Information leakage analysis of inner-product functional encryption based data classification

Damien Ligier
CEA LIST, France, and
IMT-Atlantique Lab-STICC, France
damien.ligier@cea.fr

Sergiu Carpov
CEA LIST, France
sergiu.carpov@cea.fr

Caroline Fontaine
CNRS/Lab-STICC and UBL, France,
and IMT-Atlantique, France
caroline.fontaine@imt-atlantique.fr

Renaud Sirdey
CEA LIST, France
renaud.sirdey@cea.fr

***Abstract*—In this work, we study the practical security of inner-product functional encryption. We left behind the mathematical security proof of the schemes, provided in the literature, and focus on what attackers can use in realistic scenarios without tricking the protocol, and how they can retrieve more than they should be able to. This study is based on the proposed protocol from [1]. We generalize the scenario to an attacker possessing n secret keys. We propose attacks based on machine learning, and experiment them over the MNIST dataset [2].**

I. Introduction

Functional Encryption (FE) [3], [4] is a recent paradigm of public key cryptography. It enables users to partially decrypt messages. The authority of the system generates secret keys associated with functions. For example, let sk_f be a secret key associated with a given function f, and ct_x be an encryption of x; the output of the decryption algorithm with sk_f and ct_x as inputs is $f(x)$ (rather than x, as usual). With an *Inner-Product Functional Encryption (IPFE)* [5], [6], secret keys are associated with inner-product functions. Let $\vec{v}, \vec{w}$ be vectors, $sk_{\vec{v}}$ be the secret key associated with the inner-product function $\langle \vec{v}, \cdot \rangle$, and $ct_{\vec{w}}$ be an encryption of $\vec{w}$. The decryption algorithm with $sk_{\vec{v}}$ and $ct_{\vec{w}}$ as inputs, outputs $\langle \vec{v}, \vec{w} \rangle$.

FE schemes enable to compute algorithms over encrypted input, just like *Fully Homomorphic Encryption (FHE)* does. One difference between them is that FE evaluation algorithm's output is unencrypted, while FHE's evaluation remains encrypted. Another difference between FE and FHE is that one can evaluate any circuit over encrypted data with a FHE system, while in a FE context algorithms that can be computed are fixed by secret keys and cannot be modified. For a given plaintext, FE schemes reveal not all of it but just a part. So, there is a possible unintentional leakage of information in a FE context, while there is by construction no such leakage in a FHE context. This leakage is independent of the cryptographic system security. In a FE system, functions (associated with the secret keys) may be computed over every plaintext, so as a consequence all plaintexts share some similarities. Those shared similaries could be used along with the leaked information to rebuild approximations of messages that have been encrypted. This is our concern in this work.

Authors of [1] proposed a privacy preserving classification algorithm based on inner-product functional encryption. By partially decrypting ciphertexts, the classification can be done without revealing original plaintexts. The authors experimented its performance over the well known MNIST database [2]. Their goal is, for example, to deal with medical data that sometimes have to observe restrictive laws concerning their open access. A privacy preserving classification scheme would be useful in this context (or for every kind of private data) and would enable to perform statistical studies without having full access to the data.

In the present work, we study the leakage of this privacy preserving classification algorithm based on Inner-Product Functional Encryption (IPFE) [1]. We consider attackers who respect the protocol, but try to get more information than what they should have access to. Our use case is the following: the malicious classifier (the attacker) knows exactly which kind of data has been encrypted, and has access to a training set. Our goal is to study the security of this particular use case. We propose attacks based on machine learning which can be used to estimate the information leakage for such a system. Note that the scenario and the attacks may be generalized to any functional encryption use case, not only IPFE. Finally, we experiment the attacks on the MNIST dataset [2], also used in [1], and we give concrete results.

II. Preliminaries And Related Work

Since this work is dealing with cryptography as well as with machine learning, we are providing in this section some needed background knowledge in both areas. We also discuss related works about privacy preserving classification and information leakage.

