR and FHE over bits), and then present a different protocol for switching from QR to Paillier.

Concretely, consider two additively homomorphic encryption schemes $E_1$ and $E_2$, both semantically secure with the same plaintext space $M$. Let $[\cdot ]_1$ be an encryption using $E_1$ and $[\cdot ]_2$ an encryption using $E_2$. Consider that party $B$ has the secret keys $SK_1$ and $SK_2$ for both schemes and $A$ has the corresponding public keys $PK_1$ and $PK_2$. Party $A$ also has a value encrypted with $PK_1$, $[c]_1$. Our protocol, protocol 2, enables $A$ to obtain an encryption of $c$ under $E_2$, $[c]_2$, without revealing anything to $B$ about $c$.

Protocol intuition. The idea is for $A$ to add a random noise $r$ to the ciphertext using the homomorphic property of $E_1$. Then $B$ decrypts the resulting value with $E_1$ (obtaining $x + r \in M$) and encrypts it with $E_2$, sends the result to $A$ which removes the randomness $r$ using the homomorphic property of $E_2$. Even though $B$ was able to decrypt $[c]_1$, $B$ obtains $x + r \in M$ which hides $x$ in an information-theoretic way (it is a one-time pad).

Note that, for some schemes, the plaintext space $M$ depends on the secret keys. In this case, we must be sure that party $A$ can still choose uniformly elements of $M$ without knowing it. For example, for Paillier, $M = \mathbb{Z}_N \approx \mathbb{Z}_p \times \mathbb{Z}_q$ where $p$ and $q$ are the private primes. However, in this case, $A$ can sample noise in $\mathbb{Z}_N$ that will not be in $\mathbb{Z}_N$ with negligible probability $(1 - \frac{1}{p})(1 - \frac{1}{q}) \approx 1 - \frac{2}{\sqrt{N}}$ (remember $N$ is large – 1024 bits in our instantiation).

Proposition 4.2. Protocol 2 is secure in the honest-but-curious model.

In our classifiers, we use this protocol for $M = \{0, 1\}$ and the encryption schemes are QR (for $E_1$) and an FHE scheme over bits (for $E_2$). In some cases, we might also want to switch from QR to Paillier (e.g. reuse the encrypted
Protocol 2 Changing the encryption scheme

Input A: \([e]_1\) and public keys \(PK_1\) and \(PK_2\)
Input B: Secret keys \(SK_1\) and \(SK_2\)
Output A: \([e]_2\)

1: A uniformly picks \(r \leftarrow M\)
2: A sends \([e']_1 \leftarrow [e]_1 \cdot [r]_1\) to B
3: B decrypts \(c'\) and re-encrypts with \(E_2\)
4: B sends \([e']_2\) to A
5: A: \([e]_2 = [e']_2 \cdot [r]_2^{-1}\)
6: A outputs \([e]_2\)

result of a comparison in a homomorphic computation), which has a different message space. Note that we can simulate the homomorphic XOR operation and a message space \(M = \{0, 1\}\) with Paillier: we can easily compute the encryption of \(b_1 \oplus b_2\) under Paillier when at most one of the \(b_i\) is encrypted (which we explain in the next subsection). This is the case in our setting because party A has the randomness \(r\) in the clear.

### 4.3.1 XOR with Paillier.

Suppose a party gets the bit \(b_1\) encrypted under Paillier’s encryption scheme, and that this party only has the public key. This party knows the bit \(b_2\) in the clear and wants to compute the encryption of \([b_1 \oplus b_2]\).

To do so, we just have to notice that

\[
b_1 \oplus b_2 = \begin{cases} 
b_1 & \text{if } b_2 = 0 \\
1 - b_1 & \text{if } b_2 = 1 \end{cases}
\]

Hence, it is very easy to compute an encryption of \(b_1 \oplus b_2\) if we know the modulus \(N\) and the generator \(g\) (cf. Paillier’s scheme construction):

\[
[b_1 \oplus b_2] = \begin{cases} 
[b_1] & \text{if } b_2 = 0 \\
g[b_1]^{-1} \mod N^2 & \text{if } b_2 = 1 
\end{cases}
\]

If we want to unveil the result to an adversary who knows the original encryption of \(b_1\) (but not the secret key), we have to refresh the result of the previous function to ensure semantic security.

### 4.4 Computing dot products

For completeness, we include a straightforward algorithm for computing dot products of two vectors, which relies on Paillier’s homomorphic property.

Protocol 3 Private dot product

Input A: \(x = (x_1, \ldots, x_d) \in \mathbb{Z}^d\), public key \(PK_P\)
Input B: \(y = (y_1, \ldots, y_d) \in \mathbb{Z}^d\), secret key \(SK_P\)
Output A: \(\langle x, y \rangle\)

1: B encrypts \(y_1, \ldots, y_d\) and sends the encryptions \([y_i]\) to A
2: A computes \(v = \prod [y_i]^{x_i} \mod N^2\)
3: A re-randomizes and outputs \([v]\)

Proposition 4.3. Protocol 3 is secure in the honest-but-curious model.
4.5 Dealing with floating point numbers

Although all our protocols manipulate integers, classifiers usually use floating point numbers. Hence, when developing classifiers with our protocol library, we must adapt our protocols accordingly.

Fortunately, most of the operations involved are either additions or multiplications. As a consequence, a simple solution is to multiply each floating point value by a constant $K$ (e.g. $K = 2^{52}$ for IEEE 754 doubles) and thus support finite precision. We must also consider the bit length for the comparisons. We show an example of a full analysis in Section 6 for the Naïve Bayes classifier.

5 Private hyperplane decision

Recall from Section 3.1 that this classifier computes

$$k^* = \arg\max_{i \in [k]} \langle w_i, x \rangle.$$  

Now that we constructed our library of building blocks, it is straightforward to implement this classifier securely: the client computes the encryption of $\langle (w_i, x) \rangle$ for all $i \in [k]$ using the dot product protocol and then applies the $\arg\max$ protocol (Protocol 1) to the encrypted dot products.

**Protocol 4 Private hyperplane decision**

| Client’s (C) Input: | $x = (x_1, \ldots, x_d) \in \mathbb{Z}^d$, public keys $PK_P$ and $PK_{QR}$ |
| Server’s (S) Input: | $\{ w_i \}_{i=1}^k$ where $\forall i \in [k], w_i \in \mathbb{Z}^n$, secret keys $SK_P$ and $SK_{QR}$ |
| Client’s Output:   | $\arg\max_{i \in [k]} \langle w_i, x \rangle$ |

1. for $i = 1$ to $k$ do
2. C and S run Protocol 3 for private dot product where C is party A with input $x$ and S is party B with input $w_i$.
3. C gets $\langle v_i \rangle$ the result of the protocol.
   \(\triangleright v_i \leftarrow \langle x, w_i \rangle\)
4. C and S run Protocol 1 for argmax where C is the A, and S the B, and $\langle v_1 \rangle, \ldots, \langle v_k \rangle$ the input ciphertexts. C gets the result $i_0$ of the protocol.
   \(\triangleright i_0 \leftarrow \arg\max_{i \in [k]} v_i\)
5. C outputs $i_0$

**Proposition 5.1.** Protocol 4 is secure in the honest-but-curious model.

6 Secure Naïve Bayes classifier

Section 3.1 describes the Naïve Bayes classifier. The goal is for the client to learn $k^*$ without learning anything about the probabilities that constitute the model, and the server should learn nothing about $x$. Recall that the features values domain is discrete and finite.

As is typically done for numerical stability reasons, we work with the logarithm of the probability distributions:

$$k^* = \arg\max_{i \in [k]} \log p(C = c_i | X = x)$$

$$= \arg\max_{i \in [k]} \left\{ \log p(C = c_i) + \sum_{j=1}^{d} \log p(X_j = x_j | C = c_i) \right\}$$  

(2)
6.1 Preparing the model

Since the Paillier encryption scheme works with integers, we convert each log of a probability from above to an integer by multiplying it with a large number $K$ (recall that the plaintext space of Paillier is large $\approx 2^{1024}$ thus allowing for a large $K$), thus still maintaining high accuracy. The issues due to using integers for bayesian classification have been previously studied in [TRMP12], even though their setting was even more restricting than ours. However, they use a similar idea to ours: shifting the probabilities logarithms and use fixed point representation.

As the only operations used in the classification step are additions and comparisons (cf. Equation (2)), we can just multiply the conditional probabilities $p(x_j|c_i)$ by a constant $K$ so to get integers everywhere, while keeping the same classification result.

For example, if we are able to compute the conditional probabilities using IEEE 754 double precision floating point numbers, with 52 bits of precision, then we can represent every probability $p$ as

$$p = m \cdot 2^e$$

where $m$ binary representation is $(m)_{2} = 1.d$ and $d$ is a 52 bits integer. Hence we have $1 \leq m < 2$ and we can rewrite $m$ as

$$m = \frac{m'}{2^{52}}$$

We are using this representation to find a constant $K$ such that $K \cdot v_i \in \mathbb{N}$ for all $i$. As seen before, we can write the $v_i$’s as

$$v_i = m'_i \cdot 2^{e_i - 52}$$

Let $e^* = \min_i e_i$, and $\delta_i = e_i - e^* \geq 0$. Then,

$$v_i = m'_i \cdot 2^{\delta_i} \cdot 2^{e^* - 52}$$

So let $K = 2^{52-e^*}$. We have $K \cdot v_i = m'_i \cdot 2^{\delta_i} \in \mathbb{N}$. An important thing to notice is that the $v_i$’s can be very large integers (due to $\delta_i$), and this might cause overflows errors. However, remember that we are doing all this to store logarithms of probabilities in Paillier cyphertexts, and as Paillier plaintext space is very large (more than 1024 bits in our setting) and $\delta_i$’s remain small.

