PRIVACY PRESERVING PATTERN CLASSIFICATION

Shai Avidan[1] *Ariel Elbaz*[2] *Tal Malkin*[2]

[1] Adobe Systems Inc.
[2] Department of CS, Columbia University

ABSTRACT

We give efficient and practical protocols for Privacy Preserving Pattern Classification that allow a client to have his data classified by a server, without revealing information to either party, other than the classification result. We illustrate the advantages of such a framework on several real-world scenarios and give secure protocols for several classifiers.

1. INTRODUCTION

Pattern classification is an important field within machine learning, dealing with the problem of classifying raw data into predetermined categories. Applications abound, including classifying blood samples as infected or not, classifying an e-mail as being spam or not, recognizing images of human faces, classifying an optical character into one of the 26 alphabet characters, and many more.

However, an increasing number of applications involve sensitive data, and require privacy of both the object to be classified, and the raw data used by the classifier. In particular, a client may wish to have his data classified by a commercial company without compromising the privacy of his data, while the company does not want to release its classifier. For example, the client would like to have his blood tested by the company to find if he is HIV positive. Or, consider a client (say, a bank) who would like to deploy a surveillance network of cameras and microphones in his building, and outsource the surveillance task to some outside company that will provide image and audio analysis tools to detect events such as suspicious behavior, detecting known criminals or detecting gun shots, without being able to snoop around. A different mode of operation might be when a government agency is interested in probing the records of a private company (e.g., a credit card company) to identify monetary transactions that fit, say, suspicious criminal behavior. The suspicious behavior profile is represented as a classifier that given money transactions can determine if they are suspicious or not. Obviously, the classifier itself can not be revealed to the private company. A final example involves collaboration between governments: the US government requires details of all passengers on-board inbound flights to be disclosed to the US authorities immediately after take-off, while the EU supreme court ruled this practice as violating the privacy rights of the passengers. This problem can be solved if the US government could classify passenger records and only obtain a binary answer to the question: Are there suspicious passengers on-board this flight? (where suspicious is defined by their own classifier, involving different attributes).

In this paper, we study *privacy preserving pattern classification*, which addresses exactly the type of problems above; It allows one party to classify their query using the classifier of a different party, while maintaining privacy of all data.

2. CRYPTOGRAPHIC BACKGROUND

The notion of Secure multi-party computation originated from the work of Yao [1], who defined the two-party problem of computing a known function on the parties' private inputs. Perhaps surprisingly, Yao gave a protocol that solved this problem. Yao's solution used a Boolean circuit for the function to be computed, then have one party 'garble' it, shuffling each gate's truth table and associating random strings with the 0 and 1 that usually go through a Boolean circuit's wires (only the output gates are left ungarbled). The other party then evaluates the garbled circuit, following random truth tables until she reaches the output gates to get the real output. This technique solved the two-party problem, as long as the parties can trust each other to 'follow the protocol'. Also, we note that its complexity is linear in the size of the circuit, and thus from a theorist point of view it is considered very efficient. However, for real world application this method is not practical and is much slower then the corresponding non-secure computation. Soon afterward, Goldriech *et al.* [2] extended the problem to the case of $n > 2$ parties, and gave a slightly different solution, which was was still too demanding to be of practical use. See [3] for advanced and theoretical treatment of the problem.

Since then this field has became central to Cryptography and received significant amount of research attention. Part of the research effort went to showing stronger theoretical results (security against parties allowed to cheat in various ways, and using weaker cryptographic assumption). Another direction was to make these protocols more practical and efficient. The goal in this line of research is to find an efficient protocol for some *specific* problem, bringing the complexity of the secure solution closer to the complexity of the stan-

dard, non-secure, solution. Of particular interest for us are those for secure dot-product and oblivious polynomial evaluation [4, 5], which also gave a practical secure protocol for learning decision trees [6]. The Oblivious Polynomial Evaluation (OPE) problem is when Bob's input is some polynomial $P(x)$ of degree d, while Alice's input is a value x, and she wants to learn $P(x)$, subject to Bob not learning her input and Alice not learning Bob's polynomial. This was later used by [6] to devise an ID3 decision tree learning algorithm where each party holds part of the training data, yet both parties are interested in learning a decision tree that uses all the available training data. In the end both parties learn the parameters of the decision tree, but nothing about the training data of the other party.