A. Machine Learning background

Machine learning (ML) [7] aims to automatically detec meaningful patterns in data. It provides algorithms enabling programs to "learn". The "learn" word refers to the process of converting training data into a program that can perform some task. This program represents, in a way, some expertise or knowledge. *Supervized ML* designates machine learning algorithms using, as input, a training set and its labels. Those algorithms are split into two successive parts. The first one is the *learning* phase, which analyses the inputs and learns how to make proper predictions on such kind of data. The second part is the prediction phase, where classification of new data is performed with the algorithm produced by the first phase.

When the prediction result is a discrete value, we speak about *classification* (class prediction for example), while for a continuous value, we speak about *regression* (temperature prediction for example).

Principal Component Analysis (PCA) [8] is a statistical procedure that transforms a number of correlated variables into a smaller set of uncorrelated variables, still containing most of the information in it. The transformation is performed with a simple matrix product between input variables and the generated *correlation matrix*.

Artificial Neural Networks (ANN) [9] are a computational model inspired by the biological brain, included in the supervized machine learning methods. ANNs are built with artificial neurons,*i.e.* a high level abstraction of real neurons. An artificial neuron is a function that firstly computes an inner product between its input vectors and a predefined weights vector; then, the obtained value is passed through a so called activation function. The activation function's output is the neuron output. Artificial neural networks are organized in layers of neurons, and neurons outputs from one layer are used as the next layer's inputs. *Convolutional Neural Networks (CNN)* [10], [11] are a specific type of ANN inspired by the organization of the animal visual cortex, and have demonstrated excellent performance for image processing.

The MNIST database of handwritten digits [2] is a database composed of a training set, a test set and all the corresponding labels. Images are composed of $784 = 28 \times 28$ pixels, each pixel being encoded on a byte, thus having 256 levels of grey. The digits have been size-normalized and centered in a fixed-size image. There are 10 possible labels (digits from 0 to 9). This database has been used a lot for the validation of classification algorithms. Please refer to [12], [10] for more information about machine learning studies over this dataset.

B. Cryptological background

Functional Encryption (FE) is a quite recent generalization of public-key cryptography. This paradigm adds a new party, an authority, to the two traditional asymmetric parties. The authority generates and keeps the *master secret key* MSK. This particular key is necessary to derive what is called *secret keys*, which are given to users and must remain secret. Those secret keys are associated with functions; for instance, we denote by sk_f the secret key associated with function f. Obviously, there is a *public key* also generated by the authority, which is required to encrypt messages. Let ct_x be an encryption of a message x. A user owning sk_f and ct_x can run the evaluation/decryption algorithm and get $f(x)$ as plaintext output. Hence, this is not a traditional way to decrypt, but a kind of evaluation of f over encrypted messages, with an unencrypted form at the end.

Since different secret keys are given out, a primordial security property is *collusion resistance*. This means that owning a set $\{sk_{f_1}, ..., sk_{f_n}\}$ of n secret keys and an encryption ct_x of a message x only provides access to $\{f_1(x), ..., f_n(x)\}_i$, and to nothing more. This property could be achieved either with *indistinguishability-based security* or with *simulation-based security*.

Because of the public-key setting, every constant in a secret key function is known by its owner. On the contrary, it is possible to have hidden values in secret key functions with a private-key setting. There is, of course, no public key in such systems, and the encryption algorithm takes instead as input the master secret key. This work focuses on public-key functional encryption.

The cryptographic community is looking for public-key functional encryption schemes permitting to evaluate any polynomial time computable function. Such candidates from literature [13] remain mostly of theoretical interest. Nevertheless, schemes for simpler functionalities have been proposed. For example *inner-product functional encryption* [5], [6] where messages are vectors, secret keys are associated with vectors, and the decryption of $ct_{\vec{w}}$ with the secret key $sk_{\vec{v}}$ is the inner product $\langle \vec{v}, \vec{w} \rangle$.

C. Related work

Privacy preserving classification is a complex challenge. Several approaches have been proposed, using perturbation techniques like randomization [14], [15] or condensation [16], and even using k-anonymization [17]. Our study focuses on inner-product functional encryption leakage, and to our knowledge, there is no such equivalent in the literature.

