MACHINE LEARNING CLASSIFICATION OVER ENCRYPTED DATA

CRYPTOGRAPHY COURSE
9TH SEMESTER, 2021-2022

Project Presentation

Christos Tsoufis

ECE, NTUA

Contents

1. Introduction

- Classifiers are an invaluable tool for many tasks today, such as medical or genomics predictions, spam detection, face recognition, and finance. Many of these applications handle sensitive data, so it is important that the data and the classifier remain private.
- Supervised learning algorithms consist of two phases: (i) the training phase during which the algorithm learns a model w from a data set of labeled examples, and (ii) the classification phase that runs a classifier C over a previously unseen feature vector x, using the model w to output a prediction C(x, w).
- In applications that handle sensitive data, it is important that the feature vector x and the model w remain secret to one or some of the parties involved (e.g. medical study). Throughout this paper, we refer to this goal shortly as privacy-preserving classification. Concretely, a client has a private input represented as a feature vector x, and the server has a private input consisting of a private model w.
- 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. These classifiers are widely used – even though there are many machine learning algorithms, most of them end up using one of these three classifiers.

1. Introduction

- Protocols specialized to the classification problem promise better performance. 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, making it hard to support efficiently. The second is providing a solution that is more generic than the 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 improving 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 use a set of sub-techniques:
    - The input and output of all our building blocks are data encrypted with additively homomorphic encryption. In addition, we provide a mechanism to switch from one encryption scheme to another. Intuitively, this enables a building block's output to become the input of another building block
    - The API of these building blocks is flexible: even though each building block computes a fixed function, it allows a choice of which party provides the inputs to the protocol, which party obtains the output of the computation, and whether the output is encrypted or decrypted
    - The security of these protocols composes using modular sequential composition

2. Related work

- Our work is the first to provide efficient privacy-preserving protocols for a broad class of classifiers
- Previous work focusing on privacy-preserving machine learning can be broadly divided into two categories: (i) techniques for privacy-preserving training, and (ii) techniques for privacy-preserving classification.
- Most existing work falls in the first category. Our work falls in the second category, where little work has been done.
- It is worth mentioning that our work on privacy-preserving classification is complementary to work on differential privacy in the machine learning community. Our work aims to hide each user's input data to the classification phase, whereas differential privacy seeks to construct classifiers/models from sensitive user training data that leak a bounded amount of information about each individual in the training data set.

2.1 Privacy-preserving training

- A set of techniques have been developed for privacy-preserving training algorithms such as Naïve Bayes, decision trees, linear discriminant classifiers, and more general kernel methods.
- Grapel show how to train several machine learning classifiers using a somewhat homomorphic encryption scheme. They focus on a few simple classifiers, and do not elaborate on more complex algorithms such as SVMs. 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 FHE alone is inefficient, a difficulty we overcome with an interactive setting.

2.2 Privacy-preserving classification
2.3 Work related to our building blocks

- 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, a third party can compute medical prediction functions over the encrypted data of a patient using fully homomorphic encryption. In their setting, everyone knows the predictive model, and their algorithm hides only the input of the patient from the cloud. Our protocols, on the other hand, also hide the model from the patient. Their algorithms cannot be applied to our setting.
- Barni 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, and our evaluation shows that our construction is twice as fast on branching programs.
- Other works construct specific face recognition or detection classifiers. We focus on providing a set of generic classifiers and building blocks to construct more complex classifiers.
- Finally, two of the basic components we use are private comparison and private computation of dot products. These items have been well-studied previously for comparison techniques and for the computation of dot products.

3. Background and preliminaries
3.1 Classification in machine learning algorithms

- The user's input x is a vector of d elements $x = (x_1, \dots, x_d) \in \mathbb{R}^d$, called a feature vector.
- To classify the input x means to evaluate a classification function $C_w: \mathbb{R}^d \mapsto \{c_1, \dots, c_k\}$ on x.
- The output is $c_{k^*} = C_w(x)$, where $k^* \in \{1, \dots k\}$ and c_{k^*} is the class to which x corresponds, based on the model w.
- We now describe how three popular classifiers work on regular, unencrypted data.