Also notice that this shifting procedure can be done without any loss of precision as we can directly work with the bit representation of the floating points numbers.

Finally, we must also ensure that we do not overflow Paillier’s message space when doing all the operations (homomorphic additions, comparisons, . . . ). If – as before – $d$ is the number of features, the maximum number of bits when doing the computations will be $l_{\text{max}} = d + 1 + (52 + \delta^*)$ where $\delta^* = \max \delta_i$: we have to add the probabilities for the $d$ features and the probability of the class label (the $d + 1$ term), and each probability is encoded using $(52 + \delta^*)$ bits. Hence, the value $l$ used for the comparison protocols must be chosen larger than $l_{\text{max}}$.

Hence, we must ensure that $\log_2 N > l_{\text{max}} + 1 + \lambda$ where $\lambda$ is the security parameter and $N$ is the modulus for Paillier’s cryptosystem plaintext space (cf. Section 4.1.2). This condition is easily fulfilled as, for a good level of security, we have to take $\log_2 N \geq 1024$ and we usually take $\lambda \approx 100$.

Let $D_j$ be the domain of possible values of $x_j$ (the $j$-th attribute of the feature vector $x$). The server prepares $kd + 1$ tables as part of the model, where $K$ is computed as described just before:

- One table for the priors on the classes $P$: $P(i) = [K \log p(C = c_i)]$.
- One table per feature $j$ per class $i$, $T_{i,j}$: $T_{i,j}(v) \approx [K \log p(X_j = v|C = c_i)]$, for all $v \in D_j$.

The tables remain small: $P$ has one entry by category i.e. $k$ entries total, and $T$ has one entry by category and feature value i.e. $k \cdot D$ entries where $D = \sum |D_j|$. In our examples, this represents less than 3600 entries. Moreover, this preparation step can be done once and for all at server startup, and is hence amortized.

\footnote{If the biggest $\delta_i$ is 10, the ratio between the smallest and the biggest probability is of order $2^{10} = 2^{1024}$ ...}
Let us begin with some intuition. The server encrypts each entry in these tables with Paillier and gives the resulting encryption (the encrypted model) to the client. For every class \( c_i \), the client uses Paillier’s additive homomorphism to compute \( [p_i] = [P(i)] \prod_{j=1}^{d} [T_{i,j}(x_j)] \). Finally, the client runs the argmax protocol, Protocol 1, to get \( \text{argmax}_i p_i \). For completeness, the protocol is shown in Protocol 5.

### Protocol 5 Naïve Bayes Classifier

**Client’s (C) Input:** \( x = (x_1, \ldots, x_d) \in \mathbb{Z}^d \), public key \( \text{PK}_P \), secret key \( \text{SK}_{QR} \)

**Server’s (S) Input:** The secret key \( \text{SK}_P \), public key \( \text{PK}_{QR} \) and probability tables \( \{ \log p(C = c_i) \}_{1 \leq i \leq k} \) and \( \{ \log p(X_j = v|C = c_i) \}_{v \in D_j} \) \( 1 \leq j \leq d, 1 \leq i \leq k \)

**Client’s Output:** \( i_0 \) such that \( p(x, c_{i_0}) \) is maximum

1. The server prepares the tables \( P \) and \( \{T_{i,j}\}_{1 \leq i \leq k, 1 \leq j \leq d} \) and encrypts their entries using Paillier.
2. The server sends \( [P] \) and \( \{[T_{i,j}]\}_{i,j} \) to the client.
3. For all \( 1 \leq i \leq k \), the client computes \( [p_i] = [P(i)] \prod_{j=1}^{d} [T_{i,j}(x_j)] \).
4. The client runs the argmax protocol (Protocol 1) with the server and gets \( i_0 = \text{argmax}_i p_i \)
5. C outputs \( i_0 \)

### Proposition 6.1

Protocol 5 is secure in the honest-but-curious model.

**Proof intuition.** Given the security property of the argmax protocol, Protocol 1, and the semantic security of the Paillier cryptosystem, the security of this classifier follows trivially, by invoking a modular composition theorem.

**Efficiency.** Note that the tables \( P \) and \( \{T_{i,j}\}_{1 \leq i \leq k, 1 \leq j \leq d} \) can be prepared in advance. Hence the cost of constructing the tables can be amortized over many uses. To compute the encrypted probabilities \( p_i \)’s, the client runs \( d \) homomorphic operations (here multiplications) for each \( i \), hence doing \( kd \) modular multiplications. Then the parties run a single argmax protocol i.e. \( k - 1 \) comparisons and \( O(k) \) homomorphic operations. Thus, compared to non-encrypted computation, the overhead comes only from the use of homomorphic encryption operations instead of plaintext operations. Regarding the number of round trips, these are due to the argmax protocol: \( k - 1 \) runs of the comparison protocol and \( k - 1 \) additional roundtrips.

### 7 Private decision trees

A private decision tree classifier allows the server to traverse a binary decision tree using the client’s input \( x \) such that the server does not learn the input \( x \), and the client does not learn the structure of the tree and the thresholds at each node. A challenge is that, in particular, the client should not learn the path in the tree that corresponds to \( x \) – the position of the path in the tree and the length of the path leaks information about the model. The outcome of the classification does not necessarily leak the path in the tree.

The idea is to express the decision tree as a polynomial \( P \) whose output is the result of the classification, the class predicted for \( x \). Then, the server and the client privately compute inputs to this polynomial based on \( x \) and the thresholds \( w_t \). Finally, the server evaluates the polynomial \( P \) privately.

#### 7.1 Polynomial form of a decision tree

Consider that each node of the tree has a boolean variable associated to it. The value of the boolean at a node is 1 if, on input \( x \), one should follow the right branch, and 0 otherwise. For example, denote the boolean variable at the root of the tree by \( b_1 \). The value of \( b_1 \) is 1 if \( x_1 \leq w_1 \) (recall Figure 2), and 0 otherwise.

We construct a polynomial \( P \) that, on input all these boolean variables and the value of each class at a leaf node, outputs the class predicted for \( x \). The idea is that \( P \) is a sum of terms, where each term (say \( t \)) corresponds
to a path in the tree from root to a leaf node (say \(c\)). A term \(t\) evaluates to \(c\) iff \(x\) is classified along that path in \(T\), else it evaluates to zero. Hence, the term corresponding to a path in the tree is naturally the multiplication of the boolean variables on that path and the class at the leaf node. For example, for the tree in Figure 3, \(P\) is

\[
P(b_1, b_2, b_3, b_4, c_1, \ldots, c_5) = b_1 (b_3 \cdot (b_4 \cdot c_5 + (1 - b_4) \cdot c_4) + (1 - b_3) \cdot c_3) + (1 - b_1)(b_2 \cdot c_2 + (1 - b_2) \cdot c_1).
\]

![Figure 3: Decision tree with booleans](image)

7.2 Private evaluation of a polynomial

Let us first explain how to compute the values of the boolean variables securely. Let \(n\) be the number of nodes in the tree and \(n_{\text{leaves}}\) be the number of leaves in the tree. These values must remain unknown to the server because they leak information about \(x\): they are the result of the intermediate computations of the classification criterion. For each boolean variable \(b_i\), the server and the client engage in the comparison protocol to compare \(w_i\) and the corresponding attribute of \(x\). As a result, the server obtains \([b_i]\) for \(i \in 1 \ldots n\); the server then changes the encryption of these values to FHE using Protocol 2, thus obtaining \([J_{b_i}K]\).

The server evaluates \(P\) on \(([b_1], \ldots, [b_n])\) using the homomorphic properties of FHE. In most cases, FHE evaluation is very slow, but we succeed to make it efficient through a combination of techniques we now discuss. To understand these techniques, recall that a typical FHE evaluation happens over a circuit whose gates are modular addition and multiplication. The performance of FHE depends a lot on the depth of multiplications in this circuit.

First, we use a leveled FHE scheme: a scheme that supports only an a priori fixed multiplicative depth instead of an arbitrary such depth. As long as this depth is small, such a scheme is much faster than a full FHE scheme.

Second, we ensure that the multiplicative depth is very small using a tree-based evaluation. If \(h_{\text{max}}\) is the maximum height of the decision tree, then \(P\) has a term \(a_1 \cdot \ldots \cdot a_{h_{\text{max}}}\). If we evaluate this term naïvely with FHE, we multiply these values sequentially. This yields a multiplicative depth of \(h_{\text{max}}\), which makes FHE slow for common \(h_{\text{max}}\) values. Instead, we construct a binary tree over these values and multiply them in pairs based on the structure of this tree. This results in a multiplicative depth of \(\log_2 h_{\text{max}}\) (e.g., 4), which makes FHE evaluation significantly more efficient.
Finally, we use $\mathbb{F}_2$ as the plaintext space and SIMD slots for parallelism. FHE schemes are significantly faster when the values encrypted are bits (namely, in $\mathbb{F}_2$); however, $P$ contains classes (e.g., $c_1$) which are usually more than a bit in length. To enable computing $P$ over $\mathbb{F}_2$, we represent each class in binary. Let $l = \lceil \log_2 k \rceil$ ($k$ is the number of classes) be the number of bits needed to represent a class. We evaluate $P$ $l$ times, once for each of the $l$ bits of a class. Concretely, the $j$-th evaluation of $P$ takes as input $b_1, \ldots, b_n$ and for each leaf node $c_j$, its $j$-th bit $c_{ij}$. The result is $P(b_1, \ldots, b_n, c_{1j}, c_{2j}, \ldots, c_{n_{\text{leaves}j}})$, which represents the $j$-th bit of the outcome class. Hence, we need to run the FHE evaluation $l$ times.

