Machine Learning Classification over Encrypted Data

Raphaël Bost
DGA*

Raluca Ada Popa
MIT

Stephen Tu
MIT

Shafi Goldwasser
MIT

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. These protocols may also be combined with AdaBoost. They rely on a library of building blocks for constructing classifiers securely, and we demonstrate the versatility of this library by constructing a face detection classifier.

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 in many settings today, such as medical or genomics predictions, spam detection, face recognition, and finance. Many of these applications handle sensitive data [WGH12, SG11, SG13], and it is important that the data and the classifier remain confidential.

Consider the typical setup with 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 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 $C(x, w)$.

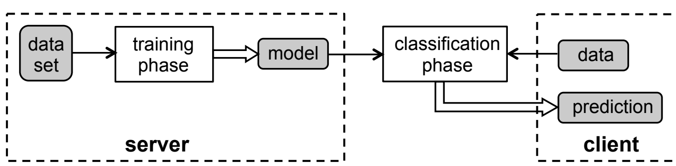


Figure 1: Model overview. Each shaded box indicates private data that should be accessible to only one party: the dataset and the model to the server, and the input and prediction result to the client. Each straight non-dashed rectangle indicates an algorithm, single arrows indicate inputs to these algorithms, and double arrows indicate outputs.

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 hospital having a model built out of the medical profiles of current patients; the model is sensitive because it can leak information about the current patients, and its usage has to be HIPPA[1] compliant. A new person would like to predict if she would be treated successfully at the hospital or whether she is likely to develop a certain disease, but does not want to reveal her sensitive medical profile to the hospital (unless

*Direction Générale de l'Armement. 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.

[1]Health Insurance Portability and Accountability Act of 1996

Learning algorithm	Classifier
Perceptron	Hyperplane decision
Least squares	Hyperplane decision
Fischer linear discriminant	Hyperplane decision
Support vector machine	Hyperplane decision
Naive Bayes	Naïve Bayes
Decision trees (ID3/C4.5)	Decision trees

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

she actually gets treated there). Ideally, the hospital and the patient run a protocol at the end of which the patient 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 model w as input. 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.

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

While generic secure multi-party computation [Yao82, GMW87, MNPS04] can implement any classifier in principle, due to their generality, such schemes are not efficient. Most existing work in machine learning and privacy [LP00, DHC04, WY04, ZW05, BDMN05, VKC08, GLN13] 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 [EFG$^+$09, SSW10, 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 harder to support efficiently. The second is how to provide 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 these are three of the most common classifiers, various settings use other classifiers or use a combination of these three classifiers. 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 adapting existing schemes or by constructing new schemes.

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 ensure that:

- The input and output of all our building blocks are data encrypted with additively homomorphic encryption. Then, we provide a mechanism to switch from one encryption scheme to another to enable composing our building blocks;
- The API of these building blocks is flexible: even though each building block computes a fixed function, it allows the choice of which party obtains the output of the computation, whether the output is encrypted or decrypted, and of which party provides the inputs to the protocol;
- The security of these protocols composes using modular sequential composition [Can98].

We then use these building blocks to construct our three classifiers. Some of these incorporate additional techniques, such as using SIMD slots with fully homomorphic encryption to improve performance of our decision trees by at least twice.

We emphasize that the contribution of our building blocks library goes beyond the classifiers we build in this paper: one can use them to 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.

Our last contribution is an implementation and evaluation of our building blocks and classifiers. We evaluate our classifiers on real datasets with sensitive data about breast cancer, credit card approval, audiology, and nursery data; our algorithms are efficient, running in milliseconds up to at most a few seconds.

The rest of the paper is organized as follows. Section 2 describes related work, Sections 3–4 provide the necessary machine learning and cryptographic background, Section 5 presents our building blocks, Sections 6–9 describe our classifiers, and Sections 10–11 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 [MNPS04]. However, these rely on heavy cryptographic machinery, and applying them directly to our problem setting would be far too inefficient. Hence, when practicality is important, researchers attempt to design specialized protocols.

Previous work focusing on privacy-preserving machine learning can be broadly divided into two categories: (i) techniques for private training, and (ii) techniques for private classification (recall the distinction from Figure 1). There is a large body of literature related to the first category which we discuss in Section 2.1. Our work falls in the second category, where there has been less work done, which 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 incomparable to the work in the differential privacy community on private machine learning (see *e.g.* [CMS11]). Our work aims to protect the confidentiality of user data, whereas differential privacy seeks to bound the amount of statistical inference that can be performed on a particular individual.

2.1 Privacy-preserving training

Existing techniques have been developed for privacy preserving *training* for Naïve Bayes [VKC08, WY04, ZW05], decision trees [BDMN05, LP00], linear discriminant classifiers [DHC04], and more general kernel methods [LLM06].

Grapel *et al.* [GLN13] 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) is impractical, 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, would 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 with the existing constructions.

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), keeps the model private, and is twice as fast on branching programs.

Other works [EFG+09, SSW10, AB06, AB07] construct specific face recognition or detection classifiers. We focus on providing a set of generic building blocks to construct more complex classifiers. In Section 11.1.2, 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 subjects have been well studied previously; see [Yao82, DGK07, DGK09, Veu11, LT05, AB06] for comparison techniques and [AD01, GLLM05, Kil05, AB06] for techniques to compute dot products. Our contribution is not focused on improving the performance of these techniques, but instead on showing how to combine them in a secure and modular fashion.

3 Background and definitions

3.1 Classification in machine learning algorithms

The standard problem in classification is to construct a function $f : \mathbb{R}^d \mapsto \{c_1, ..., c_k\}$ that takes as input a feature vector $x \in \mathbb{R}^d$ and returns which of k classes the input belongs to. We now describe how three popular classifiers work on regular, unencrypted data. For more details, we refer the reader to [BN06].

Hyperplane decision-based classifiers. Given a hypothesis space $\mathcal{H}$ equipped with an inner product $\langle \cdot, \cdot \rangle$, a hyperplane based classifier uses a linear model $w \in \mathcal{H}$ to solve a binary $(k = 2)$ classification problem; any user input x such that $\langle w, \phi(x) \rangle \geq 0$ is classified into class c_2, otherwise it is labelled as part of class c_1. Here, $\phi : \mathbb{R}^d \mapsto \mathcal{H}$ denotes the feature mapping from $\mathbb{R}^d$ to $\mathcal{H}$ [BN06]. These kinds of classifiers are inherently binary, and there are many techniques for extending them to k-class classification for $k > 2$. 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 (*cf.* [BN06]):

$$k^* = \operatorname*{argmax}_{i \in [k]} \langle w_i, \phi(x) \rangle$$

This framework is general enough to cover many common algorithms, such as support vector machines (SVMs), logistic regression, and least squares. In this work, we focus on the case when $\mathcal{H} = \mathbb{R}^d$ (and $\phi(x) = x$) 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, $\phi(x) = Px$ for a randomized projection matrix P chosen during training. Notice that Px 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.

Naïve Bayes classifiers. Whereas hyperplane decision algorithms try to learn the decision surface directly from the data, generative models (such as Naïve Bayes) work by learning the underlying joint probability distribution of the data and labels, and making predictions based on the distribution. More specifically, given k classes $\{c_1, ..., c_k\}$, a generative model using a *maximum a posteriori* decision rule works by choosing the class with the highest posterior probability:

$$k^* = \operatorname*{argmax}_{i \in [k]} p(c_i | x) = \operatorname*{argmax}_{i \in [k]} p(c_i, x)$$

where the second equality follows from applying Bayes' rule. In general the joint probabilities $p(c_i, x)$ can be arbitrarily complex. We focus specifically on the Naïve Bayes model, which assumes that $p(c_i, x)$ has the following factorization:

$$p(c_i, x_1, ..., x_d) = p(c_i) \prod_{j=1}^{d} p(x_j | c_i)$$

namely, each of the d features are conditionally independent given the class. For simplicity, we will assume that each x_i is discrete, so the $p(x_i|c_j)$'s are really probability masses. The model w in the Naïve Bayes model is completely specified by $\{p(c_i)\}_{i=1}^k$ and $\{\{p(x_j|c_i)\}_{j=1}^d\}_{i=1}^k$.

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 leaf node's label is the feature vector's classification.

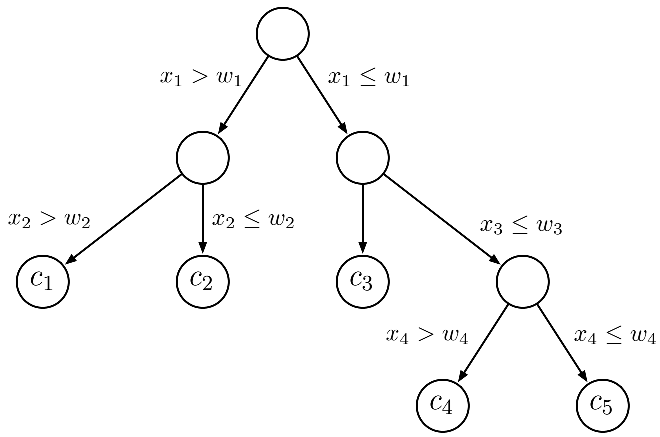


Figure 2: Decision tree

Figure 2 gives an example of a decision tree. The model consists of the tree itself and the decision criteria (in this case the thresholds $w_1, \ldots, w_4$). In Section 8, we present the tools to efficiently perform private classification on binary decision trees (binary refers to the structure of the tree, not the domain of the attributes).

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 $\text{HE}.\mathsf{Enc}(a)$ and $\text{HE}.\mathsf{Enc}(b)$, there exists a public key operation $\oplus$ such that $c \leftarrow \text{HE}.\mathsf{Enc}(a) \oplus \text{HE}.\mathsf{Enc}(b)$ is an encryption of $a + b$. The cryptosystems we use are:

1. the QR 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 cryptosytems 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 A.1. To enable us to compose various protocols into a bigger protocol in a secure way, we invoke modular sequential composition (see Appendix A.2).