3. Background and preliminaries
3.1 Classification in machine learning algorithms

Hyperplane decision-based classifiers

For this classifier, the model w consists of k vectors in $\mathbb{R}^d$ $\left(w = \{w_i\}_{i=1}^k\right)$. The classifier is:

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

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

A hyperplane based classifier typically works with a hypothesis space $\mathcal{H}$ equipped with an inner product. This classifier usually solves a binary classification problem (k = 2): given a user input x, x is classified in class c_2 if $\langle w, \varphi(x) \rangle \geq 0$, otherwise it is labeled as part of class c_1. In this work, we focus on the case when $\mathcal{H} = \mathbb{R}^d$ and note that a large class of infinite dimensional spaces can be approximated with a finite dimensional space, including the popular gaussian kernel (RBF). In this case $\varphi(x) = x$ or $\varphi(x) = Px$ for a randomized projection matrix P chosen during training.

3. Background and preliminaries
3.1 Classification in machine learning algorithms

Naive Bayes classifiers

For this classifier, the model w consists of various probabilities: the probability that each class c_i occurs, namely $\{p(C = c_i)\}_{i=1}^{k}$, and the probabilities that an element x_j of x occurs in a certain class c_i. More concretely, the latter is the probability of the j-th component x_j of x to be v when x belongs to category c_i; this is denoted by $\left\{\left\{\left\{p\left(X_j = v | C = c_i\right)\right\}_{v \in D_j}\right\}_{j=1}^{d}\right\}_{i=1}^{k}$, where D_j is X_j's domain. The classification function, using a maximum a posteriori decision rule, works by choosing the class with the highest posterior probability:

$$k^* = \underset{i \in [k]}{\mathrm{arg\,max}}\, p(C = c_i | X = x) = \underset{i \in [k]}{\mathrm{arg\,max}}\, p(C = c_i, X = x) = \underset{i \in [k]}{\mathrm{arg\,max}}\, p(C = c_i, X_1 = x_1, \dots, X_d = x_d)$$

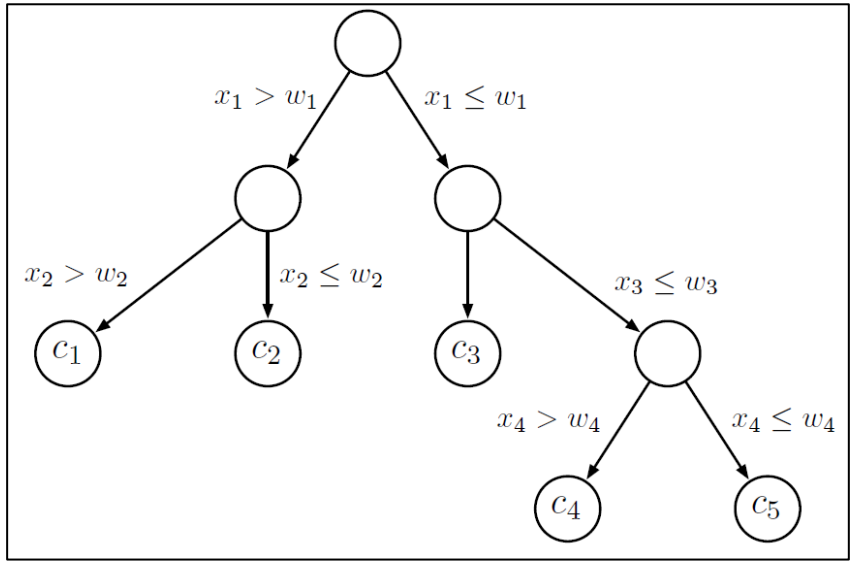
The Naïve Bayes model assumes that $p(C = c_i, X = x)$ has the following factorization:

$$p(C = c_i, X_1 = x_1, \dots, X_d = x_d) = p(C = c_i) \prod_{j=1}^{d} p\left(X_j = x_j | C = c_i\right)$$