When analyzing secure multi-party computation protocols, the security and correctness are usually analyzed *together*. The technique used is to compare the transcripts of the protocol with the transcripts from interacting with a trusted third party, who is trusted by all the parties involved in the computation. The trusted third party is only getting the inputs from all involved parties, computes the function, and gives the respective outputs to the parties. It is easy to see that if the transcripts are similar, than the protocol is both secure and correct; the parties' output is part of the transcript, so correctness is trivial. Also, none of the involved computing parties can learn anything after interacting with the protocol, as compared to after interaction with the trusted party. On the other hand, it is possible to have protocols that are secure and correct, separately, but still allow malicious parties to achieve things that should not be possible when interacting with the trusted third party.

We assume that the parties are *honest but curious*, meaning that they will follow the agreed-upon protocol but will try to learn as much as possible from the data-flow between the two parties. Put another way, one party is willing to trust the other party but is concerned that virus attacks on the other party will reveal the information. Finally, in complexity, one shows the computational and communication complexity of the secure algorithm.

3. CRYPTOGRAPHIC BUILDING BLOCKS

We give the necessary cryptographic building blocks we need to derive the main protocols of the paper. For lack of space, and due to their standard nature, we provide only informal definitions.

3.1. Preliminaries

We will assume that computations are done over a prime field $F = Z_p$, for a proper choice of p. If the inputs are rational numbers, we first scale them so we can represent them by integers. Let s be an integer such that any intermediate value of the computation is within the range $[-s, s]$, and choose a prime $p > 2s$.

For correctness, let $x \in [-s, s]$ be the result of some computation, and assume that Alice and Bob each hold a random share of x, such that s_A, s_B are random and uniformly distributed in F, and $s_A + s_B \pmod{|F|} \equiv x$. Note that because $x < |F|$, we get that $x \pmod{|F|} = x$, and thus $s_A + s_B \pmod{|F|}{=}x$.

Because the value of x is in the range $[-s, s]$ and $|F| > 2s$, we can distinguish negative values from positive ones: negative values are congruent to the range $[p - s - 1, p - 1]$ whereas non-negative numbers are congruent to $[0, p-1]$ (and similarly, it's easy to compare numbers).

Most of the protocols we present give output to both Alice and Bob, such that Alice and Bob get private shares that are uniformly random in F, and their sum is congruent to some desired output. Thus, neither Alice nor Bob learn anything from the output of the protocol (since it is just a random element in F). However, if both parties are honest, Bob can send Alice his private share, and Alice can compute the desired result.

We design our protocols such that the inputs to the protocols are also given as private shares. This approach allows us to compose these protocols while keeping them private.

3.2. Oblivious Transfer

Oblivious Transfer is a cryptographic tool that allows Alice to choose one element from a database of elements that Bob holds, learning this element without revealing to Bob which element was chosen and without learning anything about the rest of the elements. The most common variant of OT is $\binom{2}{1}$OT, where Bob has (v_0, v_1) and Alice chooses $b \in \{0, 1\}$, and after the OT Alice learns v_b and nothing else, and Bob learned nothing. OT can be built from any enhanced trapdoor permutation (such as the RSA trapdoor permutation).

3.3. The Millionaire's Problem

Alice and Bob would like to compare and find which one has a larger number, without revealing anything else about their inputs. This problem was introduced by Yao [1] who solved it, assuming the existence of Oblivious Transfer. When the inputs are two m-bit numbers, the communication complexity of Yao's protocol is $O(m)$ $\binom{2}{1}$OT, and this can be done in constant number of rounds.