Input A	Input B	Output A	Output B	Implementation
a	b	$[a < b]$	–	DGK
$[\![a]\!]$, $[\![b]\!]$	–	$a \leq b$	–	Protocol 1
$[\![a]\!]$, $[\![b]\!]$	–	–	$[a \leq b]$	Protocol 1 without the last two steps
$[\![a]\!]$, $[\![b]\!]$	–	–	$a \leq b$	Protocol 2
$[\![a]\!]$, $[\![b]\!]$	–	$[a \leq b]$	–	Protocol 2 without the last two steps

Table 2: The various setups under which our comparison protocol works and their corresponding implementations. Whenever a party has an encrypted output, the output can be decrypted by the other party.

4 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 encryptions 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 consists in $\mathbb{Z}_N$ where N is the public modulus, and we denote by $[\![m]\!]$ an integer m encrypted under Paillier.

The plaintext space of the FHE scheme is $\mathbb{F}_2$, but we can pack more than one bit in a single ciphertext using SIMD slots (*cf.* [SV11]). Hence, $[\![\![b_1, \ldots, b_n]\!]\!]$ denotes the encryption of the bit vector $(b_1, \ldots, b_n)$ using these slots, and $[\![\![b]\!]\!]$ the encryption of the single bit b (b is copied in every slot).

5 Building blocks

In this section, we develop a library of building blocks, which we later use to build the classifiers. We also intend this library for being useful in constructing other classifiers than the ones described in our paper. The building blocks in this section combine existing techniques with either new techniques or engineering.

5.1 Comparison

Here we describe our comparison protocol. In order for this protocol to be used in various classifiers, its setup needs to be *flexible*: namely, it has to support a range of choices for which party gets the input or the output and in what form. Table 2 shows the various ways our comparison protocol can be used. To implement each row of the table, we build on existing protocols.

5.1.1 Comparison with unencrypted inputs

For comparison with unencrypted inputs, we use a variation of the protocol described in [DGK07, DGK09, EFG+09], which we call DGK.

DGK. In this protocol, two parties A and B want to compare two private l bit integers a and b. B has a Paillier secret key SK_P and a GM secret key SK_{GM}, whereas A has the associated public keys. At the end of the protocol, A outputs the encrypted bit $[t]$ where $t = (a < b)$. We refer the reader to [EFG+09] (*c.f.* Section 5) and [DGK07] (*c.f.* Section 3) for the full description and proof.

Performance. In [Veu11], another comparison protocol is presented, called LSIC. LSIC's appeal versus DGK is in performing a lot less modular multiplications per invocation. However, experimentally we found that the number of rounds needed by LSIC (linear in the size of the input) is a bottleneck in practical uses, and therefore we stick to only using DGK.

5.1.2 Comparison with encrypted inputs

To develop our protocols, we require the ability to compare two encrypted inputs. More specifically, suppose that party A wants to compare two encrypted l bit unsigned integers a and b, but party B holds the decryption key. Protocol 1 is a new protocol for this scenario, which is based mostly on [Veu11] (*c.f.* Section 2.1).

Protocol 1 Comparing encrypted data

Input A: $[\![a]\!]$ and $[\![b]\!]$, the bit length l of a and b, the secret key SK_{QR}, public key PK_P
Input B: Secret key SK_P, public key PK_{QR}, the bit length l
Output A: $(a \leq b)$

A: $[\![x]\!] \leftarrow [\![b]\!] \cdot [\![2^l]\!] \cdot [\![a]\!]^{-1} \bmod N^2$
A chooses a random number $r \leftarrow (0, 2^{\lambda+l}) \cap \mathbb{Z}$
A: $[\![z]\!] \leftarrow [\![x]\!].[\![r]\!] \bmod N^2$ ▷ Blind x
A sends $[\![z]\!]$ to B
B decrypts $[\![z]\!]$
A: $c \leftarrow r \bmod 2^l$
B: $d \leftarrow z \bmod 2^l$
With A, B privately computes the encrypted bit $[t']$ such that $t = (d < c)$ using DGK
A encrypts r_l and sends $[r_l]$ to B
B encrypts z_l
B: $[t] \leftarrow [t'] \cdot [z_l] \cdot [r_l]$
B: sends $[t]$ to A
A decrypts and outputs t

Proposition 5.1. *Protocol 1 is correct and secure in the honest-but-curious model.*

5.1.3 Reversed comparison over encrypted data

In some cases, we want the result of the comparison to be revealed to the party that does not hold the encrypted data. We use Protocol 2 for this, which is Protocol 1 from [Veu11].

Protocol 2 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: $(a \leq b)$

A: $[\![x]\!] \leftarrow [\![b]\!] \cdot [\![2^l]\!] \cdot [\![a]\!]^{-1}$
A chooses a random number $r \leftarrow (0, 2^{\lambda+l}) \cap \mathbb{Z}$
A: $[\![z]\!] \leftarrow [\![x]\!] \cdot [\![r]\!]$ ▷ Blind x
A sends $[\![z]\!]$ to B
B decrypts $[\![z]\!]$
A: $c \leftarrow r \bmod 2^l$
B: $d \leftarrow z \bmod 2^l$
With B, A privately computes the encrypted bit $[t']$ such that $t' = (d < c)$ using DGK
B encrypts z_l and sends $[z_l]$ to A
A encrypts r_l
A: $[t] \leftarrow [t'] \cdot [z_l] \cdot [r_l]$
A: sends $[t]$ to B
B decrypts and outputs t

Proposition 5.2. *Protocol 2 is secure in the honest-but-curious model.*

5.1.4 Keeping the output secret

In some cases (*cf.* Section 8), the party not holding the QR secret key might want to keep the encryption of the result secret and not reveal it to the other party. To do so, we simply run Protocol 1 and 2 without sending the encrypted result $[t]$ to the party who holds the secret key. For these protocols, the proofs of security are identical to the ones for the original protocols.

5.1.5 Negative integers comparison and sign determination

So far, we assumed that the input integers a and b were unsigned. Indeed the protocols do not allow carry-overs; the algorithm would break because $2^l + b - a < 0$. To enable negative comparison, we change the protocol to use $l + 1$ instead of l as the exponent of 2; hence, we will always have $2^{l+1} + b - a \geq 0$.

We also need to compute the sign of an encrypted integer $[\![b]\!]$. In this case, we simply call Protocol 1 or 2 with $[\![a]\!]$ being an encryption of 0. Note that we do not need to use the $l + 1$ exponent trick described previously because $b + 2^l$ will always stay positive for a l bit signed integer.

5.2 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 biggest value), but neither party should learn anything else. In particular, B should not learn the order relations between a_i's. A simple way to achieve this is to have A randomly permute the k values. This way, B can compare the permuted a_i, get the maximum and send this index to A, who inverts the permutation to get the result. In the following, we denote $(a_i^\pi)_i$ the permuted family ($a_i^\pi = a_{\pi(i)}$).

However, B needs to hide from A the partial results of the comparison. This means that B needs to hide even the indices of the values its comparing. One way to do so is for B to compare every pair of inputs from A, but this would result in a quadratic number of comparisons.

Instead, our protocol only performs a linear number of comparisons by sequentially comparing the current element with the maximum element and updating the maximum element. The challenge is that party A knows which element becomes the maximum. To hide this information, we have A maintain the maximum in an encrypted form and B update the maximum according to the result of the comparisons without A being able to link the maximum to one of the values compared; this is achieved by having B *refresh* the encryption of the maximum.

The resulting protocol is shown in Protocol 3. Here, $\mathsf{Refresh}$ is the procedure to refresh 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.

Proposition 5.3. *Protocol 3 is correct and secure in the honest-but-curious model.*

5.3 Changing the encryption scheme

To enable us to compose various building blocks, we need to convert ciphertexts from one public-key encryption scheme to random ciphertexts of the same plaintext in another public-key encryption schemes, given only the public keys of the corresponding schemes.

In this section, we develop such a protocol. Suppose we have two additively homomorphic encryption schemes E_1 and E_2, both semantically secure with the same message space M. Once again we have two parties A and B. 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, in addition to a value encrypted with PK_1. Protocol 4 enables A to get an encryption of its value under E_2 without revealing anything to B.

Here, $[\![.]\!]_1$ is an encryption using E_1 and $[\![.]\!]_2$ is an encryption using E_2.

One can notice that, for some schemes, the message space M depends on the secret keys. It this case, we must be sure that party A can still uniformly pick elements of M without knowing it. For example, for Paillier, $M = \mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*$

Protocol 3 argmax over encrypted data

Input A: k encrypted integers $(\llbracket a_1 \rrbracket, \dots, \llbracket a_k \rrbracket)$, 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: $\mathrm{argmax}_i\, a_i$

```
A: chooses a random permutation π over {1, ..., k}
A: ⟦m⟧ ← ⟦a_π(1)⟧
B: i_0 ← 1
for i = 2 to k do
    Using Protocol 2, B gets the bit b_i = (m ≤ a_π(i))
    A picks two random integers r_i, s_i ← (0, 2^(λ+l)) ∩ ℤ
    A: ⟦m'_i⟧ ← ⟦m⟧ · ⟦r_i⟧                          ▷ m'_i = m + r_i
    A: ⟦a'_i⟧ ← ⟦a_π(i)⟧ · ⟦s_i⟧                     ▷ a'_i = a_π(i) + s_i
    A sends ⟦m'_i⟧ and ⟦a'_i⟧ to B
    if b_i is true then
        B: i_0 ← i
        B: ⟦v_i⟧ ← Refresh⟦a'_i⟧                     ▷ v_i = a'_i
    else
        B: ⟦v_i⟧ ← Refresh⟦m'_i⟧                     ▷ v_i = m'_i
    end if
    B sends to A ⟦v_i⟧
    B sends to A the couple (⟦x_i⟧, ⟦y_i⟧) = (⟦b̄_i⟧, ⟦b_i⟧)
    A: ⟦m⟧ ← ⟦v_i⟧ · ⟦x_i⟧^(−r_i) · ⟦y_i⟧^(−s_i)
                                                     ▷ m = v_i − x_i · r_i − y_i · t_i
end for
B sends i_0 to A
A outputs π^(−1)(i_0)
```

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

We use this protocol in the setting where $M = \{0, 1\}$ 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 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 (*cf.* Appendix C). Indeed, party A knows the randomness r in the clear.