To avoid this factor of $l$, the idea is to use a nice feature of FHE called SIMD slots (as described in [SV11]): these allow encrypting multiple bits in a single ciphertext such that any operation applied to the ciphertext gets applied in parallel to each of the bits. Hence, for each class $c_j$, the server creates an FHE ciphertext $\llbracket c_{j0}, \ldots, c_{j(l-1)} \rrbracket$. For each node $b_i$, it creates an FHE ciphertext $\llbracket b_{i1}, \ldots, b_{il} \rrbracket$ by simply repeating the $b_i$ value in each slot. Then, the server runs one FHE evaluation of $P$ over all these ciphertexts and obtains $\llbracket c_{o0}, \ldots, c_{ol-1} \rrbracket$ where $c_o$ is the outcome class. Hence, instead of $l$ FHE evaluations, the server runs the evaluation only once. This results in a performance improvement of $\log k$, a factor of 2 and more in our experiments. We were able to apply SIMD slots parallelism due to the fortunate fact that the same polynomial $P$ had to be computed for each slot.

Finally, evaluating the decision tree is done using $2n$ FHE multiplications and $2n$ FHE additions where $n$ is the number of criteria. The evaluation circuit has multiplication depth $\lceil \log_2(n+1) \rceil$.

### 7.3 Formal description

Protocol 6 describes the resulting protocol.

**Protocol 6 Decision Tree Classifier**

**Client’s (C) Input:** $x = (x_1, \ldots, x_n) \in \mathbb{Z}^n$, secret keys $SK_{QR}, SK_{FHE}$

**Server’s (S) Input:** The public keys $PK_{QR}, PK_{FHE}$, the model as a decision tree, including the $n$ thresholds $\{w_i\}_{i=1}^n$.

**Client’s Output:** The value of the leaf of the decision tree associated with the inputs $b_1, \ldots, b_n$.

1. S produces an $n$-variate polynomial $P$ as described in section 7.1.
2. S and C interact in the comparison protocol, so that S obtains $[b_i]$ for $i \in [1 \ldots n]$ by comparing $w_i$ to the corresponding attribute of $x$.
3. Using Protocol 2, S changes the encryption from QR to FHE and obtains $\llbracket b_1 \rrbracket, \ldots, \llbracket b_n \rrbracket$.
4. To evaluate $P$, S encrypts the bits of each category $c_i$ using FHE and SIMD slots, obtaining $\llbracket c_{i1}, \ldots, c_{il} \rrbracket$. S uses SIMD slots to compute homomorphically $\llbracket P(b_1, \ldots, b_n, c_{10}, \ldots, c_{n_{\text{leaves}0}}, \ldots, P(b_1, \ldots, b_n, c_{1l-1}, \ldots, c_{n_{\text{leaves}l-1}}) \rrbracket$. It rerandomizes the resulting ciphertext using FHE’s rerandomization function, and sends the result to the client.
5. C decrypts the result as the bit vector $(v_0, \ldots, v_{l-1})$ and outputs $\sum_{i=0}^{l-1} v_i \cdot 2^i$.

**Proposition 7.1.** Protocol 6 is secure in the honest-but-curious model.

**Proof intuition.** The proof is in Appendix C, but we give some intuition here. During the comparison protocol, the server only learns encrypted bits, so it learns nothing about $x$. During FHE evaluation, it similarly learns nothing about the input due to the security of FHE. The client does not learn the structure of the tree because the server performs the evaluation of the polynomial. Similarly, the client does not learn the bits at the nodes in the tree because of the security of the comparison protocol.

The interactions between the client and the server are due to the comparisons almost exclusively: the decision tree evaluation does not need any interaction but sending the encrypted result of the evaluation.
bool Linear_Classifier_Client::run()
{
    exchange_keys();

    // values_ is a vector of integers
    // compute the dot product
    mpz_class v = compute_dot_product(values_);
    mpz_class w = 1; // encryption of 0

    // compare the dot product with 0
    return enc_comparison(v, w, bit_size_, false);
}

void Linear_Classifier_Server_session::run_session()
{
    exchange_keys();

    // enc_model_ is the encrypted model vector
    // compute the dot product
    help_compute_dot_product(enc_model_, true);

    // help the client to get
    // the sign of the dot product
    help_enc_comparison(bit_size_, false);
}

Figure 4: Implementation example: a linear classifier

<table>
<thead>
<tr>
<th>Bit size</th>
<th>A Computation</th>
<th>B Computation</th>
<th>Total Time</th>
<th>Communication</th>
<th>Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>14.11 ms</td>
<td>8.39 ms</td>
<td>105.5 ms</td>
<td>4.60 kB</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>18.29 ms</td>
<td>14.1 ms</td>
<td>117.5 ms</td>
<td>8.82 kB</td>
<td>3</td>
</tr>
<tr>
<td>32</td>
<td>22.9 ms</td>
<td>18.8 ms</td>
<td>122.6 ms</td>
<td>13.89 kB</td>
<td>3</td>
</tr>
<tr>
<td>64</td>
<td>34.7 ms</td>
<td>32.6 ms</td>
<td>134.5 ms</td>
<td>27.38 kB</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 3: Comparison with unencrypted input protocols evaluation.

8 Combining classifiers with AdaBoost

AdaBoost is a technique introduced in [FS97]. The idea is to combine a set of weak classifiers $h_i(x) : \mathbb{R}^d \rightarrow \{-1, +1\}$ to obtain a better classifier. The AdaBoost algorithm chooses $t$ scalars $\{\alpha_i\}_{i=1}^t$ and constructs a strong classifier as:

$$H(x) = \text{sign} \left( \sum_{i=1}^{t} \alpha_i h_i(x) \right)$$

If each of the $h_i(x)$’s is an instance of a classifier supported by our protocols, then given the scalars $\alpha_i$, we can easily and securely evaluate $H(x)$ by simply composing our building blocks. First, we run the secure protocols for each of $h_i$, except that the server keeps the intermediate result, the outcome of $h_i(x)$, encrypted using one of our comparison protocols (Rows 2 or 4 of Table 2). Second, if necessary, we convert them to Paillier’s encryption scheme with Protocol 2, and combine these intermediate results using Paillier’s additive homomorphic property as in the dot product protocol Protocol 3. Finally, we run the comparison over encrypted data algorithm to compare the result so far with zero, so that the client gets the final result.

9 Implementation

We have implemented the protocols and the classifiers in C++ using GMP4, Boost, Google’s Protocol Buffers5, and HElib [Hal13] for the FHE implementation.

The code is written in a modular way: all the elementary protocols defined in Section 4 can be used as black boxes with minimal developer effort. Thus, writing secure classifiers comes down to invoking the right API calls to the protocols. For example, for the linear classifier, the client simply calls a key exchange protocol to setup the various keys, followed by the dot product protocol, and then the comparison of encrypted data protocol to output the result, as shown in Figure 4.

4http://gmplib.org/
5https://code.google.com/p/protobuf/
### 10 Evaluation

To evaluate our work, we answer the following questions: (i) can our building blocks be used to construct other classifiers in a modular way (Section 10.1), (ii) what is the performance overhead of our building blocks (Section 10.3), and (iii) what is the performance overhead of our classifiers (Section 10.4)?

#### 10.1 Using our building blocks library

Here we demonstrate that our building blocks library can be used to build other classifiers modularly and that it is a useful contribution by itself. We will construct a multiplexer and a face detector. A face detection algorithm over encrypted data already exists [AB06, AB07], so our construction here is not the first such construction, but it serves as a proof of functionality for our library.

##### 10.1.1 Building a multiplexer classifier

A multiplexer is the following generalized comparison function:

\[ f_{\alpha,\beta}(a, b) = \begin{cases} 
\alpha & \text{if } a > b \\
\beta & \text{otherwise}
\end{cases} \]

We can express \( f_{\alpha,\beta} \) as a linear combination of the bit \( d = (a \leq b) \):

\[ f_{\alpha,\beta}(d) = d \cdot \beta + (1 - d) \cdot \alpha = \alpha + d \cdot (\beta - \alpha). \]

To implement this classifier privately, we compute \([d]\) by comparing \(a\) and \(b\), keeping the result encrypted with QR, and then changing the encryption scheme (cf. Section 4.3) to Paillier.

Then, using Paillier’s homomorphism and knowledge of \(\alpha\) and \(\beta\), we can compute an encryption of \( f_{\alpha,\beta}(d) \):

\[ \llbracket f_{\alpha,\beta}(d) \rrbracket = \llbracket \alpha \rrbracket \cdot \llbracket d \rrbracket^{\beta - \alpha}. \]

##### 10.1.2 Viola and Jones face detection

The Viola and Jones face detection algorithm [VJ01] is a particular case of an AdaBoost classifier. Denote by \(X\) an image represented as an integer vector and \(x\) a particular detection window (a subset of \(X\)’s coefficients). The strong classifier \(H\) for this particular detection window is

\[ H(x) = \text{sign} \left( \sum_{i=1}^{t} \alpha_i h_i(x) \right) \]

where the \(h_i\) are weak classifiers of the form \(h_i(x) = \text{sign} \left( \langle x, y_i \rangle - \theta_i \right) \).
Data set | Model size | Computation | Time per protocol | Total running time | Comm. | Interactions
---|---|---|---|---|---|---
Breast cancer (2) | 30 | 46.4 ms | 194 ms | 204 ms | 35.84 kB | 7
Credit (3) | 47 | 55.5 ms | 194 ms | 217 ms | 40.19 kB | 7

(a) Linear Classifier. Time per protocol includes communication.