3.4. Secure Dot Product

We are using dot product to compute the distance between two points in F^d. In our case Alice owns $X \in F^d$ and Bob knows $Y \in F^d$. At the end, Alice and Bob learns a and b, respectively, such that $a + b \pmod{F} = X^T Y$. This can be solved using Yao's protocol.

3.5. Oblivious Polynomial Evaluation

Oblivious Polynomial Evaluation (OPE) [4, 5], is a special case of private computation where Alice has a private scalar input x and Bob has a private input polynomial P, and they want to privately compute $P(x)$. Observe that OPE can be naturally extended to handle multivariate polynomials $P(x_1, ..., x_N)$ and that a degree-1 multivariate polynomial degenerates to a dot product.

4. PROTOCOLS FOR PRIVACY PRESERVING PATTERN CLASSIFICATION

In this section we give a sequence of protocols for various kernel functions. In all cases the general classifier assumes the following form

$$H(\mathbf{x}) = sign(\sum_{i=1}^{N} h_i(\mathbf{x}))$$

but the kernels $h_i(\mathbf{x})$ may be of different types. All the kernels we consider assume the same structure. They involve a dot-product of the data $\mathbf{x}$, that is known only to Alice, with some weight vector $\mathbf{y}$, known only to Bob, followed by a non-linear operation such as thresholding, Gaussian, sigmoid or polynomial. Each kernel function $h_i(\mathbf{x})$ uses a different weight vector $\mathbf{y}_i$ and the classifier $H(\mathbf{x})$ is a weighted sum of kernel functions of the same type.

1. The threshold function:

$$h(\mathbf{x}) = \begin{cases} \alpha & \mathbf{x}^T\mathbf{y} > \Theta \\ \beta & \text{otherwise} \end{cases}$$

   where α, β, Θ are scalars known only to Bob. This was presented in [7] and is mentioned here for completeness only.

2. The polynomial function (see Table 1):

$$h(\mathbf{x}) = \alpha(\mathbf{x}^T\mathbf{y} + c)^d$$

   where α, c and d are scalars known only to Bob.

3. The Gaussian function (see Table 2):

$$h(\mathbf{x}) = \alpha e^{(\gamma||\mathbf{x}-\mathbf{y}||_2^2)}$$

   where α and γ is a scalar known only to Bob.

4. The sigmoid function (see Table 3):

$$h(\mathbf{x}) = \frac{\alpha}{1 + e^{(\mathbf{x}^T\mathbf{y})}}$$

   where α is a scalar known only to Bob.

In all cases, we give sub-protocols that let Alice and Bob obtain additive shares $s_{A,i}$ and $s_{B,i}$ for $h_i(\mathbf{x})$. Alice and Bob then accumulate their private shares to get $s_A = \sum_{i=1}^{N} s_{A,i}$ and $s_B = \sum_{i=1}^{N} s_{B,i}$, such that $s_A + s_B = \sum_{i=1}^{N} h_i(X)$. Then Alice and Bob use the updated Millionaire protocol to compare the sum of their values to 0, getting the sign of the classification. In addition to these four classes of widely used kernels we propose a method to approximate arbitrary kernel functions using lookup tables. In some of these protocols we use the Secure Polynomial Evaluation of [5]. For brevity, we omit security and complexity analysis.

4.1. A Note on Secure Classification

There is an inherent tension between secure multi-party computation and machine learning, in that one tries to hide his classifier and the other tries to infer what the classifier is. In the extreme case, Alice can use her secure computations with Bob in a training stage, and have Bob label examples for her, and Alice train her own classifier on these examples. This is unavoidable, and in that sense, the best we can hope for is to ensure that Bob will not learn anything about Alice's data and will not help Alice's training algorithm, other than supplying it with labeled examples.

Input: Alice has $\mathbf{x} \in F^L$
Input: Bob has a polynomial kernel function $h(\mathbf{x}) = \alpha(\mathbf{x}^T\mathbf{y} + c)^d$
Output: Alice and Bob learn private shares $r_A + r_B = h(\mathbf{x})$.