5.4 Computing dot products

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

Proposition 5.5. *Protocol 5 is secure in the honest-but-curious model.*

5.5 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.

Protocol 4 Changing the encryption scheme

Input A: $[\![c]\!]_1$ and public keys PK_1 and PK_2
Input B: Secret keys SK_1 and SK_2
Output A: $[\![c]\!]_2$

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

Protocol 5 Private dot product

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

1: B encrypts $y_1, \dots, y_d$ and sends the encryptions $[\![y_i]\!]$ to A
2: A: $[\![v]\!] \leftarrow 1$ $\triangleright\ v \leftarrow 0$
3: **for** $i = 1$ **to** d **do**
4: A: $[\![v]\!] \leftarrow [\![v]\!] \cdot [\![y_i]\!]^{x_i} \mod N^2$ $\triangleright\ v \leftarrow v + x_i \cdot y_i$
5: **end for**
6: A re-randomizes and outputs $[\![v]\!]$

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). We must also take care of the bit length for the comparisons.

An example of a full analysis is given for the Naïve Bayes classifier in Appendix D.

6 Private hyperplane decision

For this classifier, we use a *one-versus-all* approach for hyperplane based classifier with k classes: the client has as input a feature vector x and the server has k different models $\{w_i\}_{i=1}^k$. Recall that classification result is given by

$$k^* = \operatorname*{argmax}_{i \in [k]} \langle w_i, x\rangle$$

To compute k^*, the client computes the encryption of $[\![\langle w_i, x\rangle]\!]$ for all $i \in [k]$ and then applies the argmax protocol (Protocol 3) to the encrypted dot products.

Proposition 6.1. *Protocol 6 is secure in the honest-but-curious model.*

7 Secure Naïve Bayes classifiers

7.1 General principle

Section 3.1 describes the Naïve Bayes classifier in some detail. Recall that each feature vector x is $x = (x_1, ..., x_d)$, with each of the x_i taking on discrete values. As is typically done for numerical stability reasons, we work with the

Protocol 6 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: $\underset{i\in[k]}{\operatorname{argmax}}\langle w_i, x\rangle$

1: **for** $i = 1$ **to** k **do**
2: C and S run Protocol 5 where C is party A with input is x and S is party B with input is w_i.
3: C gets $[\![v_i]\!]$ the result of the protocol. $\triangleright\ v_i \leftarrow \langle x, w_i\rangle$
4: **end for**
5: C and S run Protocol 3 where C is the Owner, and S the Helper, and $[\![v_1]\!], \ldots, [\![v_k]\!]$ the input cyphertexts. C gets the result i_0 of the protocol. $\triangleright\ i_0 \leftarrow \underset{i\in[k]}{\operatorname{argmax}}\, v_i$
6: C outputs i_0

logarithm of the probability distributions:

$$\begin{aligned} k^* &= \underset{i\in[k]}{\operatorname{argmax}}\ \log p(c_i|x) \\ &= \underset{i\in[k]}{\operatorname{argmax}} \left\{ \log p(c_i) + \sum_{j=1}^{d} \log p(x_j|c_i) \right\} \end{aligned} \tag{1}$$

Since the x_i's are discrete, we can look at the conditional probabilities as entries of a $D \times k$ table, where k is the number of categories and $D = \sum_{i=1}^{d} D_i$, with D_i as the number of possible outcomes (the domain size) of the variable x_i.

The general idea is that the server can encrypt this table under Paillier. For each class c_i, the client does $d + 1$ lookups in this table to get the entries $[\![\log p(c_i)]\!]$ and $[\![\log p(x_j|c_i)]\!]$ corresponding to its feature vector x and then computes an encryption $[\![\log p(x, c_i)]\!]$ for every $i \in [k]$ (there are $k(d + 1)$ lookups in total). Finally, the client runs the argmax protocol (Protocol 3).

7.2 Building a secure Naïve Bayes classifier

As noticed in Section 5.5, we need to work with integers. To do so, we prepare our data by multiplying all the probabilities by a constant factor K and then we apply the algorithm described in Section 7.1 with these new values.

We prepare $kd + 1$ tables:

- One table for the priors on the classes T^C: $T^C(i) \approx \log p(c_i)$
- One table per feature per class T^j: $T^j(x_j, i) \approx \log p(x_j|c_i)$

We refer the reader to Appendix D for more precise explanations.

Once the probability table is converted to integers, we can use Paillier's additively homomorphic cryptosystem to do the computations: for every class c_i, the client computes

$$[\![p_i]\!] = [\![T^C(i)]\!] \prod_{j=1}^{d} [\![T^j(x_j, i)]\!]$$

where x_j is value for the j-th feature. Finally, the client runs Protocol 3 to get $\operatorname{argmax} p_i$. Given the fact that Protocol 3 is secure and Paillier cryptosystem is semantically secure, the security of this classifier is trivial. For completeness, the protocol is shown in Protocol 7.

Proposition 7.1. *Protocol 7 is secure in the honest-but-curious model.*

Protocol 7 Naïve Bayes Classifier

Client's (C) Input: $x = (x_1, \ldots, x_d) \in \mathbb{Z}^d$, public key PK_P, secret key SK_{QR}

Server's (S) Input: The secret key SK_P, public key PK_{QR} and probability tables $\{\log p(c_i)\}_{1\le i\le k}$ and $\left\{\left\{\log p(x_j^{(l)}|c_i)\right\}_{x_j^{(l)}\in X_j}\right\}_{1\le j\le d, 1\le i\le k}$

Client's Output: i_0 such that $p(x, c_{i_0})$ is maximum

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

8 Oblivious decision trees

The goal here is to design a protocol that allows the server to traverse a binary decision tree using an element x input from the client in a way that hides the client's input from the server, and hides both the structure of the tree and the thresholds from the client.

8.1 Polynomial form of a decision tree

Our protocol works off the following observation: classifying using a binary decision tree is the same thing as evaluating a simple multi-variate polynomial. As before, we associate a condition to a boolean variable that is 1 if the condition is true, and 0 otherwise. In other words, we can view the decision tree from Figure 2 as the same as the tree of Figure 3. Note that $c_1, \ldots, c_5$ correspond to the class labels. The 4-variables polynomial associated with this tree is

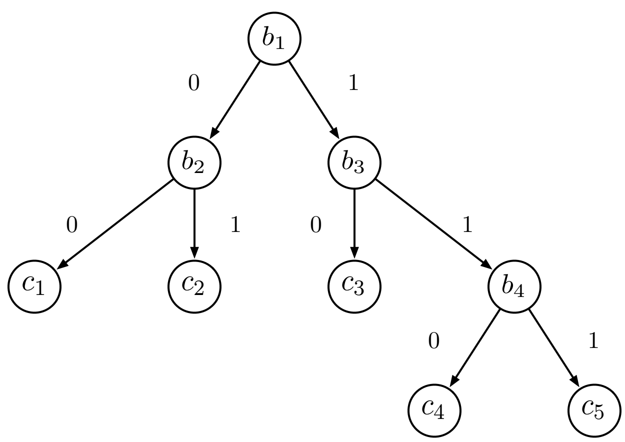
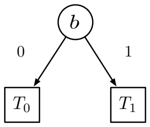


Figure 3: Decision tree with booleans

$$
\begin{aligned}
&P(b_1, b_2, b_3, b_4) = \\
&\quad b_1 \left[b_3 \cdot (b_4 \cdot c_5 + (1 - b_4) \cdot c_4) + (1 - b_3) \cdot c_3\right] \\
&\qquad + (1 - b_1) \left[b_2 \cdot c_2 + (1 - b_2) \cdot c_1\right]
\end{aligned}
$$

We can build a simple recursive procedure $\mathcal{F}$ to build the polynomial form of a binary decision tree T

If T is a leaf with category index c_i, $\mathcal{F}(T) = c_i$.



If T is an inner node using boolean b as its associated condition and T_0 and T_1 its two subtrees, then $\mathcal{F}(T) = b \cdot \mathcal{F}(T_1) + (1 - b) \cdot \mathcal{F}(T_0)$.

Hence, given a tree T with n nodes, this procedure outputs a polynomial P with overall degree roughly $\log_2 n$. We now focus on privately evaluating this polynomial.

8.2 Private evaluation of a polynomial

We reduced our problem to the private evaluation of an n variables polynomial. Remember that the values of the boolean variables must remain unknown to the server (the owner of the tree). Therefore, the server has to do the computation over encrypted data.

More formally, suppose the server has the encryptions $[\![b_1]\!], \ldots, [\![b_n]\!]$ of the n variables under a fully-homomorphic encryption scheme and a polynomial P corresponding to its decision tree with k possible categories. It then has to compute $P([\![b_1]\!], \ldots, [\![b_n]\!]) = [\![P(b_1, \ldots, b_n)]\!]$ and return the (re-randomized) result to the client. Technically, we do not need a *real* FHE scheme; a leveled homomorphic scheme is enough as the depth of the tree – hence the degree of the polynomial – remains quite small.

One last note is that FHE schemes are faster over a binary $\mathbb{F}_2$ message space, but the result of P is usually more than one bit in length. To overcome this limitation, we use SIMD slots with FHE (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, instead of one polynomial P, we create $l = \lceil \log_2 k \rceil$ (k is the number of categories) polynomials P_j where P_j computes the j-th bit of the result given by P. Then, with one FHE evaluation, we evaluate all P_j's at once, without any additional computational overhead, instead of computing the $\log k$ polynomials one-by-one. In practice, it means improving the performances by a factor at least 2 when classifying in more than two categories.

When using a (leveled) FHE scheme, we must be aware of the depth of the circuit we are computing. In our case, by cleverly regrouping multiplications pairwise, we can reduce a polynomial P of degree d to a $\log d$ depth circuit.

8.3 Formal description

Protocol 8 describes the resulting protocol. See Section 4 for the FHE notation.