3. Background and preliminaries
3.1 Classification in machine learning algorithms

Decision trees

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



3.2 Cryptographic preliminaries
3.2.1 Cryptosystems

In this work, we use three additively homomorphic cryptosystems. A public-key encryption scheme HE is additively homomorphic if, given two encrypted messages HE.Enc(a) and HE.Enc(b), there exists a public-key operation $\oplus$ such that HE.Enc(a) $\oplus$ HE.Enc(b) is an encryption of a + b. The cryptosystems we use are:

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

3.2.2 Cryptographic assumptions
3.2.3 Adversarial model

- We prove that our protocols are secure based on the semantic security of the above cryptosystems. These cryptosystems rely on standard and well-studied computational assumptions: the Quadratic Residuosity assumption, the Decisional Composite Residuosity assumption, and the Ring Learning With Error (RLWE) assumption.
- We prove security of our protocols using the secure two-party computation framework for passive adversaries (or honest-but-curious).

3.3 Notation

- All our protocols are between two parties: parties A and B for our building blocks and parties C (client) and S (server) for our classifiers.
- Inputs and outputs of our building blocks are either unencrypted or encrypted with an additively homomorphic encryption scheme.
- The plaintext space of QR is $\mathbb{F}_2$ (bits), and we denote by [b] a bit b encrypted under QR; the plaintext space of Paillier is $\mathbb{Z}_N$ where N is the public modulus of Paillier, and we denote by $[\![m]\!]$ an integer m encrypted under Paillier. The plaintext space of the FHE scheme is $\mathbb{F}_2$. We denote by $\mathrm{SK_P}$, $\mathrm{PK_P}$, a secret and a public key for Paillier, respectively. Also, we denote by $\mathrm{SK_{QR}}$, $\mathrm{PK_{QR}}$, a secret and a public key for QR. For a constant b, a ← b means that a is assigned the value of b. For a distribution D, a ← D means that a gets a sample from D.

4. Building blocks
4.1 Comparison

- In this section, we develop a library of building blocks, which we later use to build our classifiers.
- We now describe our comparison protocol. In order for this protocol to be used in a wide range of classifiers, its setup needs to be flexible. Our comparison protocol can be used in various ways. In each case, each party learns nothing else about the other party's input other than what is indicated as the output.

Type	Input A	Input B	Output A	Output B	Implementation
1	PK_P, PK_{QR}, a	SK_P,SK_{QR}, b	$[a < b]$	–	Sec. 4.1.1
2	PK_P, SK_{QR}, $[\![a]\!]$, $[\![b]\!]$	SK_P,PK_{QR}	–	$[a \leq b]$	Sec. 4.1.2
3	PK_P, SK_{QR}, $[\![a]\!]$, $[\![b]\!]$	SK_P,PK_{QR}	$a \leq b$	$[a \leq b]$	Sec. 4.1.2
4	PK_P, PK_{QR}, $[\![a]\!]$, $[\![b]\!]$	SK_P,SK_{QR}	$[a \leq b]$	–	Sec. 4.1.3
5	PK_P, PK_{QR},$[\![a]\!]$, $[\![b]\!]$	SK_P,SK_{QR}	$[a \leq b]$	$a \leq b$	Sec. 4.1.3

- There are at least two approaches to performing comparison efficiently: using specialized homomorphic encryption (efficient for comparison of encrypted values), or using garbled circuits (efficient for comparison of unencrypted values).

4.1.1 Comparison with unencrypted inputs
4.1.2 Comparison with encrypted inputs

- To compare unencrypted inputs, we use:

1. garbled circuits implemented with the state-of-the-art garbling scheme of Bellare
2. the short circuit for comparison of Kolesnikov
3. a well-known oblivious transfer (OT) scheme due to Naor and Pinkas

Finally, the conversion from a garbled output to homomorphic encryption to enable composition can be implemented easily using the random shares technique. The above techniques combined give us the desired comparison protocol. Actually, we can directly combine them to build an even more efficient protocol: we use an enhanced comparison circuit that also takes as input a masking bit.