Data set | Specs. | Computation | Time per protocol | FHE | Comm. | Interactions
---|---|---|---|---|---|---
Breast Cancer (1) | 2 | 150 ms | 82.9 ms | 479 ms | 72.47 kB | 14
Nursery (5) | 5 | 537 ms | 82.8 ms | 1415 ms | 150.7 kB | 42
Audiology (4) | 24 | 1652 ms | 431 ms | 3810 ms | 1911 kB | 166

(b) Naïve Bayes Classifier. \( C \) is the number of classes and \( F \) is the number of features. The Prob. Comp. column corresponds to the computation of the probabilities \( p(c_i|x) \) (cf. Section 6). Time per protocol includes communication.

Data set | Tree Specs. | Computation | Time per protocol | FHE | Comm. | Interactions
---|---|---|---|---|---|---
Nursery (5) | 4 | 1579 ms | 446 ms | 239 ms | 2639 kB | 30
ECG (6) | 6 | 2297 ms | 1410 ms | 899 ms | 3555 kB | 44

(c) Decision Tree Classifier. ES change indicates the time to run the protocol for changing encryption schemes. \( N \) is the number of nodes of the tree and \( D \) is its depth. Time per protocol includes communication.

Table 6: Classifiers evaluation.

In our setting, Alice owns the image and Bob the classifier (e.g. the vectors \( \{y_i\} \) and the scalars \( \{\theta_i\} \) and \( \{\alpha_i\} \)). Neither of them wants to disclose their input to the other party. Thanks to our building blocks, Alice can run Bob’s classifier on her image without her learning anything about the parameters and Bob learning any information about her image.

The weak classifiers can be seen as multiplexers; with the above notation, we have \( h_t(x) = f_1, -1(\langle x, y_t \rangle - \theta_t) \).

Using the elements of Section 10.1.1, we can easily compute the encrypted evaluation of every one of these weak classifiers under Paillier, and then, as described in Section 8, compute the encryption of \( H(x) \).

10.2 Performance evaluation setup

Our performance evaluations were run using two desktop computers each with identical configuration: two Intel Core i7 (64 bit) processors for a total 4 cores running at 2.66 GHz and 8 GB RAM. Since the machines were on the same network, we inflated the roundtrip time for a packet to be 40 ms to mimic real network latency. We used 1024-bit cryptographic keys, and chose the statistical security parameter \( \lambda \) to be 100. When using HElib, we use 80 bits of security, which corresponds to a 1024-bit asymmetric key.

10.3 Building blocks performance

We examine performance in terms of computation time at the client and server, communication bandwidth, and also number of interactions (round trips). We can see that all these protocols are efficient, with a runtime on the order of milliseconds.

10.3.1 Comparison protocols

Comparison with unencrypted input. Table 3 gives the running time of the comparison protocol with unencrypted input for various input size.
Comparison with encrypted input. Table 4 presents the performance of the comparison with encrypted inputs protocols.

### 10.3.2 argmax

Figure 5 presents the running times and the communication overhead of the argmax of encrypted data protocol (cf. Section 4.2). The input integers were 64 bit integers.

![Figure 5: Argmax of encrypted data protocol evaluation. The bars represent the execution of the protocol when the comparisons are executed one after each other, linearly. The line represents the execution when comparisons are executed in parallel, tree-wise.](image)

**10.3.3 Consequences of the latency on performances**

It is worth noticing that for most blocks, most of the running time is spend communicating: the network’s latency has a huge influence on the performances of the protocols (running time almost linear in the latency for some protocols). To improve the performances of a classifier implemented with our blocks, we might want to run several instances of some building blocks in parallel. This is actually what we did with the tree-based implementation of the argmax protocol, greatly improving the performances of the protocol (cf. Figure 5).

### 10.4 Classifier performance

Here we evaluate each of the classifiers described in Sections 5–7. The models are trained non-privately using scikit-learn\(^6\). We used the following datasets from the UCI machine learning repository [BL13]:

1. the Wisconsin Diagnostic Breast Cancer data set,
2. the Wisconsin Breast Cancer (Original) data set, a simplified version of the previous dataset,
3. Credit Approval data set,
4. Audiology (Standardized) data set,
5. Nursery data set, and

\(^6\)http://scikit-learn.org
6. ECG (electrocardiogram) classification data from Barni et al. [BFK+09]

   These data sets are scenarios when we want to ensure privacy of the server’s model and client’s input.
   Based on the suitability of each classifier, we used data sets 2 and 3 to test the hyperplane decision classifier, sets 1, 4 and 5 for the Naïve Bayes classifier, and sets 5 and 6 for the decision tree classifier.

   Table 6 shows the performance results. Our classifiers run in at most a few seconds, which we believe to be practical for sensitive applications. Note that even if the datasets become very large, the size of the model stays the same – the dataset size only affects the training phase which happens on unencrypted data before one uses our classifiers. Hence, the cost of our classification will be the same even for very large data sets.

   For the decision tree classifier, we compared our construction to Barni et al. [BFK+09] on the ECG dataset (by turning their branching program into a decision tree). Their performance is 2609 ms for the client and 6260 ms for the server with communication cost of 112.2KB. Even though their evaluation does not consider the communication delays, we are still more than three times as fast for the server and faster for the client.

10.5 Comparison to generic two-party tools

A set of generic secure two- or multi-party computation tools have been developed, such as TASTY [HKS+10] and Fairplay [MNPS04, BDNP08]. These support general functions, which include our classifiers.

   However, they are prohibitively slow for our specific setting. Our efficiency comes from specializing to classification functionality. To demonstrate their performance, we attempted to evaluate the Naïve Bayes classifier with these. We used FairplayMP to generate the circuit for this classifier and then TASTY to run the private computation on the circuit thus obtained. We tried to run the smallest Naïve Bayes instance, the Nursery dataset from our evaluation, which has only 3 possible values for each feature, but we ran out of memory during the circuit generation phase on a powerful machine with 256GB of RAM.

   Hence, we had to reduce the classification problem to only 3 classes (versus 5). Then, the circuit generation took more than 2 hours with FairplayMP, and the time to run the classification with TASTY was 413196 msec (with no network delay), which is ≈ 500 times slower than our performance (on the non-reduced classification problem with 5 classes). Thus, our specialized protocols improve performance by orders of magnitude.

11 Conclusion

In this paper, we constructed three major privacy-preserving classifiers as well as provided a library of building blocks that enables constructing other classifiers. We demonstrated the efficiency of our classifiers and library on real datasets.

Acknowledgment

We thank Thijs Veugen, Thomas Schneider, and the anonymous reviewers for their helpful comments.

---

7In Barni et al. [BFK+09], the evaluation was run over two 3GHz computers directly connected via Gigabit Ethernet. We scaled the given results by $\frac{3}{2}$ to get a better comparison basis.
References


A Comparison protocols

A.1 Comparison with unencrypted inputs

Our protocol for comparing with encrypted inputs is Protocol 7 and here is some intuition. We follow the main idea from Veugen [Veu11] (found in Section 2.1): compute $2^l + b - a$ (over encrypted data) and check the $l + 1$-th bit (the bit corresponding to the power $2^l$). If it is 1, it means that $b \geq a$, else $b < a$.

We also assume that the encryption scheme is additively homomorphic. In [Veu11] (Section 2.1), Veugen presents a solution for a similar problem except that A only gets the encrypted bit, not in the clear. So we modify his protocol in Protocol 7.

In the description of protocol 7, $N$ is the modulus associated with Paillier’s cryptosystem.

**Protocol 7** Comparing encrypted data

<table>
<thead>
<tr>
<th>Input A:</th>
<th>$[a]$ and $[b]$, the bit length $l$ of $a$ and $b$, the secret key $SK_{QR}$, public key $PK_P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input B:</td>
<td>Secret key $SK_P$, public key $PK_{QR}$, the bit length $l$</td>
</tr>
<tr>
<td>Output A:</td>
<td>$(a \leq b)$</td>
</tr>
</tbody>
</table>

1. A: $[x] \leftarrow [b] \cdot [2^l] \cdot [a]^{-1} \mod N^2$
2. A chooses a random number $r \leftarrow (0, 2^{\lambda+1}) \cap \mathbb{Z}$
3. A: $[z] \leftarrow [x] \cdot [r] \mod N^2$  \hspace{1cm} $\triangleright$ Blind $x$
4. A sends $[z]$ to B
5. B decrypts $[z]$
6. A: $c \leftarrow r \mod 2^l$
7. B: $d \leftarrow z \mod 2^l$
8. With A, B privately computes the encrypted bit $[t']$ such that $t = (d < c)$ using DGK
9. A encrypts $r_l$ and sends $[r_l]$ to B
10. B encrypts $z_l$
11. B: $[t] \leftarrow [t'] \cdot [z_l] \cdot [r_l]$
12. B: sends $[t]$ to A
13. A decrypts and outputs $t$

We will show the correctness of the protocol and then give a proof of security in the honest-but-curious model using modular composition. For the correctness, we just modify the proof of [Veu11].