Those previous approaches are different than those using Fuly Homomorphic Encryption (FHE), as for example [18], [19]. Indeed, with FHE one delegates the computation of the classification algorithm to a server and gets an encryption of the result, while with FE the computation is also delegated to the server but the server gets the unencrypted result of the classification too. Hence, with FE the server is able to perform the classification while with FHE it cannot. They are really different settings aiming at very different use cases.

III. Attacks

In this paper, we consider an attacker who is using the information he gets by running correctly the IPFE cryptographic scheme. The attacker is the classification party, so he owns a set of n secret keys $\{sk_{\vec{v}^\ell}\}_{1 \leq \ell \leq n}$ associated with the n vectors $\vec{v}^\ell$, $1 \leq \ell \leq n$. When he gets an encryption $ct_{\vec{w}}$ of a vector $\vec{w}$, he is able to compute $\{\langle \vec{w}, \vec{v}^\ell \rangle\}_{1 \leq \ell \leq n}$ with the help of the decryption algorithm of the IPFE scheme; this gives him the class c of $\vec{w}$ according to his own classifier. There are k classes in his classifier : $1, ..., k$.

The attacker's challenge is to recover input vector $\vec{w}$ from $\{\langle \vec{w}, \vec{v}^\ell \rangle\}_{1 \leq \ell \leq n}$ and c. Because we assume that he knows c, he is able to design attacks for every class of its classification process. The attacker also knows exactly what kind of data has been encrypted, so he is able to get a valid training set. He

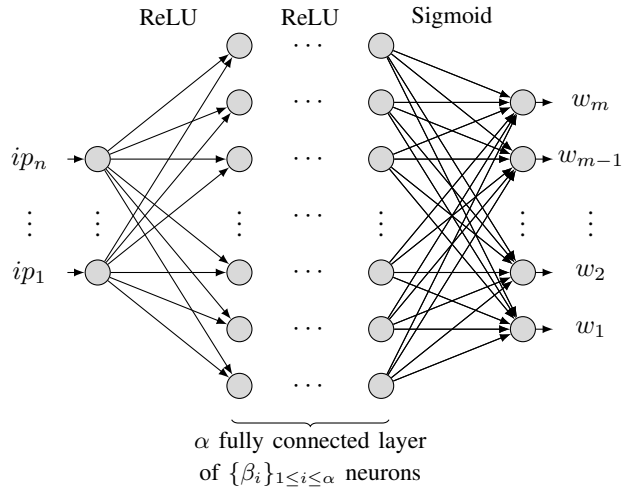

Fig. 1. Description of the ANN model used for the ANN-attack.

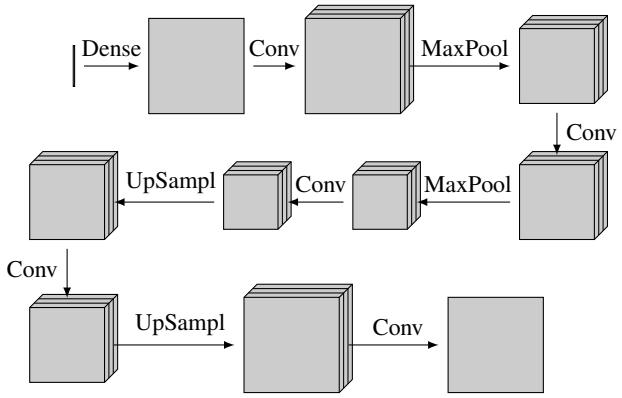

Fig. 2. Description of the CNN model used for the CNN-attack.

can then use it as he wants. We propose two types of attacks based on the use of this training set. The first one is using Principal Component Analysis (PCA) and the second one is using Artificial Neural Networks (ANN).

Although we have not done so in this paper, note that to build up the attack it is not necessary for the attacker to use the same dataset that was used to generate the classifier. Any relevant set will do.

We experimented these attacks on the MNIST dataset of handwritten digits (also used in [1]). Also, as this dataset is composed of images, we propose a variation of our ANN attack using Convolutional Neural Networks (CNN).