- Our classifiers also require the ability to compare two encrypted inputs. More specifically, suppose that party A wants to compare two encrypted integers a and b, but party B holds the decryption key. To implement this task, we slightly modify Veugen's protocol

4.1.3 Reversed comparison over encrypted data
4.1.4 Negative integers comparison and sign determination

- In some cases, we want the result of the comparison to be held by the party that does not hold the encrypted data. For this, we modify Veugen's protocol to reverse the outputs of party A and party B: we achieve this by exchanging the role of party A and party B in the last few steps of the protocol, after invoking the comparison protocol with unencrypted inputs.
- Negative numbers are handled by the protocols above unchanged. Even though the Paillier plaintext size is "positive", a negative number simply becomes a large number in the plaintext space due to cyclicity of the space.

4.2 argmax over encrypted data

- In this scenario, party A has k values $a_1, \ldots, a_k$ encrypted under party B's secret key and wants party B to know the argmax over these values (the index of the largest value), but neither party should learn anything else. For example, if A has values $[\![1]\!]$, $[\![100]\!]$ and $[\![2]\!]$, B should learn that the second is the largest value, but learn nothing else. In particular, B should not learn the order relations between the a_i's. Our protocol for argmax is shown in Protocol 1.
- In the end, our protocol performs k–1 encrypted comparisons of l bits integers and 7(k–1) homomorphic operations (refreshes, multiplications and subtractions). In terms of round trips, we add k–1 roundtrips to the comparison protocol, one roundtrip per loop iteration.
- Proposition: Protocol 1 is correct and secure in the honest-but-curious model.

4.2 argmax over encrypted data

Protocol 1 argmax over encrypted data

Input A: k encrypted integers $([\![a_1]\!], \ldots, [\![a_k]\!])$, the bit length l of the a_i, and public keys PK_{QR} and PK_P
Input B: Secret keys SK_P and SK_{QR}, the bit length l
Output A: $\text{argmax}_i\, a_i$

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

4.3 Changing the encryption scheme

- To enable us to compose various building blocks, we developed a protocol for converting ciphertexts from one encryption scheme to another while maintaining the underlying plaintexts. We first present a protocol that switches between two encryption schemes with the same plaintext size (such as QR and FHE over bits), and then present a different protocol for switching from QR to Paillier.
- Concretely, consider two additively homomorphic encryption schemes E_1 and E_2, both semantically secure with the same plaintext space M. Let $[\![\cdot]\!]_1$be an encryption using E_1 and $[\![\cdot]\!]_2$ an encryption using E_2. Consider that party B has the secret keys SK_1 and SK_2 for both schemes and A has the corresponding public keys PK_1 and PK_2. Party A also has a value encrypted with PK_1, $[\![c]\!]_1$. Our protocol, protocol 2, enables A to obtain an encryption of c under E_2, $[\![c]\!]_2$ without revealing anything to B about c.
- Proposition: Protocol 2 is secure in the honest-but-curious model.
- See more in paper, section “XOR with Paillier”.

4.3 Changing the encryption scheme

Protocol 2 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 $[\![c]\!]_2$

4.4 Computing dot products
4.5 Dealing with floating point numbers

- For completeness, we include a straightforward algorithm for computing dot products of two vectors, which relies on Paillier's homomorphic property.
- Proposition: Protocol 3 is secure in the honest-but-curious model.

Protocol 3 Private dot product

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

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

- 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. a simple solution is to multiply each floating point value by a constant K.

5. Private hyperplane decision

- This classifier computes: $k^* = \underset{i \in [k]}{argmax} \langle w_i, x \rangle$
- Now that we constructed our library of building blocks, it is straightforward to implement this classifier securely.
- Proposition: Protocol 4 is secure in the honest-but-curious model.