**Proposition A.1.** Protocol 7 is correct and secure in the honest-but-curious model.

See proof in Appendix C.

A.2 Reversed encrypted comparison

We constructed Protocol 8 which is the same as Protocol 7, except that the roles of A and B are exchanged in Steps 8–13.
Proposition A.2. Protocol 8 is secure in the honest-but-curious model.

The proof is in Appendix C.

Protocol 8 Reversed comparing encrypted data

Input A: $[a]$ and $[b]$, public keys $PK_{QR}$ and $PK_P$
Input B: Secret keys $SK_P$ and $SK_{QR}$
Output B: $(a \leq b)$

Run Steps 1–7 of Protocol 7.

8: With B, A privately computes the encrypted bit $[t']$ such that $t' = (d < c)$ using DGK
9: B encrypts $z_l$ and sends $[z_l]$ to A
10: A encrypts $r_l$
11: A: $[t] \leftarrow [t'] \cdot [z_l] \cdot [r_l]$
12: A: sends $[t]$ to B
13: B decrypts and outputs $t$

B Preliminaries for proofs

B.1 Secure two-party computation framework

All our protocols are two-party protocols, which we label as party A and party B. In order to show that they do private computations, we work in the honest-but-curious (semi-honest) model as described in [Gol04].

Let $f = (f_A, f_B)$ be a (probabilistic) polynomial function and $\Pi$ a protocol computing $f$. A and B want to compute $f(a, b)$ where $a$ is A’s input and $b$ is B’s input, using $\Pi$ and with the security parameter $\lambda$. The view of party A during the execution of $\Pi$ is the tuple $V_A(\lambda, a, b) = (1^\lambda; a; r^A; m_1^A, \ldots, m_t^A)$ where $r$ is A’s random tape and $m_1^A, \ldots, m_t^A$ are the messages received by A. We define the view of B similarly. We also define the outputs of parties A and B for the execution of $\Pi$ on input $(a, b)$ as $\text{Output}_A(\lambda, a, b)$ and $\text{Output}_B(\lambda, a, b)$, and the global output as $\text{Output}_\Pi(\lambda, a, b) = (\text{Output}_A(\lambda, a, b), \text{Output}_B(\lambda, a, b))$.

To ensure security, we have to show that whatever A can compute from its interactions with B can be computed from its input and output, which leads us to the following security definition.

Definition B.1. The two-party protocol $\Pi$ securely computes the function $f$ if there exists two probabilistic polynomial time algorithms $S_A$ and $S_B$ such that for every possible input $a, b$ of $f$,

$$\{S_A(1^\lambda, a, f_A(a, b)), f(a, b)\} \equiv_c \{V_A(\lambda, a, b), \text{Output}_\Pi(\lambda, a, b)\}$$

and

$$\{S_B(1^\lambda, a, f_B(a, b)), f(a, b)\} \equiv_c \{V_B(\lambda, a, b), \text{Output}_\Pi(\lambda, a, b)\}$$

where $\equiv_c$ means computational indistinguishability against probabilistic polynomial time adversaries with negligible advantage in the security parameter $\lambda$.

To simplify the notation (and the proofs), hereinafter we omit the security parameter. As we mostly consider deterministic functions $f$, we can simplify the distributions we want to show being indistinguishable (see [Gol04]): when $f$ is deterministic, to prove the security of $\Pi$ that computes $f$, we only have to show that

$$S_A(a, f_A(a, b)) \equiv_c V_A(a, b)$$
$$S_B(b, f_B(a, b)) \equiv_c V_B(a, b)$$

Unless written explicitly, we will always prove security using this simplified definition.
B.2 Modular Sequential Composition

In order to ease the proofs of security, we use sequential modular composition, as defined in [Can98]. The idea is that the parties run a protocol $\Pi$ and use calls to an ideal functionality $f$ in $\Pi$ (e.g. $A$ and $B$ compute $f$ privately by sending their inputs to a trusted third party and receiving the result). If we can show that $\Pi$ respects privacy in the honest-but-curious model and if we have a protocol $\rho$ that privately computes $f$ in the same model, then we can replace the ideal calls for $f$ by the execution of $\rho$ in $\Pi$; the new protocol, denoted $\Pi^\rho$ is then secure in the honest-but-curious model.

We call hybrid model with ideal access to $f_1, \ldots, f_m$ or $(f_1, \ldots, f_m)$-hybrid model the semi-honest model augmented with an incorruptible trusted party $T$ for evaluating functionalities $f_1, \ldots, f_m$. The parties run a protocol $\Pi$ that contain calls to $T$ for the evaluation of one of $f_1, \ldots, f_m$. For each call, each party sends its input and wait until the trusted party sends the output back. We emphasize on the fact that the parties must not communicate until receiving $T$’s output (we consider only sequential composition). Ideal calls to the trusted party can be done several times, even for the same function, but each call is independent: $T$ does not maintain state between two calls.

Let $\Pi$ be a two-party protocol in the $(f_1, \ldots, f_m)$-hybrid model. Let $\rho_1, \ldots, \rho_m$ be real protocols (i.e. protocols in the semi-honest model) computing $f_1, \ldots, f_m$ and define $\Pi^{\rho_1, \ldots, \rho_m}$ as follows. All ideals calls of $\Pi$ to the trusted party for $f_i$ is replaced by a real execution of $\rho_i$; if party $P_j$ has to compute $f_i$ with input $x_j$, $P_j$ halts, starts an execution of $\rho_i$ with the other parties, gets the result $\beta_j$ when $\rho_i$ concludes, and continues as if $\beta_j$ was received from $T$.

**Theorem B.2.** [Can98] (Theorem 5) restated as in [LP08] (Theorem 3) – Let $f_1, \ldots, f_m$ be two-party probabilistic polynomial time functionalities and $\rho_1, \ldots, \rho_m$ protocols that compute respectively $f_1, \ldots, f_m$ in the presence of semi-honest adversaries.

Let $g$ be a two-party probabilistic polynomial time functionality and $\Pi$ a protocol that securely computes $g$ in the $(f_1, \ldots, f_m)$-hybrid model in the presence of semi-honest adversaries.

Then $\Pi^{\rho_1, \ldots, \rho_m}$ securely computes $g$ in the presence of semi-honest adversaries.

B.3 Cryptographic assumptions

**Assumption 1.** (Quadratic Residuosity Assumption – from [GM82]) Let $N = p \times q$ be the product of two distinct odd primes $p$ and $q$. Let $\mathbb{QR}_N$ be the set of quadratic residues modulo $N$ and $\mathbb{QR}_N$ be the set of quadratic non residues (i.e. $x \in \mathbb{QR}_N$ if $x$ is not a square modulo $N$ and its Jacobi symbol is 1).

$\{ (N, \mathbb{QR}_N) : |N| = \lambda \}$ and $\{ (N, \mathbb{QR}_N) : |N| = \lambda \}$ are computationally indistinguishable with respect to probabilistic polynomial time algorithms.

**Assumption 2.** (Decisional Composite Residuosity Assumption – from [Pai99]) Let $N = p \times q, |N| = \lambda$ be the product of two distinct odd primes $p$ and $q$. A number $z$ is said to be a $N$-th residue modulo $N^2$ if there exists a number $y \in \mathbb{Z}_{N^2}$

\[ z = y^N \mod N^2 \]

$N$-th residues are computationally indistinguishable from non $N$-th residues with respect to probabilistic polynomial time algorithms.

For further explanations about the last assumption, used for the FHE scheme, we refer the reader to [BGV12].

**Assumption 3.** (RLWE) For security parameter $\lambda$, let $f(x) = x^d + 1$ where $d$ is a power of 2. Let $q \geq 2$ be an integer. Let $R = \mathbb{Z}[x]/(f(x))$ and let $R_q = R/qR$. Let $\chi$ be a distribution over $R$. The $\text{RLWE}_{d,q,X}$ problem is to distinguish between two distributions: In the first distribution, one samples $(a_i, b_i)$ uniformly from $R_q^2$. In the second distribution, one first draws $s \leftarrow R_q$ uniformly and then samples $(a_i, b_i) \in R_q^2$ by sampling $a_i \leftarrow R_q$ uniformly, $e_i \leftarrow \chi$, and setting $b_i = a_i s + e_i$.

The $\text{RLWE}_{d,q,X}$ assumption is that the $\text{RLWE}_{d,q,X}$ problem is infeasible.
C Proofs

C.1 Comparison protocols

Proof of Proposition A.1. Correctness As a and b are l bits integers, \( x = 2^l + b - a \) is a \( l + 1 \) bits integer and its most significant bit (the \( l + 1 \)-th bit) is 1 iff \( a \leq b \). What protocol 7 actually does is computing this bit. The computations are done over encrypted data, using Paillier’s encryption scheme. In the rest of the proof, we will do as if the data were not encrypted under Paillier. The correctness will hold as long as we do not experience carry-overs modulo \( N \). In particular, this implies that \( l + 1 + \lambda < \log_2 N \). For operations over bits using QR, we don’t have this problem as we are operating on \( \mathbb{F}_2 \).