Protocol 8 Decision Tree Classifier

Client's (C) Input: $x = (x_1, \ldots, x_n) \in \mathbb{Z}^n$, secret keys $\mathsf{SK}_{QR}, \mathsf{SK}_{FHE}$
Server's (S) Input: The public keys $\mathsf{PK}_{QR}, \mathsf{PK}_{FHE}$, the model as a decision tree, with n boolean inputs, the boolean inputs $[b_1], \ldots, [b_n]$ encrypted under QR
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 8.1 (including the SIMD slots).
2: Using Protocol 4, S gets the encryptions $[\![b_1]\!], \ldots, [\![b_n]\!]$ of $b_1, \ldots, b_n$ under the FHE scheme, homomorphically computes $[\![P_0(b_1, \ldots, b_n), \ldots, P_{l-1}(b_1, \ldots, b_n)]\!]$ and sends the result to the client.
3: 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 8.1. *Protocol 8 is secure in the honest-but-curious model.*

```
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

9 Combining classifiers with AdaBoost

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

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

If each of the $h_i(\cdot)$'s is an instance of a classifier supported by our protocols, then given the scalars α_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 (*e.g.* using Section 5.1.4). Second, if necessary, we convert them to Paillier's encryption scheme with Protocol 4, and combine these intermediate results using Paillier's additive homomorphic property. Finally, we run the comparison of encrypted data algorithm, so that the client gets the final result.

10 Implementation

We have implemented the protocols and the classifiers in C++ using GMP[2], Boost, Google's Protocol Buffers[3], and HELib [Hal13] for the FHE implementation.

The code is written in a modular way: all the elementary protocols defined in Section 5 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 the key exchange protocol, followed by the dot product protocol, and then the comparison of encrypted data protocol to output the result, as shown in Figure 4.

11 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 11.1), (ii) what is the performance overhead of our building blocks (Section 11.3), and (iii) what is the performance overhead of our classifiers (Section 11.4)?

11.1 Using our building blocks library

Here we demonstrate that our building blocks library can be used to build other classifiers modularly and is a useful contribution by itself. We will construct a multiplexer and a face detector. A face detection algorithm over encrypted

[2] http://gmplib.org/
[3] https://code.google.com/p/protobuf/

data already exists [AB06, AB07], so our construction here is not the first such construction, but it serves as a proof of functionality of our library.

11.1.1 Building a multiplexer

It is often useful to compute the following generalized comparison function:

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

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

$$f_{\alpha,\beta}(b) = b \cdot \beta + (1-b) \cdot \alpha = \alpha + b \cdot (\beta - \alpha)$$

If we have the encryption $[\![b]\!]$ of b under Paillier, and we know both α and β, using Paillier's scheme homomorphism, we can compute an encryption of $f_{\alpha,\beta}(b)$:

$$[\![f_{\alpha,\beta}(b)]\!] = [\![\alpha]\!] \cdot [\![b]\!]^{\beta-\alpha}$$

Hence, we just need to compare a and b, keeping the result encrypted with QR, and then change the encryption scheme (*cf.* Section 5.3) to get $b = (a \leq b)$ encrypted with Paillier.

11.1.2 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

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

where the h_t are weak classifiers of the form

$$h_i(x) = sign\left(\langle x, y_i\rangle - \theta_i\right)$$

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 11.1.1, we can easily compute the encrypted evaluation of every one of these weak classifiers under Paillier, and then, as described in Section 9, compute the encryption of $H(x)$.

11.2 Performance evaluation setup

Our performance evaluations were run using two desktop computers each with identical configuration: two Intel i7 (64 bit) processors for a total 8 cores running at 3.4 GHz and 8 GB RAM. Since the machines were on the same network, we artificially inflated the roundtrip time for a packet to be 40 ms to mimic real network latency. We used 1024 bit crypto keys, and chose the statistical security parameter λ to be 100. When using HELib, we use 80 bits of security, which corresponds to a 1024 bit asymmetric key.

11.3 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).

11.3.1 Comparison protocols

Comparison with unencrypted input. Table 3 gives the running time of the DGK comparison protocol with unencrypted input for various input size. The DGK protocol runs in parallel using four threads for each party.

Bit size	A Computation	B Computation	Total Time	Communication	Interactions
10	5.03 ms	12.38 ms	98.4 ms	5.26 kB	3
20	8.34 ms	17.3 ms	107 ms	10.4 kB	3
32	13.70 ms	17.58 ms	112 ms	16.5 kB	3
64	26.15 ms	39.03 ms	149 ms	32.9 kB	3

Table 3: Comparison with unencrypted input protocols evaluation

Comparison with encrypted input. Table 4 presents the performance of the comparison with encrypted inputs protocols, with DGK as underlying comparison protocols.

11.3.2 argmax

Figure 5 presents the running times and the communication overhead of the argmax of encrypted data protocol (*cf.* Section 5.2). The input integers were 64 bit integers.

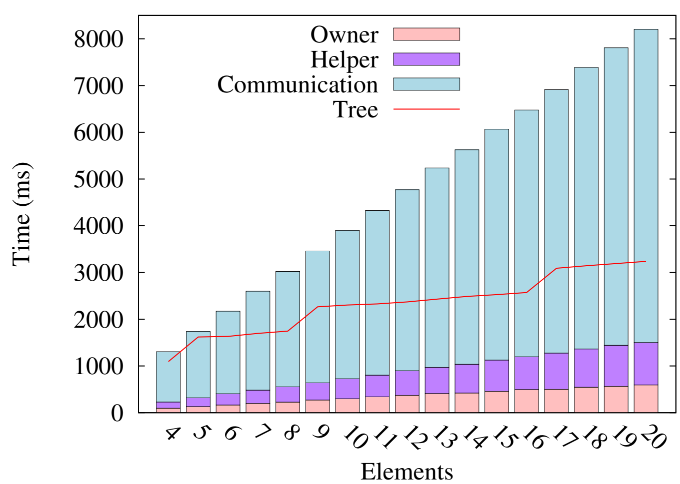


Figure 5: Argmax of encrypted data protocol evaluation. The bars represent the execution of the protocol when the comparisons are executed one after each other, linearly. The line represents the execution when comparisons are executed in parallel, tree-wise.

Protocol	Bit size	Computation Owner	Computation Helper	Total Time	Communication	Interactions
Comparison	64	38.07 ms	20.99 ms	258.5 ms	33.41 kB	6
Reversed Comp.	64	25.83 ms	36.93 ms	292.1 ms	33.41 kB	6

Table 4: Comparison with encrypted input protocols evaluation

Owner Computation	Helper Computation	Total Time	Communication	Interactions
47.0 ms	232 ms	549 ms	481.2 kB	2

Table 5: Change encryption scheme protocol evaluation

Data set	Model size	Computation		Time per protocol[4]		Total running time	Comm.	Interactions
		Client	Server	Compare	Dot product			
Breast cancer (1)	30	52.0 ms	35.7 ms	281 ms	45.4 ms	326 ms	41.35 kB	7
Credit (3)	47	55.8 ms	44.0 ms	275 ms	48.2 ms	323 ms	45.70 kB	7

(a) Linear Classifier

Data set	Specs.		Computation		Time per protocol[4]		Total running time	Comm.	Interactions
	C	F	Client	Server	Prob. Comp.	Argmax			
Breast Cancer (2)	2	9	122 ms	148 ms	82.1 ms	469 ms	551 ms	77.26 kB	14
Nursery (5)	5	9	434 ms	393 ms	82.9 ms	1563 ms	1646 ms	171.1 kB	42
Audiology (4)	24	70	1557 ms	1931 ms	636 ms	3388 ms	4024 ms	2067 kB	166

(b) Naïve Bayes Classifier. C is the number of categories and F is the number of features. The Prob. Comp. column corresponds to the computation of the probabilities $p(c_i|x)$ (*cf.* Section 7).

Data set	Tree Specs.		Computation		Time per protocol[4]		FHE		Comm.	Interactions
	N	D	Client	Server	Compare	ES Change	Eval.	Decrypt		
Nursery (5)	4	4	991 ms	1756 ms	1474 ms	1709 ms	182 ms	1443 ms	3210 kB	30
ECG (6)	6	4	1485 ms	2595 ms	2309 ms	2627 ms	689 ms	2005 ms	4272 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.

Table 6: Classifiers evaluation

11.4 Classifier performance

Here we evaluate each of the classifiers described in Sections 6–8. The models are trained non-privately using `scikit-learn`[5]. 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
6. ECG (electrocardiogram) classification data from Barni *et al.* [BFK+09]

These data sets represent some typical cases where we want to ensure privacy of both the server's model and client's input.

More specifically, we used 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 very practical for sensitive applications.

[4]Including communication

[5]`http://scikit-learn.org`

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 1765 ms[6] for the client and 4235 ms for the server with communication cost of 112.2KB. Even though their evaluation does not consider the communication delays, we are still twice as fast for the server and faster for the client. Moreover, we must not forget that we keep the tree private while Barni *et al.* construction reveals the computation circuit (by revealing the garbled circuit).

12 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 also demonstrated the efficiency of our classifiers on real datasets.