1. Alice and Bob use the inner product sub-protocol to get random additive shares of $\mathbf{x}^T\mathbf{y}$, such that $r_A + r_B = \mathbf{x}^T\mathbf{y}$.
2. Bob defines the polynomial $f(z) = \alpha(z + r_B + c)^d$
3. Alice and Bob use the OPE sub-protocol to get random additive shares of $f(r_A)$, such that $s_A + s_B = f(r_A) = \alpha(r_A + r_B + c)^c = \alpha(\mathbf{x}^T\mathbf{y} + c)$.

Table 1. Polynomial Kernel

4.2. Kernel Evaluation Using Lookup Tables

We now give a lookup table approach to kernel function evaluation, where a lookup table approximates the range of values taken by the dot product $\mathbf{x}^T\mathbf{y}$. If this range is small then it is feasible to enumerate all possible values of the vector $\mathbf{x}$, otherwise Bob and Alice can agree on "binning" of the vector X and proceed as follows. Let the classifier be of the following form

$$H(\mathbf{x}) = sign(\sum_{i=1}^{N} h_i(\mathbf{x}^T\mathbf{y_i}))$$

where $h_i(\mathbf{x}^T\mathbf{y_i})$ may be of an arbitrary kernel function not of any type discussed above and let Alice and Bob map the domain $\mathcal{D}$ into "bins" and generate the lookup table values for each "bin". A *mapping function* $b : \mathcal{D} \rightarrow [K]$ maps each $x \in$

Input: Alice has $\mathbf{x} \in F^L$
Input: Bob has a Gaussian kernel function of the form $h(\mathbf{x}) = \alpha e^{\gamma||\mathbf{x}-\mathbf{y}||_2^2}$.
Output: Alice and Bob learn private shares $t_A + t_B = h(\mathbf{x})$.

1. Bob prepares the polynomial $f_1(z) = \gamma z$.
2. Alice and Bob use the OPE sub-protocol to get random additive shares of $f(\mathbf{x}^2)$, such that $r_A + r_B = \gamma \mathbf{x}^2$.
3. Alice computes e^{r_A}, Bob computes e^{r_B}, and note that $e^{r_A} \cdot e^{r_B} = e^{r_A + r_B} = e^{\gamma \mathbf{x}^2}$.
4. Alice and Bob use the inner-product sub-protocol to get random additive shares of $\mathbf{x}$ and $-2\gamma\mathbf{y}$, such that $s_A + s_B = -2\gamma \mathbf{x}^T\mathbf{y}$.
5. Alice computes e^{s_A}, Bob computes e^{s_B}, and note that $e^{s_A} \cdot e^{s_B} = e^{s_A + s_B} = e^{-2\gamma \mathbf{x}^T\mathbf{y}}$.
6. Bob computes $e^{\gamma \mathbf{y}^2}$.
7. Bob prepares the polynomial $f_2(z_1, z_2) = e^{r_B} z_1 + e^{s_B} z_2 + e^{\gamma \mathbf{y}^2}$,
8. Alice and Bob use the OPE sub-protocol (for multivariate polynomials) to get random additive shares of $f_2(e^{r_A}, e^{r_B})$, such that $t_A + t_B = f_2(e^{r_A}, e^{r_B}) = e^{r_B} e^{r_A} + e^{s_B} e^{r_B} + e^{\gamma \mathbf{y}^2} = e^{\gamma \mathbf{x}^2} + e^{-2\gamma \mathbf{x}^T\mathbf{y}} + e^{\gamma \mathbf{y}^2} = e^{\gamma(\mathbf{x}-\mathbf{y})^2}$

Table 2. Gaussian Kernel

Input: Alice has $\mathbf{x} \in F^L$
Input: Bob has a sigmoid kernel of the form $h(\mathbf{x}) = \frac{\alpha}{1+e^{\mathbf{x}^T\mathbf{y}}}$
Output: Alice and Bob learn private shares $t_A + t_B = h(\mathbf{x})$.