Again, since \( x \) is a \( l + 1 \) bit number, its most significant bit is \( x \div 2^l \) where \( \div \) denotes the integer division. We have \( x = 2^l (x \div 2^l) + (x \mod 2^l) \) where \( 0 \leq (x \mod 2^l) < 2^l \). As \( z = x + r \),

\[
\begin{align*}
z &= 2^l (z \div 2^l) + (z \mod 2^l) \\
&= 2^l ((x \div 2^l) + (r \div 2^l)) + ((x \mod 2^l) + (r \mod 2^l))
\end{align*}
\]

Hence, \( z \div 2^l = x \div 2^l + r \div 2^l \) if \( (x \mod 2^l) + (r \mod 2^l) < 2^l \) and \( z \div 2^l = (x \div 2^l) + (r \div 2^l) + 1 \) otherwise. More generally, \( z \div 2^l = (x \div 2^l) + (r \div 2^l) + t' \) where \( t' = 0 \iff (x \mod 2^l) + (r \mod 2^l) < 2^l \).

We can also notice that, if \( t' = 0 \), \( z \mod 2^l = (x \mod 2^l) + (r \mod 2^l) \) and \( z \mod 2^l = (x \mod 2^l) + (r \mod 2^l) - 2^l \) otherwise. As a consequence,

\[
t' = 0 \iff z \mod 2^l = (x \mod 2^l) + (r \mod 2^l) \\
\iff z \mod 2^l \geq (r \mod 2^l)
\]

In the end, as \( x \div 2^l \) is either 0 or 1, we can compute everything modulo 2

\[
x \div 2^l = (z \div 2^l) - (r \div 2^l) - t' \mod 2 \\
= z_t \oplus r_l \oplus t'
\]

Security We suppose that the encrypted bit \([t']\) is ideally computed (using calls to a trusted party in the hybrid model). We show that the protocol is secure in this model and conclude using the sequential modular composition theorem.

A’s view is \( V_A = ([a], [b], l, SK_{QR}, PK_P; r, \text{coins}; \{\tilde{t}\}) \) where \( SK_{QR} \) is the secret key for the QR cryptosystem, \( PK_P \) is the public key for Paillier’s cryptosystem, and coins are the random coins used for the encryptions of \( 2^l \), \( r \) and \( r_l \). Given \( ([a], [b], l, SK_{QR}, PK_P, a \leq b) \), we build the simulator \( S_A \):
1. Compute \( \{\tilde{t}\} \), an encryption of the bit \((a \leq b)\) under QR.
2. Pick \( \tilde{r} \leftarrow (0, 2^{l+1}) \cap \mathbb{Z} \).
3. Let \( \tilde{\text{coins}} \) be random coins for two Paillier encryptions and one QR encryption.
4. Output \( ([a], [b], l, SK_{QR}, PK_P; \tilde{r}, \tilde{\text{coins}}; \{\tilde{t}\}) \)

The distributions \( V_A([a], [b], l, SK_{QR}, PK_{QR}, PK_P, PK_P) \) and \( S_A([a], [b], SK_{QR}, PK_P, a \leq b) \) are exactly the same because the randomness is taken from the same distribution in both cases, and the QR cyphertext encrypts the same bit.

B’s view is \( V_B = (PK_{QR}, SK_P, l, [z]; \text{coins}; [\tilde{t}], [r_l]) \) where coins are the random coins used for the encryption of \( z_l \). The simulator \( S_B(PK_{QR}, SK_P, l) \) runs as follows:
1. Pick \( \tilde{z} \leftarrow (0, 2^{l+1}) \cap \mathbb{Z} \).
2. Encrypt \( \tilde{z} \) under Paillier: \([\tilde{z}]\).
3. Generate \( [\tilde{t}] \) and \( [\tilde{r}_l] \), two encryptions of random bits under QR
4. Let \( \text{coins} \) be random coins for one QR encryption.
5. Output \( (PK_{QR}, SK_P, l, [\tilde{z}]; \text{coins}; [\tilde{t}], [r_l]) \)
The random tapes coins and $\hat{\text{coins}}$ are generated in the exact same manner and independently from any other parameter, so

$$(\text{PK}_{QR}, \text{SK}_P, \{\hat{z}\}; \hat{\text{coins}}; \{\hat{t}\}, [\hat{r}_1]) = (\text{PK}_{QR}, \text{SK}_P, \{\hat{z}\}; \hat{\text{coins}}; \{\hat{t}\}, [\hat{r}_1])$$

Recall that $z = x + r \mod N$ where $x$ is an $l$ bits integer and $r$ is an $l + \lambda$ bits integer. But as we chose $l + 1 + \lambda < \log_2 N$, we have $z = x + r$. The distribution of $z$ is statistically indistinguishable from the distribution of $z$ (the distributions are distinguishable with an advantage of $2^{-\lambda}$ at most).

We also directly have that $(\text{SK}_P, \{z\}) =_{s} (\text{SK}_P, \{\hat{z}\})$ and as a consequence, as the distribution of $\hat{z}$ and $z$ is independent from $\hat{t}$ and $\hat{r}_1$,

$$(\text{PK}_{QR}, \text{SK}_P, \{\hat{z}\}; \hat{\text{coins}}; \{\hat{t}\}, [\hat{r}_1]) =_{s} (\text{PK}_{QR}, \text{SK}_P, \{z\}; \text{coins}; \{t\}, [r_1])$$

By semantic security of QR,

$$(\text{PK}_{QR}, \text{SK}_P, l, \{z\}; \text{coins}; \{\hat{t}\}, [\hat{r}_1]) =_{c} (\text{PK}_{QR}, \text{SK}_P, l, \{z\}; \text{coins}; \{t\}, [r_1])$$

and

$S_B(\text{PK}_{QR}, \text{SK}_P, l) =_{c} V_B([a], [b], l, \text{SK}_{QR}, \text{PK}_{QR}, \text{SK}_P, \text{PK}_P)$

We conclude the proof of security using modular sequential composition. We replace the ideal calls for computing the encrypted bit $[t']$ by the provable secure DGK protocol and invoke Theorem B.2 to prove security in the semi-honest model. $\square$

**Proof of Proposition A.2.** The proof of security is similar to the one of Proposition A.1. Again we first suppose that $[t']$ is ideally computed (hybrid model).

A’s view is $V_A = ([a], [b], l, \text{PK}_{QR}, \text{PK}_P; r, \text{coins}; [t'], [z])$ where $\text{PK}_{QR}$ is the public key for the QR cryptosystem, $\text{PK}_P$ is the public key for Paillier’s cryptosystem and coins is the random tape used for the Paillier encryptions of $r$ and $2^l$, and the QR encryption of $r_1$.

Given $([a], [b], \text{PK}_{QR}, \text{PK}_P)$, we build the simulator $S_A$:

1. Pick $\hat{r} \leftarrow (0, 2^{l+1}) \cap \mathbb{Z}$.
2. Generate $[\hat{t}]$ and $[\hat{r}_1]$, two encryptions of random bits under QR
3. Let $\hat{\text{coins}}$ be random coins for two Paillier encryptions and one QR encryption.
4. Output $([a], [b], l, \text{PK}_{QR}, \text{PK}_P; \hat{r}, \hat{\text{coins}}; [\hat{z}])$

For both cases (A’s view and the simulator $S_A$), $r$ and $\hat{r}$ are taken from the same uniform distribution over $(0, 2^{l+1}) \cap \mathbb{Z}$, and coins and $\hat{\text{coins}}$ are random tapes of the same length, so

$S_A([a], [b], \text{PK}_{QR}, \text{PK}_P) = ([a], [b], l, \text{PK}_{QR}, \text{PK}_P; r, \text{coins}; [z])$

By semantic security of the QR cryptosystem, we conclude with the computational indistinguishability of $S_A$ and $V_A$ distributions:

$S_A([a], [b], \text{PK}_{QR}, \text{PK}_P) = ([a], [b], l, \text{PK}_{QR}, \text{PK}_P; r, \text{coins}; [z])$

$=_{c} ([a], [b], l, \text{SK}_{QR}, \text{PK}_{QR}, \text{SK}_P, \text{PK}_P)$

$= V_A([a], [b], l, \text{SK}_{QR}, \text{PK}_{QR}, \text{SK}_P, \text{PK}_P)$

28
Finally, we have the end of the loop for iteration $i$.

**Correctness**

Proof of Proposition 4.1.

C.1.1 Argmax

- **semi-honest model.**

Recall that $z = x + r$ where $x$ is an $l$ bits integer and $r$ is an $l + \lambda$ bits integer. The distribution of $\tilde{z}$ is statistically indistinguishable from the distribution of $z$. We also directly have that $(SK_P, [\tilde{z}]) \equiv_s (SK_P, [z])$ and as a consequence, as the distribution of $\tilde{z}$ and $z$ is independent from $P'$,

$$(SK_QR, SK_P, l, \tilde{z}, \tilde{t}; \text{coins}) \equiv_s (SK_QR, SK_P, l, z, t; \text{coins})$$

Moreover, by construction, $(SK_QR, [\tilde{t}]) = (SK_QR, [a < b])$ and

$$(SK_QR, SK_P, l, [\tilde{z}], \tilde{t}; \text{coins}) \equiv_s (SK_QR, SK_P, l, [z], t; \text{coins})$$

Finally, we have

$$S_B(SK_QR, SK_P, a \leq b) \equiv_s V_B([a], [b], l, SK_QR, PK_QR, SK_P, PK_P).$$

Again, we conclude the proof of security using modular sequential composition. We replace the ideal calls for computing the encrypted bit $[t']$ by the provable secure DGK protocol and invoke Theorem B.2 to prove security in the semi-honest model.