References

[AB06] Shai Avidan and Moshe Butman. Blind vision. In *Computer Vision–ECCV 2006*, pages 1–13. Springer, 2006.

[AB07] Shai Avidan and Moshe Butman. Efficient methods for privacy preserving face detection. In *Advances in Neural Information Processing Systems: Proceedings of the 2006 Conference*, volume 19, page 57. The MIT Press, 2007.

[AD01] Mikhail J Atallah and Wenliang Du. Secure multi-party computational geometry. In *Algorithms and Data Structures*, pages 165–179. Springer, 2001.

[BDMN05] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: the sulq framework. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 128–138. ACM, 2005.

[BFK+09] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In *Computer Security–ESORICS 2009*, pages 424–439. Springer, 2009.

[BFL+09] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Annika Paus, A-R Sadeghi, Thomas Schneider, and Vladimir Kolesnikov. Efficient privacy-preserving classification of ecg signals. In *Information Forensics and Security, 2009. WIFS 2009. First IEEE International Workshop on*, pages 91–95. IEEE, 2009.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[BL13] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[BLN13] Joppe W. Bos, Kristin Lauter, and Michael Naehrig. Private predictive analysis on encrypted medical data, 2013.

[BN06] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.

[Can98] Ran Canetti. Security and composition of multi-party cryptographic protocols. *JOURNAL OF CRYPTOLOGY*, 13:2000, 1998.

[CMS11] Kamalika Chaudhuri, Claire Monteleoni, and Anand D. Sarwate. Differentially private empirical risk minimization. *J. Mach. Learn. Res.*, 12, 2011.

[6] In Barni *et al.* [BFK+09], the evaluation was run over two 3GHz computers directly connected via Gigabit Ethernet. We scaled the given results by $\frac{3}{3.4}$ to get a better comparison basis.

[DGK07] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Efficient and secure comparison for on-line auctions. In *Information Security and Privacy*, pages 416–430. Springer, 2007.

[DGK09] Ivan Damgard, Martin Geisler, and Mikkel Kroigard. A correction to'efficient and secure comparison for on-line auctions'. *International Journal of Applied Cryptography*, 1(4):323–324, 2009.

[DHC04] Wenliang Du, Yunghsiang S Han, and Shigang Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *Proceedings of the 4th SIAM International Conference on Data Mining*, volume 233. Lake Buena Vista, Florida, 2004.

[EFG+09] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In *Privacy Enhancing Technologies*, pages 235–253. Springer, 2009.

[FS97] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[GLLM05] Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. On private scalar product computation for privacy-preserving data mining. In *Information Security and Cryptology–ICISC 2004*, pages 104–120. Springer, 2005.

[GLN13] Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. In *Information Security and Cryptology–ICISC 2012*, pages 1–21. Springer, 2013.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377. ACM, 1982.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, pages 218–229, 1987.

[Gol04] Oded Goldreich. *Foundations of Cryptography - Basic Applications*. Cambridge University Press, 2004.

[Hal13] Shai Halevi. Helib - an implementation of homomorphic encryption. `https://github.com/shaih/HElib`, 2013.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer – efficiently. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 572–591. Springer Berlin Heidelberg, 2008.

[Kil05] Eike Kiltz. Unconditionally secure constant round multi-party computation for equality, comparison, bits and exponentiation. *IACR Cryptology ePrint Archive*, 2005:66, 2005.

[LLM06] Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. Cryptographically private support vector machines. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 618–624. ACM, 2006.

[LP00] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Advances in Cryptology—CRYPTO 2000*, pages 36–54. Springer, 2000.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer Berlin Heidelberg, 2007.

[LP08] Yehuda Lindell and Benny Pinkas. Secure multiparty computation for privacy-preserving data mining, 2008.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptol.*, 22:161–188, April 2009.

[LT05] Hsiao-Ying Lin and Wen-Guey Tzeng. An efficient solution to the millionaires' problem based on homomorphic encryption. In *Applied Cryptography and Network Security*, pages 456–466. Springer, 2005.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay-secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[RR07] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NIPS*, 2007.

[SG11] Anima Singh and John Guttag. Cardiovascular risk stratification using non-symmetric entropy-based classification trees. In *NIPS workshop on personalized medicine*, 2011.

[SG13] Anima Singh and John Guttag. Leveraging hierarchical structure in diagnostic codes for predicting incident heart failure. In *ICML workshop on role of machine learning in transforming healthcare*, 2013.

[SSW10] Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Efficient privacy-preserving face recognition. In *Information, Security and Cryptology–ICISC 2009*, pages 229–244. Springer, 2010.

[SV11] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. Cryptology ePrint Archive, Report 2011/133, 2011.

[Veu11] Thijs Veugen. Comparing encrypted data. `http://msp.ewi.tudelft.nl/sites/default/files/Comparing%20encrypted%20data.pdf`, 2011.

[VJ01] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[VKC08] Jaideep Vaidya, Murat Kantarcıoğlu, and Chris Clifton. Privacy-preserving naive bayes classification. *The International Journal on Very Large Data Bases*, 17(4):879–898, 2008.

[WGH12] Jenna Wiens, John Guttag, and Eric Horvitz. Learning evolving patient risk processes for c. diff colonization. In *ICML*, 2012.

[WY04] Rebecca Wright and Zhiqiang Yang. Privacy-preserving bayesian network structure computation on distributed heterogeneous data. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 713–718. ACM, 2004.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.

[ZW05] Zhiqiang Yang1 Sheng Zhong and Rebecca N Wright. Privacy-preserving classification of customer data without loss of accuracy. In *SIAM International Conference on Data Mining (SDM), Newport Beach*. Citeseer, 2005.

A Preliminaries for proofs

A.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 Π 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 Π and with the security parameter λ. The view of party A during the execution of Π 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 Π on input $(a.b)$ as $\mathsf{Output}_A^\Pi(\lambda, a, b)$ and $\mathsf{Output}_B^\Pi(\lambda, a, b)$, and the global output as $\mathsf{Output}^\Pi(\lambda, a, b) = (\mathsf{Output}_A^\Pi(\lambda, a, b), \mathsf{Output}_B^\Pi(\lambda, a, b))$.

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 A.1. *The two-party protocol* Π 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*,*

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

and

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

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

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 Π 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.

A.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 Π and use calls to an ideal functionality f in Π (*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 Π respects privacy in the honest-but-curious model and if we have a protocol ρ that privately computes f in the same model, then we can replace the ideal calls for f by the execution of ρ in Π; the new protocol, denoted Π^ρ 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 Π 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 Π 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 Π to the trusted party for f_i is replaced by a real execution of ρ_i: if party P_j has to compute f_i with input x_j, P_j halts, starts an execution of ρ_i with the other parties, gets the result β_j when ρ_i concludes, and continues as if β_j was received from T.

Theorem A.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* Π *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.*

A.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 $\mathbb{QNR}_N$ be the set of quadratic non residues (*i.e. *$x \in \mathbb{QNR}_N$ if x is not a square modulo N and its Jacobi symbol is 1).*