1. Alice and Bob use the OPE sub-protocol to privately compute the inner product of $\mathbf{x}$ and $\mathbf{y}$, and get random additive shares $s_A + s_B = \mathbf{x}^T\mathbf{y}$
2. Alice computes e^{s_A}, Bob computes e^{s_B}
3. Bob selects r_B at random, defines the polynomial $f(z) = r_B(1 + z \cdot e^{s_B})$
4. Alice and Bob use the OPE sub-protocol to privately compute $f(e^{s_A})$ and get random additive shares $u_A + u_B = r_B(1 + e^{\mathbf{x}^T\mathbf{y}})$
5. Bob sends u_B to Alice. Alice now has $v_A = r_B(1+e^{\mathbf{x}^T\mathbf{y}})$.
6. Alice defines the polynomial $g(z) = \frac{z}{v_A}$.
7. Alice and Bob use the OPE sub-protocol to privately compute $g(\alpha r_B)$, and get the random additive shares $t_A + t_B = \frac{\alpha \cdot r_B}{r_B(1+\mathbf{x}^T\mathbf{y})} = \frac{\alpha}{1+e^{\mathbf{x}^T\mathbf{y}}}$.

Table 3. Sigmoid Kernel

$\mathcal{D}$ into an integer in the range $[1, K]$ (which can be thought of as the "bin index"). We also define $b^{-1} : [K] \to \mathcal{D}$ that maps each bin index into one element of $\mathcal{D}$ which we call the "bin representative', such that for any k, we have $b(b^{-1}(k)) = k$. See Table 4.

Input: Alice has $\mathbf{x} \in F^L$
Input: Bob has a general kernel $h(\mathbf{x}^T\mathbf{y})$.
Output: Alice and Bob learn private shares of the approximation $r_A + r_B \approx h(\mathbf{x}^T\mathbf{y})$.

1. Alice and Bob use OPE to obtain private shares s_A, s_B such that $s_A + s_B = \mathbf{x}^T\mathbf{y}$.
2. Bob chooses a value r_B at random.
3. Bob constructs a table T with K entries; $T(k) = h(b^{-1}(k) + s_B) - r_B$.
4. Alice uses $\binom{K}{1}$OT to select

$$\begin{aligned} r_A &= T(b(S_A)) \\ &= h(b^{-1}(b(S_A)) + s_B) - r_B \\ &\approx h(\mathbf{x}^T\mathbf{y}) - r_B \end{aligned}$$

Table 4. General Kernel using Lookup Table

5. CONCLUSIONS

We presented several protocols for Secure data Classification that allow two parties to evaluate a particular classifier on a particular data point without revealing information, other than the final outcome, to either party. In the future we plan to investigate ways to accelerate the protocols either by relying on the large body of literature on accelerating secure protocols or by taking advantage of domain specific knowledge. In addition we are exploring ways to apply Secure Multiparty Computations to general image processing tasks that might benefit from a secure protocol.

6. REFERENCES

[1] A. C. Yao, "Protocols for secure computations," in *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, Chicago, 1982, pp. 160–164, IEEE.

[2] Oded Goldreich, Silvio Micali, and Avi Wigderson, "How to play any mental game or a completeness theorem for protocols with honest majority," in *ACM Symposium on Theory of Computing*, 1987, pp. 218–229.

[3] O. Goldreich, *Foundations of Cryptography: Volume 1, Basic Tools*, Cambridge University Press, New York, 2001.

[4] M. Naor and B. Pinkas, "Oblivious polynomial evaluation," in *Proc. of the 31st Symp. on Theory of Computer Science (STOC)*, 1999, pp. 254–254.

[5] Y.C. Chang and C.J.Lu, "Oblivious polynomial evaluation and oblivious neural learning," in *Advances in Cryptology — (ASIACRYPT 2001)*. 2001, LNCS, Springer-Verlag.

[6] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *Advances in Cryptology — (CRYPTO 2000)*, 2000.

[7] S. Avidan and M. Butman, "Blind vision," in *European Conference on Computer Vision*, 2006, pp. 1–13.