C.1.1 Argmax

**Proof of Proposition 4.1. Correctness** To prove correctness, we have to show that the following invariant holds: at the end of the loop for iteration $i$, $m$ is the maximum of $\{a_{\pi(j)}\}_{1 \leq j \leq i}$ and $a_{\pi(i)} = m$.

If this holds, at the end of the loop iterations $a_{\pi(i)}$ is the maximum of $\{a_{\pi(j)}\}_{1 \leq j \leq k} = \{a_j\}_{1 \leq j \leq k}$, hence $i_0 = \arg\max_j a_{\pi(j)}$ and $\pi^{-1}(i_0) = \arg\max_j a_j$.

At initialization (line 4), the invariant trivially holds as the family $\{a_{\pi(j)}\}_{1 \leq j \leq i}$ contains only one element. Suppose the property is true for iteration $i - 1$. Let us distinguish two cases:

- **If** $b_i$ is true (i.e. $m \leq a_{\pi(i)}$), $\max\{a_{\pi(j)}\}_{1 \leq j \leq i} \leq a_{\pi(i)}$, as the invariant holds for the previous iteration, and then $\max\{a_{\pi(j)}\}_{1 \leq j \leq i} = a_{\pi(i)}$. Then $i_0$ is set to $i$, $v_i = a'_i$ and $b_i = 1$. As a consequence, $m$ is set by A to

$$v_i + (b_i - 1).r_i - b_i.s_i = a'_i - s_i = a_{\pi(i)}$$

We have clearly that $a_{\pi(i_0)} = a_{\pi(i)} = m$ and $m = \max\{a_{\pi(j)}\}_{1 \leq j \leq i}$, the invariant holds at the end of the $i$-th iteration in this case.
We prove security in the hybrid model where line 5 of the protocol is ideally executed: we ask a trusted party $T$ to compute the function $f$ and $m$ is unchanged. As both $m$ and $i_0$ stayed the same and $\max\{a_{\pi(j)}\}_{1 \leq j \leq i-1} = \max\{a_{\pi(j)}\}_{1 \leq j \leq i-1}$, the invariant holds at the end of the $i$-th iteration in this case.

**Security** We prove security in the hybrid model where line 5 of the protocol is ideally executed: we ask a trusted party $T$ to compute the function $f([x],[y]), l, SK_{QR}, PK_{QR}, SK_P, PK_P$ in the $f$-hybrid model where

$$f([x],[y]), l, SK_{QR}, PK_{QR}, SK_P, PK_P$$

$$= (f_A(x, y, l, SK_{QR}, PK_{QR}, SK_P, PK_P);$$

$$f_B([x],[y]), l, SK_{QR}, PK_{QR}, SK_P, PK_P)$$

and $f$ computes the function of Protocol 8, i.e. $f_A$ returns nothing and $f_B$ returns the bit $x \leq y$.

We will conclude using Theorem B.2.

A’s view is

$$V_A = ([a_i])_{i=1}^k, l, PK_{QR}, PK_P;$$

$$\pi, \{r_i\}_{i=2}^k, \{s_i\}_{i=2}^k, \text{coins};$$

$$([v_i])_{i=2}^k, ([b_i])_{i=2}^k, \pi(\argmax a_i))$$

where coins is the random tape for encryptions. To simulate A’s real view, the simulator $S_A$ does the following on input $([a_1], \ldots, [a_k], l, PK_{QR}, PK_P, \argmax, a_i)$:

1. Picks a random permutation $\pi$ of $\{1, \ldots, k\}$
2. Picks $k - 1$ random integers $\tilde{r}_2, \ldots, \tilde{r}_k$ in $(0, 2)^{r+\lambda} \cap \mathbb{Z}$
3. Picks $k - 1$ random integers $\tilde{s}_2, \ldots, \tilde{s}_k$ in $(0, 2)^{r+\lambda} \cap \mathbb{Z}$
4. Generates $k - 1$ random Paillier encryptions $[\tilde{v}_2], \ldots, [\tilde{v}_k]$.  
5. Generates $k - 1$ random bits $\tilde{b}_i$
6. Generate a random tape for $2(k - 1)$ Paillier encryption coins
7. Outputs

$$([a_i])_{i=1}^k, l, PK_{QR}, PK_P;$$

$$\tilde{\pi}, \{\tilde{r}_i\}_{i=2}^k, \{\tilde{s}_i\}_{i=2}^k, \text{coins};$$

$$([\tilde{v}_i])_{i=2}^k, ([\tilde{b}_i])_{i=2}^k, \tilde{\pi}(\argmax a_i))$$

We define the following hybrids:

- $H_0 = V_A([a_1], \ldots, [a_k], l, SK_{QR}, PK_{QR}, SK_P, PK_P)$
- $H_1 = ([a_i])_{i=1}^k, l, PK_{QR}, PK_P; \pi, \{r_i\}_{i=2}^k, \{s_i\}_{i=2}^k, \text{coins};$  
  $$([v_i])_{i=2}^k, ([b_i])_{i=2}^k, \pi(\argmax, a_i))$$
- $H_2 = ([a_i])_{i=1}^k, l, PK_{QR}, PK_P; \pi, \{\tilde{r}_i\}_{i=2}^k, \{\tilde{s}_i\}_{i=2}^k, \text{coins};$  
  $$([\tilde{v}_i])_{i=2}^k, ([\tilde{b}_i])_{i=2}^k, \pi(\argmax, a_i))$$
- $H_3 = S_A([a_1], \ldots, [a_k], l, PK_{QR}, PK_P, \argmax, a_i)$
By semantic security of Paillier’s cryptosystem,
\[
\{([a_i])_{i=1}^k, l, PK_{QR}, PK_P; \pi, \{r_i\}_{i=2}^k, \{s_i\}_{i=2}^k; \\
\{[v_i]_{i=1}^k, [b_i]_{i=2}^k, \pi(\text{argmax } a_i)\}
\equiv_e \\
\{([a_i])_{i=1}^k, l, PK_{QR}, PK_P; \pi, \{r_i\}_{i=2}^k, \{s_i\}_{i=2}^k;
\{[v_i]_{i=1}^k, [b_i]_{i=2}^k, \pi(\text{argmax } a_i)\}
\]

and \(H_0 \equiv_e H_1\) as \(\pi(\text{argmax } a_i) = i_0\)

Given that the \(\tilde{r}_i, \tilde{s}_i\) and coins are generated according to the same distribution as \(r_i, s_i\) (uniform over \((0, 2)^{l+\lambda} \cap \mathbb{Z}\)) and coins (random tape for \(2(k-1)\) Paillier encryptions), and that they are completely independent from the \(\tilde{v}_i\) or \(\pi\), the hybrids \(H_1\) and \(H_2\) are equal.

Similarly, the distribution of \((\pi, \pi(\text{argmax } a_i))\) and 
\((\tilde{r}, \tilde{r}(\text{argmax } a_i))\) are exactly the same. As \(\pi\) and \(\tilde{\pi}\) are independent from the other parameters, we also have \(H_2 = H_3\).

Hence, we showed that
\[
V_A(\{([a_i])_{i=1}^k, l, SK_{QR}, PK_{QR}, SK_P, PK_P\}) 
\equiv_e S_A(\{([a_i])_{i=1}^k, l, PK_{QR}, PK_P, \text{argmax } a_i\}).
\]

B’s view is
\[
V_B = (SK_P, SK_{QR}, l; \text{coins}; \{b_i\}_{i=2}^k, \{[m_i']_{i=2}^k, ([a_i']_{i=2}^k
\]

where coins are the random coins for \(k - 1\) Paillier cyphertext refresh. The simulator \(S_B(SK_P, SK_{QR}, l)\) runs as follows:

1. Generates a random permutation \(\tilde{\pi}\) of \(\{1, \ldots, k\}\)
2. Set \([\tilde{a}_i] = [i]\
3. Run the protocol with the \([\tilde{a}_i]\) as input data, \(\tilde{\pi}\) as the permutation, and same parameters otherwise. Let
\[
(SK_P, SK_{QR}, l; \tilde{\pi}(\text{coins}); \{b_i\}_{i=2}^k, \{[m_i']_{i=2}^k, ([a_i']_{i=2}^k)
\]

be B’s view of this run.
4. Outputs
\[
(SK_P, SK_{QR}, l; \tilde{\pi}(\text{coins}); \{b_i\}_{i=2}^k, \{[m_i']_{i=2}^k, ([a_i']_{i=2}^k)
\]

Let \(p: \{a_i\}_{1 \leq i \leq k} \mapsto \{1, \ldots, k\}\) be the function that associates \(a_i\) to its rank among the \(a_i\) (in ascending order). Let us fix the permutation \(\pi\) for a while and define the following hybrids:

0. \(H_0 = V_B(\{([a_i])_{i=1}^k, l, SK_{QR}, PK_{QR}, SK_P, PK_P\})\)
1. \(H_1 = V_B(\{([p(a_i)])_{i=1}^k, l, SK_{QR}, PK_{QR}, SK_P, PK_P\})\)

We will show that these hybrids are statistically equal for every permutation \(\pi\).

As \(p(.)\) is a map that does not change the order of the \(a_i\), we have that for all \(i, j, a_i \leq a_j \leftrightarrow p(a_i) \leq p(a_j)\). As a consequence, for a given permutation \(\pi\), the bits \(b_i\) do not change if we replace the \(a_i\) by \(p(a_i)\). Similarly, the way the \(a_i'\) and \(m_i'\) are generated for \(H_0\) and \(H_1\) is the same: blinding by adding random noise from \((0, 2^{\lambda+l} \cap \mathbb{Z})\). Thus, \(H_0 \equiv_s H_1\).