Protocol 4 Private hyperplane decision

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

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

6. Secure Naïve Bayes classifier

- The goal is for the client to learn k^* without learning anything about the probabilities that constitute the model, and the server should learn nothing about x. Recall that the features values domain is discrete and finite.
- As is typically done for numerical stability reasons, we work with the logarithm of the probability distributions:

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

6.1 Preparing the model

- Since the Paillier encryption scheme works with integers, we convert each log of a probability from above to an integer by multiplying it with a large number K.
- As the only operations used in the classification step are additions and comparisons, we can just multiply the conditional probabilities $p(x_j|c_i)$ by a constant K so to get integers everywhere, while keeping the same classification result.
- The K will be $K \cdot v_i = m_i' \cdot 2^{\delta_i} \in \mathbb{N}$. The analytical process is described in the paper.
- However, remember that we are doing all this to store logarithms of probabilities in Paillier cyphertexts, and as Paillier plaintext space is very large and δ_i's remain small. Also notice that this shifting procedure can be done without any loss of precision as we can directly work with the bit representation of the floating points numbers. Finally, we make sure that we do not overflow Paillier's message space when doing all the operations.
- It is shown that this preparation step can be done once and for all at server startup, and is hence amortized.

6.2 Protocol

Protocol 5 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 $\{\log p(C = c_i)\}_{1 \le i \le k}$ and $\left\{\{\log p(X_j = v | C = c_i)\}_{v \in D_j}\right\}_{1 \le j \le d, 1 \le i \le k}$
Client's Output: i_0 such that $p(x, c_{i_0})$ is maximum

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

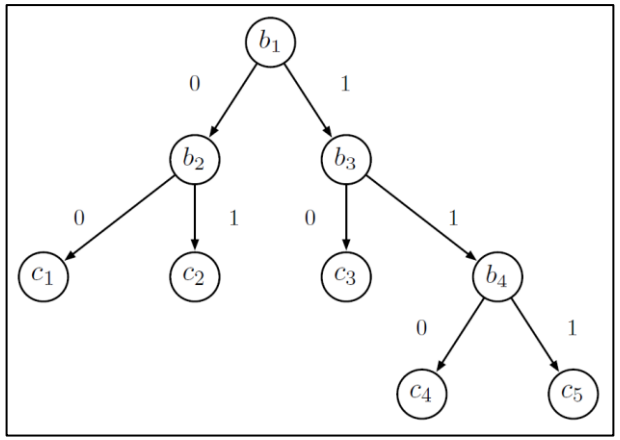
- Proposition: Protocol 5 is secure in the honest-but-curious model.
- Efficiency: Regarding the number of round trips, these are due to the argmax protocol: k–1 runs of the comparison protocol and k–1 additional roundtrips.

7. Private decision trees

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

7.1 Polynomial form of a decision tree

- Consider that each node of the tree has a boolean variable associated to it. The value of the boolean at a node is 1 if, on input x, one should follow the right branch, and 0 otherwise.
- We construct a polynomial P that, on input all these boolean variables and the value of each class at a leaf node, outputs the class predicted for x. The idea is that P is a sum of terms, where each term (say t) corresponds to a path in the tree from root to a leaf node (say c). A term t evaluates to c iff x is classified along that path in T, else it evaluates to zero.
- For example: $P(b_1, \dots, b_4, c_1, \dots, c_5) = b_1(b_3(b_4c_5 + (1 - b_4)c_4) + (1 - b_3)c_3) + +(1 - b_1)(b_2c_2 + (1 - b_2)c_1)$



7.2 Private evaluation of a polynomial

- Let us first explain how to compute the values of the boolean variables securely. Let n be the number of nodes in the tree and nleaves be the number of leaves in the tree. These values must remain unknown to the server because they leak information about x: they are the result of the intermediate computations of the classification criterion. For each boolean variable b_i, the server and the client engage in the comparison protocol to compare w_i and the corresponding attribute of x. As a result, the server obtains $[b_i]$ for $i \in 1, \dots, n$; the server then changes the encryption of these values to FHE using Protocol 2, thus obtaining $[\![[b_i]]\!]$.
- The server evaluates P on $([\![[b_1]]\!], \dots, [\![[b_n]]\!])$ using the homomorphic properties of FHE.
- First, we use a leveled FHE scheme. Second, we ensure that the multiplicative depth is very small using a tree-based evaluation. Finally, we use $\mathbb{F}_2$ as the plaintext space and SIMD slots for parallelism.
- The evaluation of the decision tree is done using 2n FHE multiplications and 2n FHE additions where n is the number of criteria. The evaluation circuit has multiplication depth $\lceil \log_2 n + 1 \rceil$.