$\{(N, \mathbb{QR}_N) : |N| = \lambda\}$ and $\{(N, \mathbb{QNR}_N) : |N| = \lambda\}$ 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 $y \in \mathbb{Z}_{N^2}$*

$$z = y^N \bmod N^2$$

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 λ, 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 χ be a distribution over R. The $\mathsf{RLWE}_{d,q,\chi}$ 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 $\mathsf{RLWE}_{d,q,\chi}$ assumption is that the $\mathsf{RLWE}_{d,q,\chi}$ problem is infeasible.

B Proofs

B.1 Comparison protocols

B.1.1 Encrypted comparison

Proof of Proposition 5.1 . **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 1 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 $x = 2^l(x \div 2^l) + (x \bmod 2^l)$ where $0 \leq (x \bmod 2^l) < 2^l$. As $z = x + r$,

$$\begin{aligned} z &= 2^l(z \div 2^l) + (z \bmod 2^l) \\ &= 2^l((x \div 2^l) + (r \div 2^l)) + ((x \bmod 2^l) + (r \bmod 2^l)) \end{aligned}$$

Hence, $z \div 2^l = x \div 2^l + r \div 2^l$ if $(x \bmod 2^l) + (r \bmod 2^l) < 2^l$ 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 $t' = 0 \Leftrightarrow (x \bmod 2^l) + (r \bmod 2^l) < 2^l$.

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

$$\begin{aligned} t' = 0 &\Leftrightarrow z \bmod 2^l = (x \bmod 2^l) + (r \bmod 2^l) \\ &\Leftrightarrow z \bmod 2^l \geq (r \bmod 2^l) \end{aligned}$$

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

$$\begin{aligned} x \div 2^l &= (z \div 2^l) - (r \div 2^l) - t' \mod 2 \\ &= z_l \oplus r_l \oplus t' \end{aligned}$$

The mistake in [Veu11], resided in that very last line: for all integers v and i, $(v \div 2^i) \bmod 2 = v_i$ and not v_{i+1}.

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 $V_A = (\llbracket a \rrbracket, \llbracket b \rrbracket, l, \mathsf{SK}_{QR}, \mathsf{PK}_P; r, \mathsf{coins}; [t])$ 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 $(\llbracket a \rrbracket, \llbracket b \rrbracket, l, \mathsf{SK}_{QR}, \mathsf{PK}_P, a \leq b)$, we build the simulator S_A:

1. Compute $[\tilde{t}]$ an encryption of the bit $(a \leq b)$ under QR.
2. Pick $\tilde{r} \leftarrow (0, 2^{\lambda+l}) \cap \mathbb{Z}$.
3. Let $\widetilde{\mathsf{coins}}$ be random coins for two Paillier encryptions and one QR encryption.
4. Output $(\llbracket a \rrbracket, \llbracket b \rrbracket, l, \mathsf{SK}_{QR}, \mathsf{PK}_P; \tilde{r}, \widetilde{\mathsf{coins}}; [\tilde{t}])$

The distributions $V_A(\llbracket a \rrbracket, \llbracket b \rrbracket, l, \mathsf{SK}_{QR}, \mathsf{PK}_{QR}, \mathsf{SK}_P, \mathsf{PK}_P)$ and $S_A(\llbracket a \rrbracket, \llbracket b \rrbracket, \mathsf{SK}_{QR}, \mathsf{PK}_P, a \leq b)$ 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 $V_B = (\mathsf{PK}_{QR}, \mathsf{SK}_P, l, \llbracket z \rrbracket; \mathsf{coins}; [t'], [r_l])$ where coins are the random coins used for the encryption of z_l. The simulator $S_B(\mathsf{PK}_{QR}, \mathsf{SK}_P, l)$ runs as follows:

1. Pick $\tilde{z} \leftarrow (0, 2^{\lambda+l}) \cap \mathbb{Z}$.
2. Encrypt $\tilde{z}$ under Paillier: $\llbracket \tilde{z} \rrbracket$.
3. Generate $[\tilde{t}']$ and $[\tilde{r}_l]$, two encryptions of random bits under QR
4. Let $\widetilde{\mathsf{coins}}$ be random coins for one QR encryption.
5. Output $(\mathsf{PK}_{QR}, \mathsf{SK}_P, l, \llbracket \tilde{z} \rrbracket; \widetilde{\mathsf{coins}}; [\tilde{t}'], [\tilde{r}_l])$

The random tapes coins and $\widetilde{\mathsf{coins}}$ are generated in the exact same manner and independently from any other parameter, so

$$\begin{aligned} &(\mathsf{PK}_{QR}, \mathsf{SK}_P, \llbracket \tilde{z} \rrbracket; \widetilde{\mathsf{coins}}; [\tilde{t}'][\tilde{r}_l]) \\ &\quad = (\mathsf{PK}_{QR}, \mathsf{SK}_P, \llbracket \tilde{z} \rrbracket; \mathsf{coins}; [\tilde{t}'][\tilde{r}_l]) \end{aligned}$$

Recall that $z = x + r \bmod N$ 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 $\tilde{z}$ 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 $(\mathsf{SK}_P, \llbracket \tilde{z} \rrbracket) \equiv_s (\mathsf{SK}_P, \llbracket z \rrbracket)$ and as a consequence, as the distribution of $\tilde{z}$ and z is independent from $\tilde{t}'$ and $\tilde{r}_l$,

$$\begin{aligned} &(\mathsf{PK}_{QR}, \mathsf{SK}_P, \llbracket \tilde{z} \rrbracket; \mathsf{coins}; [\tilde{t}'], [\tilde{r}_l]) \\ &\quad \equiv_s (\mathsf{PK}_{QR}, \mathsf{SK}_P, \llbracket z \rrbracket; \mathsf{coins}; [\tilde{t}'], [\tilde{r}_l]) \end{aligned}$$

By semantic security of QR,

$$\begin{aligned} &(\mathsf{PK}_{QR}, \mathsf{SK}_P, l, \llbracket z \rrbracket; \mathsf{coins}; [\tilde{t}'], [\tilde{r}_l]) \\ &\quad \equiv_c (\mathsf{PK}_{QR}, \mathsf{SK}_P, l, \llbracket z \rrbracket; \mathsf{coins}; [t'], [r_l]) \end{aligned}$$

and

$$\begin{aligned} &S_B(\mathsf{PK}_{QR}, \mathsf{SK}_P, l) \\ &\quad \equiv_c V_B(\llbracket a \rrbracket, \llbracket b \rrbracket, l, \mathsf{SK}_{QR}, \mathsf{PK}_{QR}, \mathsf{SK}_P, \mathsf{PK}_P) \end{aligned}$$

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 A.2 to prove security in the semi-honest model. $\square$

B.1.2 Reversed encrypted comparison

Proof of Proposition 5.2. The proof of security is similar to the one of Proposition 5.1. Again we first suppose that $[t']$ is ideally computed (hybrid model).

A's view is $V_A = (\llbracket a \rrbracket, \llbracket b \rrbracket, l, \mathsf{PK}_{QR}, \mathsf{PK}_P; r, \mathsf{coins}; [t'], [z_l])$ where PK_{QR} is the public key for the QR cryptosystem, 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_l.

Given $(\llbracket a \rrbracket, \llbracket b \rrbracket, \mathsf{PK}_{QR}, \mathsf{PK}_P)$, we build the simulator S_A:

1. Pick $\tilde{r} \leftarrow (0, 2^{\lambda+l}) \cap \mathbb{Z}$.
2. Generate $[\tilde{t}']$ and $[\tilde{z}_l]$, two encryptions of random bits under QR
3. Let $\widetilde{\mathsf{coins}}$ be random coins for two Paillier encryptions and one QR encryption.
4. Output $(\llbracket a \rrbracket, \llbracket b \rrbracket, l, \mathsf{PK}_{QR}, \mathsf{PK}_P; \tilde{r}, \widetilde{\mathsf{coins}}; [\tilde{z}_l])$

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

$$\begin{aligned} &S_A(\llbracket a \rrbracket, \llbracket b \rrbracket, \mathsf{PK}_{QR}, \mathsf{PK}_P) \\ &\quad = (\llbracket a \rrbracket, \llbracket b \rrbracket, l, \mathsf{PK}_{QR}, \mathsf{PK}_P; r, \mathsf{coins}; [\tilde{z}_l]) \end{aligned}$$

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

$$\begin{aligned} &S_A(\llbracket a \rrbracket, \llbracket b \rrbracket, \mathsf{PK}_{QR}, \mathsf{PK}_P) \\ &\quad = (\llbracket a \rrbracket, \llbracket b \rrbracket, l, \mathsf{PK}_{QR}, \mathsf{PK}_P; r, \mathsf{coins}; [\tilde{z}_l]) \\ &\quad \equiv_c (\llbracket a \rrbracket, \llbracket b \rrbracket, l, \mathsf{PK}_{QR}, \mathsf{PK}_P; r, \mathsf{coins}; [z_l]) \\ &\quad = V_A(\llbracket a \rrbracket, \llbracket b \rrbracket, l, \mathsf{SK}_{QR}, \mathsf{PK}_{QR}, \mathsf{SK}_P, \mathsf{PK}_P) \end{aligned}$$

B's view is $V_B = (\mathsf{SK}_{QR}, \mathsf{SK}_P, \llbracket z \rrbracket, [t]; \mathsf{coins})$ where SK_{QR} is the secret key for the QR cryptosystem, SK_P is the secret key for Paillier's cryptosystem, and coins are the random coins necessary for the QR encryption of z_l. The simulator $S_B(\mathsf{SK}_{QR}, \mathsf{SK}_P, a \leq b)$ runs as follows:

1. Compute $[\tilde{t}]$ an encryption of the bit $(a \leq b)$ under QR.
2. Pick $\tilde{z} \leftarrow (0, 2^{\lambda+l}) \cap \mathbb{Z}$.
3. Encrypt $\tilde{z}$ under Paillier: $\llbracket \tilde{z} \rrbracket$.
4. Let $\widetilde{\mathsf{coins}}$ be random coins for one QR encryption.
5. Output $(\mathsf{SK}_{QR}, \mathsf{SK}_P, l, \llbracket \tilde{z} \rrbracket, [\tilde{t}]; \widetilde{\mathsf{coins}})$

Once again, the distributions of coins and $\widetilde{\mathsf{coins}}$ are identical:

$$\begin{aligned} &(\mathsf{SK}_{QR}, \mathsf{SK}_P, l, \llbracket \tilde{z} \rrbracket, [\tilde{t}]; \widetilde{\mathsf{coins}}) \\ &\quad = (\mathsf{SK}_{QR}, \mathsf{SK}_P, l, \llbracket \tilde{z} \rrbracket, [\tilde{t}]; \mathsf{coins}) \end{aligned}$$

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 $(\mathsf{SK}_P, \llbracket \tilde{z} \rrbracket) \equiv_s (\mathsf{SK}_P, \llbracket z \rrbracket)$ and as a consequence, as the distribution of $\tilde{z}$ and z is independent from $\tilde{t}'$,