Now, we want to show that \(H_1 \equiv_s S_B(SK_P, SK_{QR}, l) - \text{we do not fix } \pi\text{ anymore. Let } \pi_0\text{ be the permutation such that } p(a_i) = \pi_0(i).\) We can then rewrite \(H_1\) as
\[
H_1 = V_B([\pi_0(1)], \ldots, [\pi_0(k)], l, SK_{QR}, PK_{QR}, SK_P, PK_P)
\]

As \(\tilde{\pi}\) and \(\pi \circ \pi_0\) are statistically indistinguishable, we have \(H_1 \equiv_s S_B(SK_P, SK_{QR}, l):\) recall that \(S_B\’s\ output is the view of \(B\) when the protocol is run with the set \(\{a_i = i\}\) as input set and \(\tilde{\pi}\) as the permutation. Hence
\[
V_B([a_1], \ldots, [a_k], l, SK_{QR}, PK_{QR}, SK_P, PK_P) 
\equiv_s S_B(SK_P, SK_{QR}, l)
\]
We conclude the proof of security using modular sequential composition. We replace the ideal calls for computing the encrypted bits \( b_i \) by the provable secure Protocol 8 and invoke Theorem B.2 to prove security in the semi-honest model.

### C.2 Changing the encryption scheme

**Proof of Proposition 4.2.** In this protocol the computed function is probabilistic, and we have to show security according to the full definition (cf. section B.1). The function is:

\[
f([c]_1, PK_1, PK_2, SK_1, SK_2) = ([c]_2, 0)
\]

For the sake of simplicity, we do not take into account the randomness used for the encryptions of \( r \) for A and \( c' \) for B. As before, the distribution of these coins for one party is completely independent of the other elements to be taken in account in the simulations, so we just do not mention them in security proof.

A’s view is \( V_A = (PK_1, PK_2, [c]_1; r; [c']_2) \). A’s output is \([c]_2\). The simulator \( S_A(PK_1, PK_2, [c]_1) \) runs as follows:

1. Picks uniformly at random \( \tilde{r} \leftarrow M \) and \( c' \leftarrow M \).
2. Generates the encryption \([c']_2 \) of \( c' \) under \( E_2 \).
3. Outputs \((PK_1, PK_2, [c]_1; \tilde{r}; [c']_2)\).

\( r \) and \( \tilde{r} \) are taken from the same distribution, independently from any other parameter, so

\[
\begin{align*}
\{(PK_1, PK_2, [c]_1; \tilde{r}; [c']_2); \ f([c]_1, PK_1, PK_2, SK_1, SK_2)\} \\
= \{(PK_1, PK_2, [c]_1; r; [c']_2); \ f([c]_1, PK_1, PK_2, SK_1, SK_2)\}
\end{align*}
\]

(\( c' \) depends on \( r \) but does not appear in the previous distributions). By semantic security of scheme \( E_2 \) we have that

\[
\begin{align*}
\{(PK_1, PK_2, [c]_1; r; [c']_2); \ f([c]_1, PK_1, PK_2, SK_1, SK_2)\} \\
\equiv_c \{(PK_1, PK_2, [c]_1; r; [c']_2); [c]_2\}
\end{align*}
\]

and so

\[
\begin{align*}
S_A([c]_1, PK_1, PK_2), f([c]_1, PK_1, PK_2, SK_1, SK_2) \\
\equiv_c \{V_A([c]_1, PK_1, PK_2, SK_1, SK_2), Output([c]_1, PK_1, PK_2, SK_1, SK_2)\}
\end{align*}
\]

B’s view is \( V_B = (SK_1, SK_2; [c + r]_1) \). We build a simulator \( S_B(SK_1, SK_2) \):

1. Picks a random \( \tilde{c} \leftarrow M \).
2. Encrypt \( \tilde{c} \) under \( E_1 \).
3. Outputs \((SK_1, SK_2, [\tilde{c}]_1)\).

Again, the distribution of \( \tilde{c} \) and \( c + r \) are identical, so the real distribution \( \{(SK_1, SK_2; [c + r]_1); [c]_2\} \) and the ideal distribution \( \{(SK_1, SK_2; [\tilde{r}]_1); f([c]_1, PK_1, PK_2, SK_1, SK_2)\} \) are statistically indistinguishable.

### C.3 Computing dot products

**Proof of Proposition 4.3.** As B does not receive any message, its view only consists in its input and its random tape used for the encryptions. Hence the simulator \( S_B \) simply generate random coins and

\[
S_B(y, SK_P) = (y, SK_P; \text{coins}) = V_B(x, y, SK_P, PK_P).
\]

where \( \text{rand} \) are the random coins.

A’s view is \( V_A = (x, PK_P; [r^A]; [y_1], \ldots, [y_n]) \). On input \((x, PK_P, [r])\), the simulator \( S_A \) does the following:
1. Generates $n$ encryptions of 0 using Paillier: $c_1, \ldots, c_n$.
2. Generates the random coins necessary for a Paillier re-randomization and put them in $\tilde{\text{coins}}$.
3. Outputs $(x, PK_P; \tilde{\text{coins}}; c_1, \ldots, c_n)$.

Coins and $\tilde{\text{coins}}$ come from the same distribution, independently from other parameters. Thus,

$$\{(x, PK_P; c_1, \ldots, c_n); \langle x, y \rangle \} \\ = \{(x, PK_P; \tilde{\text{coins}}; c_1, \ldots, c_n); \langle x, y \rangle \}$$

and by semantic security of Paillier,

$$\{(x, PK_P; c_1, \ldots, c_n); \langle x, y \rangle \} \\ \equiv_c \{(x, PK_P; \tilde{\text{coins}}; \langle y_1 \rangle, \ldots, \langle y_n \rangle); \langle v \rangle \}$$

i.e., when $f$ is $f(x, y, SK_P, PK_P) = (\langle x, y \rangle, \emptyset)$

$$\{SA(x, PK_P, \langle v \rangle); f(x, y, SK_P, PK_P)\} \\ \equiv_c \{VA(x, y, SK_P, PK_P); Output(x, y, SK_P, PK_P)\}$$

\[\square\]

### C.4 Classifiers

#### Hyperplane decision

**Proof of Proposition 5.1.** The client’s view is

$$V_C = (PK_P, PK_{QR}, x; \{\langle v_i \rangle\}_{i=1}^k, i_0).$$

The simulator $S_C$, on input $(PK_P, SK_{QR}, x, k^*)$ where $k^* = \arg\max_{i \in [k]} P(C = c_i | X = x)$ does the following:

1. Generate $k$ random Paillier encryptions $\langle \bar{v}_i \rangle$
2. Output $(PK_P, SK_{QR}, x; \{\langle \bar{v}_i \rangle\}_{i=1}^k, k^*)$

As the index $i_0$ that the client receives is its output, and as Paillier’s cryptosystem is semantically secure, the distributions $S_C = (PK_P, SK_{QR}, x; \{\langle v \rangle\}_{i=1}^k, k^*)$ and $V_C = (PK_P, SK_{QR}, x; \{\langle v_i \rangle\}_{i=1}^k, i_0)$ are computationally indistinguishable.

As the server views nothing but its inputs (the server does not receive any message in the hybrid model), we use for the trivial simulator that just outputs its inputs for the proof of security.

As Protocols 1 and 3 are secure in the honest-but-curious model, we obtain the security of the hyperplane decision protocol using modular sequential composition (Theorem B.2).

#### Bayes classifier

**Proof of Proposition 6.1.** The client’s view is

$$V_C = (PK_P, SK_{QR}, x; \langle \bar{P} \rangle, \{\langle T_{i,j} \rangle\}, i_0).$$

The simulator $S_C$, on input $(PK_P, SK_{QR}, x, i_{\max})$ where $i_{\max} = \arg\max_j P(C = c_j | X = x)$,

- generates tables of random Paillier encryptions $\langle \bar{P} \rangle$ and $\{\langle T_{i,j} \rangle\}$;
- outputs $(PK_P, SK_{QR}, x; \langle \bar{P} \rangle, \{\langle T_{i,j} \rangle\}, i_{\max})$.
As the integer $i_0$ that the client receives is its output, and as Paillier’s cryptosystem is semantically secure, the distributions $S_C = (PK_P, SK_{QR}, x; \|P\|, \|\{T_i\}_j\|), i_{\max}$ and $V_C = (PK_P, SK_{QR}, x; \|P\|, \|\{T_i\}_j\|, i_0)$ are computationally indistinguishable.

Again, as the server views nothing but its inputs (the server does not receive any message in the hybrid model), we use the trivial simulator that outputs its inputs and the random coins for the encryption for the proof of security.

As Protocol 1 is secure in the honest-but-curious model, we obtain the security of the hyperplane decision protocol using modular sequential composition (Theorem B.2).

Decision tree

Proof of Proposition 7.1. The proof of security for the server is very easily obtained using modular sequential composition of the comparison protocol and Protocol 2: in the hybrid model, the client receives nothing but the encrypted result.

For the client also the proof is trivial, using modular sequential composition and the semantical security of QR and of the FHE scheme: the encryptions of bits $b_i$ are computational indistinguishable from random bits whether they are encrypted under QR or the FHE scheme. 

\end{proof}