The results of these attacks are presented and discussed in Section IV. Those experimental results show that an attacker is often able to generate a vector relatively close to the original one, which is the secret to recover (from the attacker point of view). In what follows, we describe the proposed attacks.

A. Attack using PCA

Let $A \in \mathcal{M}_{n,m}(\mathbb{R})$ be the matrix composed of row vectors $\vec{v}^1, ..., \vec{v}^n$ and let $\vec{w}$ be a vector of c-class. Let $\vec{ip} = A \cdot \vec{w} = (ip^1, ..., ip^n)^T$ be the vector of all the inner products obtained with the FE scheme. So, for $1 \leq \ell \leq n$ we have $ip^\ell = \langle \vec{w}, \vec{v}^\ell \rangle$.

The attacker uses PCA for each class c over its training set. He gets k matrices : $\{M_{PCA,c}\}_{1\leq c\leq k} \subset \mathcal{M}_{n,m}(\mathbb{R})$. The PCA procedure ensures that $\vec{w} \backsimeq M^T_{PCA,c} \cdot M_{PCA,c} \cdot \vec{w}$ when $\vec{w}$ belongs to class c.

Then, the attacker computes matrices $\{P_c\}_{1\leq c\leq k}$ such that for all $1 \leq c \leq k$, $M_{PCA,c} \backsimeq P_c \cdot A$. This can be done with a gradient descent algorithm (optimization algorithm used to find a local minimum for a continuous function). We determine the matrix P_c by minimizing $\sum_{i,j}((P_c \cdot A - M_{PCA,c})_{i,j})^2$.

The attacker is now able to generate a vector relatively close to the original one. When he gets an encryption $ct_{\vec{w}}$ of a vector $\vec{w}$, he gets the $\vec{ip}$ vector with the IPFE decryption algorithm, and then determines its c-class and simply computes $M^T_{PCA,c} \cdot P_c \cdot \vec{ip}$:

$$
\begin{aligned}
M^T_{PCA,c} \cdot P_c \cdot \vec{ip} &= M^T_{PCA,c} \cdot P_c \cdot A \cdot \vec{w} && (1)\\
&\backsimeq M^T_{PCA,c} \cdot M_{PCA,c} \cdot \vec{w} && (2)\\
&\backsimeq \vec{w} && (3)
\end{aligned}
$$

Let us reemphasize that we get (1) because $\vec{ip} = A \cdot \vec{w}$, (2) because $M_{PCA,c} \backsimeq P_c \cdot A$, and finally (3) because $\vec{w} \backsimeq M^T_{PCA,c} \cdot M_{PCA,c} \cdot \vec{w}$.

B. Attacks using ANNs

1) Fully connected network: This attack uses an Artificial Neural Network (ANN). The network model is composed of α hidden layers of fully connected neurons. Each hidden layer is composed of β_i neurons for $1 \leq i \leq \alpha$. The last layer activation function is the sigmoid function, *rectified linear unit (ReLU)* is the activation function for the other layers. The sigmoid is defined as the function $f(x) = \frac{1}{1+e^{-x}}$, and the ReLU is defined as the function $f(x) = max(0, x)$. Figure 1 provides a visual description of this model. Note that we use the same notation as in the previous section.

Again, we split this attack into k parts, one for every class c, so there are k ANN to train. The ANN used for class c will be denoted by ANN_c. Let $\{\vec{t}_{c,i}\}_i$ be the set of c-class vectors from the training set. ANN_c training algorithm is run with $\{A * \vec{t}_{c,i}\}_i$ as ANN inputs, and $\{\vec{t}_{c,i}\}_i$ as ANN outputs.

The attacker is now able to generate a vector relatively close to the original secret one. When he gets an encryption of a vector $\vec{w}$, he gets the $\vec{ip}$ vector with the IPFE decryption algorithm. Then he determines its c-class, and simply uses its ANN_c with $\vec{ip}$ as input.

2) Convolutional network: This attack is equivalent to the previous one, except that it is using a different structure for the neural network. Since we are using the MNIST [2] database in Section IV to experiment our attacks, we consider using Convolutional Neural Networks (CNN) which tend to have better performances over images.