7.3 Formal description

- Protocol 6 describes the resulting protocol.
- Proposition: Protocol 6 is secure in the honest-but-curious model.

Protocol 6 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, including the n thresholds $\{w_i\}_{i=1}^n$.
Client's Output: The value of the leaf of the decision tree associated with the inputs $b_1, \ldots, b_n$.

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

8. Combining classifiers with AdaBoost

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

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

If each of the $h_i(\cdot)$'s is an instance of a classifier supported by our protocols, then given the scalars a_i, we can easily and securely evaluate H(x) by simply composing our building blocks.

9. Implementation

- We have implemented the protocols and the classifiers in C++ using GMP, Boost, Google's Protocol Buffers5, and HELib for the FHE implementation.
- The code is written in a modular way. For example, for the linear classifier:

```
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);
}
```

10. Evaluation

To evaluate our work, we answer the following questions:

(i) can our building blocks be used to construct other classifiers in a modular way?

(ii) what is the performance overhead of our building blocks?

(iii) what is the performance overhead of our classifiers?

10.1 Using our building blocks library
10.1.1 Building a multiplexer classifier

- Here we demonstrate that our building blocks library can be used to build other classifiers modularly and that it is a useful contribution by itself. We will construct a multiplexer and a face detector.
- A multiplexer is the following generalized comparison function:

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

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

$$f_{a,\beta}(d) = d \cdot \beta + (1-d) \cdot a = a + d \cdot (\beta - a)$$

Then, using Paillier's homomorphism and knowledge of a and β, we can compute an encryption

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

10.1.2 Viola and Jones face detection

The Viola and Jones face detection algorithm 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:

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

where the h_t are weak classifiers of the form $h_i(x) = sign(\langle x, y_i \rangle - \theta_i)$

In our setting, Alice owns the image and Bob the classifier. 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.

10.2 Performance evaluation setup
10.3 Building blocks performance

- Our performance evaluations were run using two desktop computers each with identical configuration. The machines were on the same network.
- We examine performance in terms of computation time at the client and server, communication bandwidth, and also number of interactions (round trips).
- Some of the results can be seen on the next slide.

10.3.1 Comparison protocols

- Comparison with unencrypted input:

Bit size	A Computation	B Computation	Total Time	Communication	Interactions
10	14.11 ms	8.39 ms	105.5 ms	4.60 kB	3
20	18.29 ms	14.1 ms	117.5 ms	8.82 kB	3
32	22.9 ms	18.8 ms	122.6 ms	13.89 kB	3
64	34.7 ms	32.6 ms	134.5 ms	27.38 kB	3

- Comparison with encrypted input:

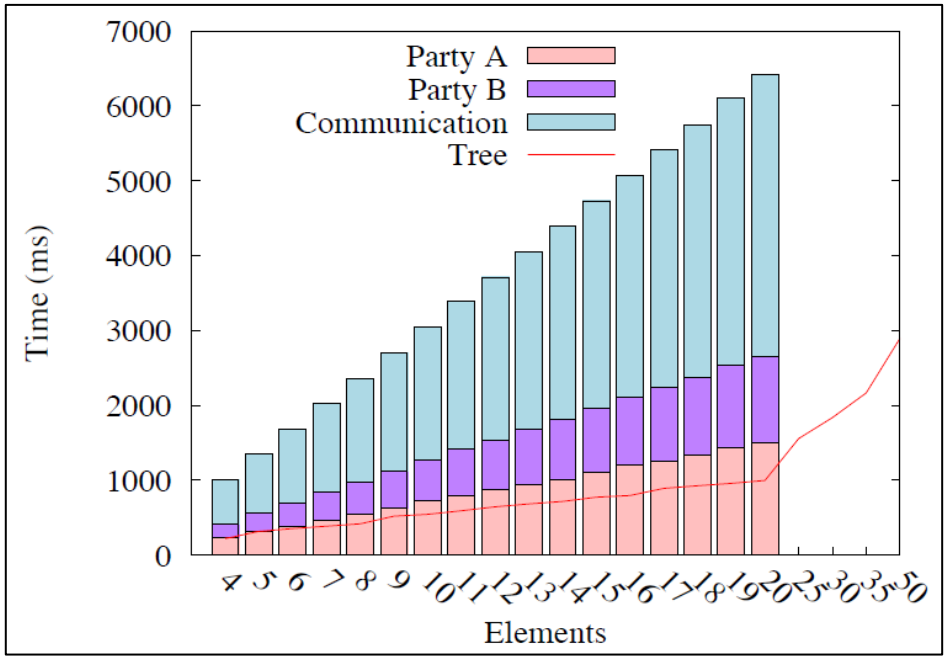
Protocol	Bit size	Computation		Total Time	Communication	Interactions
		Party A	Party B			
Comparison	64	45.34 ms	43.78 ms	190.9 ms	27.91 kB	6
Reversed Comp.	64	48.78 ms	42.49 ms	195.7 ms	27.91 kB	6

- Change encryption scheme protocol evaluation:

Party A Computation	Party B Computation	Total Time	Communication	Interactions
30.80 ms	255.3 ms	360.7 ms	420.1 kB	2

10.3.2 argmax

- The following Figure presents the running times and the communication overhead of the argmax of encrypted data protocol.
- 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.



10.3.3 Consequences of the latency on performances

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

10.4 Classifier performance

Here we evaluate each of the classifiers that were described before. The models are trained non-privately using scikit-learn. We used the following datasets from the UCI machine learning repository:

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

The next Table shows the performance results.

10.4 Classifier performance

Data set	Model size	Computation		Time per protocol		Total running time	Comm.	Interactions
		Client	Server	Compare	Dot product			
Breast cancer (2)	30	46.4 ms	43.8 ms	194 ms	9.67 ms	204 ms	35.84 kB	7
Credit (3)	47	55.5 ms	43.8 ms	194 ms	23.6 ms	217 ms	40.19 kB	7

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

Data set	Specs.		Computation		Time per protocol		Total running time	Comm.	Interactions
	C	F	Client	Server	Prob. Comp.	Argmax			
Breast Cancer (1)	2	9	150 ms	104 ms	82.9 ms	396 ms	479 ms	72.47 kB	14
Nursery (5)	5	9	537 ms	368 ms	82.8 ms	1332 ms	1415 ms	150.7 kB	42
Audiology (4)	24	70	1652 ms	1664 ms	431 ms	3379 ms	3810 ms	1911 kB	166

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

Data set	Tree Specs.		Computation		Time per protocol		FHE		Comm.	Interactions
	N	D	Client	Server	Compare	ES Change	Eval.	Decrypt		
Nursery (5)	4	4	1579 ms	798 ms	446 ms	1639 ms	239 ms	33.51 ms	2639 kB	30
ECG (6)	6	4	2297 ms	1723 ms	1410 ms	7406 ms	899 ms	35.1 ms	3555 kB	44

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

10.5 Comparison to generic two-party tools

- A set of generic secure two- or multi-party computation tools have been developed, such as TASTY and Fairplay. These support general functions, which include our classifiers.
- However, they are prohibitively slow for our specific setting. Our efficiency comes from specializing to classification functionality.
- Thus, our specialized protocols improve performance by orders of magnitude.

11. Conclusion
12. Resources

- In this paper, we constructed three major privacy-preserving classifiers as well as provided a library of building blocks that enables constructing other classifiers. We demonstrated the efficiency of our classifiers and library on real datasets.
- Raphael Bost, Raluca Ada Popa, Stephen Tu, Shafi Goldwasser: Machine Learning Classification over Encrypted Data. NDSS 2015. https://eprint.iacr.org/2014/331

End of Presentation

Thank you!