$$\begin{aligned}&(\mathsf{SK}_{QR}, \mathsf{SK}_P, l, [\![\tilde{z}]\!], [\tilde{t}]; \mathsf{coins})\\ &\quad \equiv_s (\mathsf{SK}_{QR}, \mathsf{SK}_P, l, [\![z]\!], [\tilde{t}]; \mathsf{coins})\end{aligned}$$

Moreover, by construction, $(\mathsf{SK}_{QR}, [\tilde{t}]) = (\mathsf{SK}_{QR}, [a < b])$ and

$$\begin{aligned}&(\mathsf{SK}_{QR}, \mathsf{SK}_P, l, [\![z]\!], [\tilde{t}]; \mathsf{coins})\\ &\quad = (\mathsf{SK}_{QR}, \mathsf{SK}_P, l, [\![z]\!], [a < v]; \mathsf{coins}).\end{aligned}$$

Finally, we have

$$\begin{aligned}&S_B(\mathsf{SK}_{QR}, \mathsf{SK}_P, a \le b)\\ &\quad \equiv_s V_B([\![a]\!], [\![b]\!], l, \mathsf{SK}_{QR}, \mathsf{PK}_{QR}, \mathsf{SK}_P, \mathsf{PK}_P).\end{aligned}$$

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 A.2 to prove security in the semi-honest model. □

B.2 Argmax

Proof of Proposition 5.3. **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 $\{a_{\pi(j)}\}_{1\le j\le i}$ and $a_{\pi(i_0)} = m$.

It this holds, at the end of the loop iterations $a_{\pi(i_0)}$ is the maximum of $\{a_{\pi(j)}\}_{1\le j\le k} = \{a_j\}_{1\le j\le k}$, hence $i_0 = \operatorname{argmax}_j a_{\pi(j)}$ and $\pi^{-1}(i_0) = \operatorname{argmax}_j a_j$.

At initialization (line 4), the invariant trivially holds as the family $\{a_{\pi(j)}\}_{1\le j\le i}$ 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.* $m \le a_{\pi(i)}$), $\mathsf{max}\{a_{\pi(j)}\}_{1\le j\le i-1} \le a_{\pi(i)}$, as the invariant holds for the previous iteration, and then $\mathsf{max}\{a_{\pi(j)}\}_{1\le j\le i} = a_{\pi(i)}$.
  Then i_0 is set to i, $v_i = a'_i$ and $(x_i, y_i) = (0, 1)$. As a consequence, m is set by A to

  $$v_i - x_i.r_i - y_i.s_i = a'_i - s_i = a_{\pi(i)}$$

  We have clearly that $a_{\pi(i_0)} = a_{\pi(i)} = m$ and $m = \mathsf{max}\{a_{\pi(j)}\}_{1\le j\le i}$, the invariant holds at the end of the i-th iteration in this case.
- If b_i is false ($m > a_{\pi(i)}$), $\mathsf{max}\{a_{\pi(j)}\}_{1\le j\le i-1} > a_{\pi(i)}$ and $\mathsf{max}\{a_{\pi(j)}\}_{1\le j\le i} = \mathsf{max}\{a_{\pi(j)}\}_{1\le j\le i-1} = m$.
  Then i_0 is not changed, v_i is set to m'_i and $(x_i, y_i) = (1, 0)$. As a consequence,

  $$v_i - x_i.r_i - y_i.s_i = m'_i - r_i = m$$

  m is unchanged. As both m and i_0 stayed the same and $\mathsf{max}\{a_{\pi(j)}\}_{1\le j\le i} = \mathsf{max}\{a_{\pi(j)}\}_{1\le j\le i-1}$, the invariant holds at the end of the i-th iteration in this case.

Security To prove security, we first consider that line 5 of the protocol is ideally executed: we ask a trusted party T to compute the function $f([\![x]\!], [\![y]\!], l, \mathsf{SK}_{QR}, \mathsf{PK}_{QR}, \mathsf{SK}_P, \mathsf{PK}_P)$ in the f-hybrid model where

$$\begin{aligned}&f([\![x]\!], [\![y]\!], l, \mathsf{SK}_{QR}, \mathsf{PK}_{QR}, \mathsf{SK}_P, \mathsf{PK}_P)\\ &\quad = \begin{array}{l}(f_A(x, y, l, \mathsf{SK}_{QR}, \mathsf{PK}_{QR}, \mathsf{SK}_P, \mathsf{PK}_P);\\ \ f_B([\![x]\!], [\![y]\!], l, \mathsf{SK}_{QR}, \mathsf{PK}_{QR}, \mathsf{SK}_P, \mathsf{PK}_P))\end{array}\end{aligned}$$

and f computes the function of Protocol 2, *i.e.* f_A returns nothing and f_B returns the bit $x \le y$.

We will conclude using Theorem A.2.

A's view is

$$\begin{aligned}V_A =&(\{[\![a_i]\!]\}_{i=1}^k, l, \mathsf{PK}_{QR}, \mathsf{PK}_P;\\ &\pi, \{r_i\}_{i=2}^k, \{s_i\}_{i=2}^k, \mathsf{coins}; \{[\![v_i]\!]\}_{i=2}^k, \pi(\operatorname*{argmax}_i a_i))\end{aligned}$$

where coins is the random tape for encryptions. To simulate A's real view, the simulator S_A does the following on input $([\![a_1]\!], \ldots, [\![a_k]\!], l, \mathsf{PK}_{QR}, \mathsf{PK}_P, \operatorname{argmax}_i a_i)$:

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

$$\begin{aligned}&(\{[\![a_i]\!]\}_{i=1}^k, l, \mathsf{PK}_{QR}, \mathsf{PK}_P;\\ &\tilde{\pi}, \{\tilde{r}_i\}_{i=2}^k, \{\tilde{s}_i\}_{i=2}^k, \widetilde{\mathsf{coins}}; \{[\![\tilde{v}_i]\!]\}_{i=2}^k, \tilde{\pi}(\operatorname*{argmax}_i a_i))\end{aligned}$$

We define the following hybrids:

- $H_0 = V_A([\![a_1]\!], \ldots, [\![a_k]\!], l, \mathsf{SK}_{QR}, \mathsf{PK}_{QR}, \mathsf{SK}_P, \mathsf{PK}_P)$
- $H_1 = (\{[\![a_i]\!]\}_{i=1}^k, l, \mathsf{PK}_{QR}, \mathsf{PK}_P;$
  $\pi, \{r_i\}_{i=2}^k, \{s_i\}_{i=2}^k, \mathsf{coins}; \{[\![\tilde{v}_i]\!]\}_{i=2}^k, \pi(\operatorname{argmax}_i a_i))$
- $H_2 = (\{[\![a_i]\!]\}_{i=1}^k, l, \mathsf{PK}_{QR}, \mathsf{PK}_P;$
  $\pi, \{\tilde{r}_i\}_{i=2}^k, \{\tilde{s}_i\}_{i=2}^k, \widetilde{\mathsf{coins}}; \{[\![\tilde{v}_i]\!]\}_{i=2}^k, \pi(\operatorname{argmax}_i a_i))$
- $H_3 = S_A([\![a_1]\!], \ldots, [\![a_k]\!], l, \mathsf{PK}_{QR}, \mathsf{PK}_P, \operatorname{argmax}_i a_i)$

By semantic security of Paillier's cryptosystem, as B refreshes the cyphertexts, for all i, $(\mathsf{PK}_P, [\![v_i]\!]) \equiv_c (\mathsf{PK}_P, [\![\tilde{v}_i]\!])$ and more generally,

$$\begin{aligned}&(\{[\![a_i]\!]\}_{i=1}^k, l, \mathsf{PK}_{QR}, \mathsf{PK}_P; \pi, \{r_i\}_{i=2}^k, \{s_i\}_{i=2}^k;\\ &\qquad \{[\![v_i]\!]\}_{i=2}^k, \pi(\operatorname*{argmax}_i a_i))\\ \equiv_c\ &(\{[\![a_i]\!]\}_{i=1}^k, l, \mathsf{PK}_{QR}, \mathsf{PK}_P; \pi, \{r_i\}_{i=2}^k, \{s_i\}_{i=2}^k;\\ &\qquad \{[\![\tilde{v}_i]\!]\}_{i=2}^k, \pi(\operatorname*{argmax}_i a_i))\end{aligned}$$

and $H_0 \equiv_c H_1$ as $\pi(\operatorname{argmax}_i a_i) = i_0$

Given that the $\tilde{r}_i$, $\tilde{s}_i$ and $\widetilde{\mathsf{coins}}$ 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 π, the hybrids H_1 and H_2 are equal.

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

Hence, we showed that

$$\begin{aligned}&V_A(\{[\![a_i]\!]\}_{i=1}^k, l, \mathsf{SK}_{QR}, \mathsf{PK}_{QR}, \mathsf{SK}_P, \mathsf{PK}_P)\\ &\quad \equiv_c S_A(\{[\![a_i]\!]\}_{i=1}^k, l, \mathsf{PK}_{QR}, \mathsf{PK}_P, \operatorname*{argmax}_i a_i).\end{aligned}$$

B's view is

$$V_B = (\mathsf{SK}_P, \mathsf{SK}_{QR}, l; \mathsf{coins}; \{b_i\}_{i=2}^k, \{[\![m'_i]\!]\}_{i=2}^k, \{[\![a'_i]\!]\}_{i=2}^k)$$

where coins are the random coins for $k-1$ Paillier cyphertext refresh. The simulator $S_B(\mathsf{SK}_P, \mathsf{SK}_{QR}, l)$ 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 $(\mathsf{SK}_P, \mathsf{SK}_{QR}, l; \widetilde{\mathsf{coins}}; \{b_i\}_{i=2}^k, \{[\![\tilde{m}'_i]\!]\}_{i=2}^k, \{[\![\tilde{a}'_i]\!]\}_{i=2}^k)$ be B's view of this run.
4. Outputs
$$(\mathsf{SK}_P, \mathsf{SK}_{QR}, l; \widetilde{\mathsf{coins}}; \{b_i\}_{i=2}^k, \{[\![\tilde{m}'_i]\!]\}_{i=2}^k, \{[\![\tilde{a}'_i]\!]\}_{i=2}^k)$$

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 ascendent order). Let us fix the permutation π for a while and define the following hybrids:

0. $H_0 = V_B(\{[\![a_i]\!]\}_{i=1}^k, l, \mathsf{SK}_{QR}, \mathsf{PK}_{QR}, \mathsf{SK}_P, \mathsf{PK}_P)$
1. $H_1 = (\mathsf{SK}_P, \mathsf{SK}_{QR}, l; \mathsf{coins}; \{\tilde{b}_i\}_{i=2}^k, \{[\![m'_i]\!]\}_{i=2}^k, \{[\![a'_i]\!]\}_{i=2}^k)$ (the $\tilde{b}_i$ come from the simulator).
2. $H_2 = V_B(\{[\![p(a_1)]\!]\}_{i=1}^k, l, \mathsf{SK}_{QR}, \mathsf{PK}_{QR}, \mathsf{SK}_P, \mathsf{PK}_P)$

We will show that these hybrids are perfectly equal for every permutation π.

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 π, the bits b_i do not change if we replace the a_i by $p(a_i)$. Thus, $H_0 = H_1$.

As, in the real execution of the protocol and in the simulator execution, a'_i, m'_i, $\tilde{m}'_i$ and $\tilde{a}'_i$ are statistically blinded the same way, by adding random noise from $(0, 2^{\lambda+l} \cap \mathbb{Z})$, and the random tape are generated in the same way, we have $H_1 = H_2$.

Now, we want to show that $H_2 \equiv_s S_B(\mathsf{SK}_P, \mathsf{SK}_{QR}, l)$ - we do not fix π anymore. Let π_0 be the permutation such that $p(a_i) = \pi_0(i)$. We can then rewrite H_2 as

$$H_2 = V_B([\![\pi_0(1)]\!], \ldots, [\![\pi_0(k)]\!], l, \mathsf{SK}_{QR}, \mathsf{PK}_{QR}, \mathsf{SK}_P, \mathsf{PK}_P)$$

As $\tilde{\pi}$ and $\pi \circ \pi_0$ are statistically indistinguishable, we have $H_2 \equiv_s S_B(\mathsf{SK}_P, \mathsf{SK}_{QR}, l)$: 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

$$\begin{aligned} & V_B([\![a_1]\!], \ldots, [\![a_k]\!], l, \mathsf{SK}_{QR}, \mathsf{PK}_{QR}, \mathsf{SK}_P, \mathsf{PK}_P) \\ & \quad \equiv_s S_B(\mathsf{SK}_P, \mathsf{SK}_{QR}, l) \end{aligned}$$

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 2 and invoke Theorem A.2 to prove security in the semi-honest model. □

B.3 Changing the encryption scheme

Proof of Proposition 5.4. In this protocol the computed function is probabilistic, and we have to show security according to the full definition (cf. section A.1). The function is f:

$$f([\![c]\!]_1, \mathsf{PK}_1, \mathsf{PK}_2, \mathsf{SK}_1, \mathsf{SK}_2) = ([\![c]\!]_2, \emptyset)$$

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 $V_A = (\mathsf{PK}_1, \mathsf{PK}_2, [\![c]\!]_1; r; [\![c']\!]_2)$. A's output is $[\![c]\!]_2$. The simulator $S_A(\mathsf{PK}_1, \mathsf{PK}_2, [\![c]\!]_1)$ runs as follows:

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

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

$$\begin{aligned}&\{(\mathsf{PK}_1, \mathsf{PK}_2, [\![c]\!]_1; \tilde{r}; [\![\tilde{c}']\!]_2); f([\![c]\!]_1, \mathsf{PK}_1, \mathsf{PK}_2, \mathsf{SK}_1, \mathsf{SK}_2)\}\\ &\quad = \{(\mathsf{PK}_1, \mathsf{PK}_2, [\![c]\!]_1; r; [\![\tilde{c}']\!]_2); f([\![c]\!]_1, \mathsf{PK}_1, \mathsf{PK}_2, \mathsf{SK}_1, \mathsf{SK}_2)\}\end{aligned}$$

By semantic security of scheme E_2 we have that

$$\begin{aligned}&\{(\mathsf{PK}_1, \mathsf{PK}_2, [\![c]\!]_1; r; [\![\tilde{c}']\!]_2); f([\![c]\!]_1, \mathsf{PK}_1, \mathsf{PK}_2, \mathsf{SK}_1, \mathsf{SK}_2)\}\\ &\quad \equiv_c \{(\mathsf{PK}_1, \mathsf{PK}_2, [\![c]\!]_1; r; [\![c']\!]_2); [\![c]\!]_2\}\end{aligned}$$

and so

$$\begin{aligned}&\{S_A([\![c]\!]_1, \mathsf{PK}_1, \mathsf{PK}_2), f([\![c]\!]_1, \mathsf{PK}_1, \mathsf{PK}_2, \mathsf{SK}_1, \mathsf{SK}_2)\}\\ &\quad \equiv_c \{V_A([\![c]\!]_1, \mathsf{PK}_1, \mathsf{PK}_2, \mathsf{SK}_1, \mathsf{SK}_2),\\ &\qquad\qquad \mathsf{Output}([\![c]\!]_1, \mathsf{PK}_1, \mathsf{PK}_2, \mathsf{SK}_1, \mathsf{SK}_2)\}\end{aligned}$$

B's view is $V_B = (\mathsf{SK}_1, \mathsf{SK}_2; [\![c+r]\!]_1)$. We build a simulator $S_B(\mathsf{SK}_1, \mathsf{SK}_2)$:

1. Picks a random $\tilde{c} \leftarrow M$.
2. Encrypt $\tilde{c}$ under E_1.
3. Outputs $(\mathsf{SK}_1, \mathsf{SK}_2, [\![\tilde{c}]\!]_1)$.

Again, the distribution of $\tilde{c}$ and $c + r$ are identical, so the real distribution $\{(\mathsf{SK}_1, \mathsf{SK}_2; [\![c+r]\!]_1); [\![c]\!]_2\}$ and the ideal distribution $\{(\mathsf{SK}_1, \mathsf{SK}_2; [\![\tilde{r}]\!]_1); f([\![c]\!]_1, \mathsf{PK}_1, \mathsf{PK}_2, \mathsf{SK}_1, \mathsf{SK}_2)\}$ are statistically indistinguishable. □

B.4 Computing dot products

Proof of Proposition 5.5. 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

$$S_B(y, \mathsf{SK}_P) = (y, \mathsf{SK}_P; \mathsf{coins}) = V_B(x, y, \mathsf{SK}_P, \mathsf{PK}_P).$$

where rand are the random coins.

A's view is $V_A = (x, \mathsf{PK}_P; r^A; [\![y_1]\!], \ldots, [\![y_n]\!])$. On input $(x, \mathsf{PK}_P, [\![v]\!])$, 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 $\widetilde{\mathsf{coins}}$.
3. Outputs $(x, \mathsf{PK}_P; \widetilde{\mathsf{coins}}; c_1, \ldots, c_n)$.

coins and $\widetilde{\mathsf{coins}}$ come from the same distribution, independently from other parameters. Thus,

$$\begin{aligned}&\{(x, \mathsf{PK}_P; \widetilde{\mathsf{coins}}; c_1, \ldots, c_n); [\![\langle x, y\rangle]\!]\}\\ &\quad = \{(x, \mathsf{PK}_P; \mathsf{coins}; c_1, \ldots, c_n); [\![\langle x, y\rangle]\!]\}\end{aligned}$$

and by semantic security of Paillier,

$$\begin{aligned}&\{(x, \mathsf{PK}_P; \mathsf{coins}; c_1, \ldots, c_n); [\![\langle x, y\rangle]\!]\}\\ &\quad \equiv_c \{(x, \mathsf{PK}_P; \mathsf{coins}; [\![y_1]\!], \ldots, [\![y_n]\!]); [\![v]\!]\}\end{aligned}$$

i.e., when f is $f(x, y, \mathsf{SK}_P, \mathsf{PK}_P) = (\llbracket\langle x, y\rangle\rrbracket, \emptyset)$

$$\{S_A(x, \mathsf{PK}_P, \llbracket v\rrbracket); f(x, y, \mathsf{SK}_P, \mathsf{PK}_P)\}$$
$$\equiv_c \{V_A(x, y, \mathsf{SK}_P, \mathsf{PK}_P); \mathsf{Output}(x, y, \mathsf{SK}_P, \mathsf{PK}_P)\}$$

□

B.5 Classifiers

B.5.1 Hyperplane decision

Proof of Proposition 6.1. The client's view is $V_C = (\mathsf{PK}_P, \mathsf{PK}_{QR}, x; \{\llbracket v_i\rrbracket\}_{i=1}^k, i_0)$. The simulator S_C, on input $(\mathsf{PK}_P, \mathsf{SK}_{QR}, x, k^*)$ where $k^* = \underset{i\in[k]}{\mathrm{argmax}}\langle w_i, x\rangle$ does the following:

1. Generate k random Paillier encryptions $\llbracket\tilde{v}_i\rrbracket$
2. Output $(\mathsf{PK}_P, \mathsf{SK}_{QR}, x; \{\llbracket\tilde{v}\rrbracket\}_{i=1}^k, k^*)$

As the index i_0 that the client receives is its output, and as Paillier's cryptosystem is semantically secure, the distributions $S_C = (\mathsf{PK}_P, \mathsf{SK}_{QR}, x; \{\llbracket\tilde{v}\rrbracket\}_{i=1}^k, k^*)$ and $V_C = (\mathsf{PK}_P, \mathsf{SK}_{QR}, x; \{\llbracket v_i\rrbracket\}_{i=1}^k, i_0)$ 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 3 and 5 are secure in the honest-but-curious model, we obtain the security of the hyperplane decision protocol using modular sequential composition (Theorem A.2). □

B.5.2 Bayes classifier

Proof of Proposition 7.1. The client's view is $V_C = (\mathsf{PK}_P, \mathsf{SK}_{QR}, x; \llbracket T^C\rrbracket, \llbracket T^1\rrbracket, \ldots, \llbracket T^n\rrbracket, i_0)$. The simulator S_C, on input $(\mathsf{PK}_P, \mathsf{SK}_{QR}, x, i_{max})$ where $i_{max} = \mathrm{argmax}_j\, \mathbb{P}(C = c_j | X = x)$,

- generates tables of random Paillier encryptions $\llbracket\tilde{T}^C\rrbracket$ and $\llbracket\tilde{T}^i\rrbracket$;
- outputs $(\mathsf{PK}_P, \mathsf{SK}_{QR}, x; \llbracket\tilde{T}^C\rrbracket, \llbracket\tilde{T}^1\rrbracket, \ldots, \llbracket\tilde{T}^n\rrbracket, i_{max})$.

As the integer i_0 that the client receives is its output, and as Paillier's cryptosystem is semantically secure, the distributions $S_C = (\mathsf{PK}_P, \mathsf{SK}_{QR}, x; \llbracket\tilde{T}^C\rrbracket, \llbracket\tilde{T}^1\rrbracket, \ldots, \llbracket\tilde{T}^n\rrbracket, i_{max})$ and $V_C = (\mathsf{PK}_P, \mathsf{SK}_{QR}, x; \llbracket T^C\rrbracket, \llbracket T^1\rrbracket, \ldots, \llbracket T^n\rrbracket, i_0)$ 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 3 is secure in the honest-but-curious model, we obtain the security of the hyperplane decision protocol using modular sequential composition (Theorem A.2). □

B.5.3 Decision tree

Proof of Proposition 8.1. The proof of security for the server is very easily obtained using modular sequential composition of Protocol 4.

For the client also the proof is trivial, using modular sequential composition and the semantical security of the FHE scheme. □

C Computing 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):

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

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.

D Preparing data for the Naïve Bayes classifier

First of all, to be able to use our tools, we have to work with integers and not floats. Hopefully, as the only operations used in the classification step are additions (*cf.* Equation (1)), we can just multiply the conditional probabilities $p(x_j|c_i)$ by a constant K and truncate the results to the lower integer.

For example, if we are able to compute the conditional probabilities using IEEE 754 double precision floating point numbers, with 52 bits of precision and if e is the biggest exponent of these floating point numbers, we can choose K to be $K = 2^{e+52}$. We will keep the same precision (52 bits) and use integers.

Let l_0 be the maximum bit length of an element in the integer table.

Now, if – as before – d is the number of features, the maximum number of bits when doing the computations will be $l_{max} = l_0 + d + 1$: we have to add the probabilities for the d features and the probability of the class label. Hence, the value l used for the comparison protocols must be chosen larger than l_{max}. Actually, as probabilities are number smaller than one, their logarithm is negative. As a consequence, according to Section 5.1.5, we must take $l \geq l_{max} + 1$. In the case of IEEE 754 doubles, we have $l \geq 53$.

Finally, we must also ensure that $\log_2 N > l + 1 + \lambda$ where λ is the security parameter and N is the modulus for Paillier's cryptosystem plaintext space (*cf.* Section 5.1.2). 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 $\lambda \approx 100$.

Thus, the server produces $kd + 1$ tables T using the well chosen positive constant K:

- the table for the priors on the class T^C: $T^C(i) = \lceil K \log p(c_i) \rceil$
- one table per feature per class T^j: $T^j(x_j, i) = \lceil K \log p(x_j|c_i) \rceil$

We finish the data preparation by encrypting each entry of the table using the secret key for Paillier's cryptosystem.