The convolutional neural network we use, depicted in Figure 2, has ten hidden layers, and the input is a vector of length

n, consisting of the inner products $ip_1, ..., ip_n$. The layers used are either fully connected, convolutional, max pooling or upsampling layers. This CNN is inspired by the autoencoder CNN model suggested by Keras [20] for image denoising.

The first hidden layer is a fully connected layer that converts the small vector into a two dimensional array of size 28×28.

The second layer is a convolutional layer with 32 feature maps. Each unit in each feature map is connected to a 3×3 neighborhood in the input. The size of the feature maps is the same as the input size because the area outside of the input, that is needed to compute the convolution, is padded with zeros. The activation function is the ReLU function.

The third layer is a max pooling layer. Each unit in each feature map is connected to a 2×2 neighborhood in the previous layer and its value is the maximum value of this neighborhood. The feature maps have now the size of 14×14.

The fourth layer is another convolutional layer with the same characteristics as the second layer, except that the feature maps' size is 14×14.

The fifth layer is another max pooling layer with the same characteristics as the third layer. The feature maps have now with a size of 7×7.

The sixth layer is another convolutional layer with the same characteristics as the second layer, except that the feature maps' size is 7×7.

The seventh layer is an upsampling layer. It duplicates the rows and columns of the feature maps. The feature maps have now the size of 14×14.

The eighth layer is another convolutional layer with exactly the same characteristics as the fourth layer.

The ninth layer is another upsampling layer which is exactly the same as the seventh layer. The feature maps have now the size of 28×28.

The tenth layer is the last convolutional layer with only one feature map of size 28×28. Each unit in the feature map is connected to a 3×3 neighborhood in the previous layer. The activation function is the sigmoid function.

The attack is also split into k parts, one for every c-class. Therefore, there are k CNN to train. The rest of the attack is exactly the same as the previous one.

IV. Security Analysis

The experiments were performed on a regular laptop computer with an Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz and 16GB of RAM. The Keras framework [20] was used to implement ANN and CNN attacks. The parameters used in the ANN attack are: $\alpha = 4$, $\beta_1 = 128$, $\beta_2 = 256$, $\beta_3 = 512$ and $\beta_4 = 784$. We use the binary crossentropy function as the neural network training loss function.

The *Mean Squared Error (MSE)* is used to compare the resemblance between original and recovered images. The MSE for two vectors $\vec{v}$ and $\vec{w}$ is defined as $\frac{1}{n}\sum_{i=1}^{n}(v_i - w_i)^2$. MSE measure is always non-negative, and the best value is 0.

Figures 3, 4 and 5 show the results of our attacks on some images from the MNIST [2] test set. Rows are composed, from left to right, by the original MNIST image, the result of the attack in a 30 secret key scenario, the result of the attack in a 20 secret key scenario and the result of the attack in a 10 secret key scenario. We select two images from the MNIST test set, the image in the first row is the one with the lowest MSE (in a 10 secret key scenario) and the image in the second row is the one with the highest MSE (in a 10 secret key scenario).

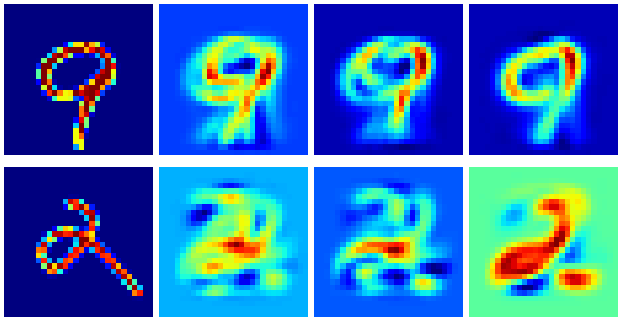

Fig. 3. Original MNIST images (first column) and PCA attack results over them, in 30 (second column), 20 (third column) and 10 (fourth columne) secret key scenario. The MNIST image in the first row is the one from the test set with the lowest MSE (0.03314) in a 10 secret key scenario, and the MNIST image in the second row is the one from the test set with the highest MSE (0.23976) in a 10 secret key scenario.

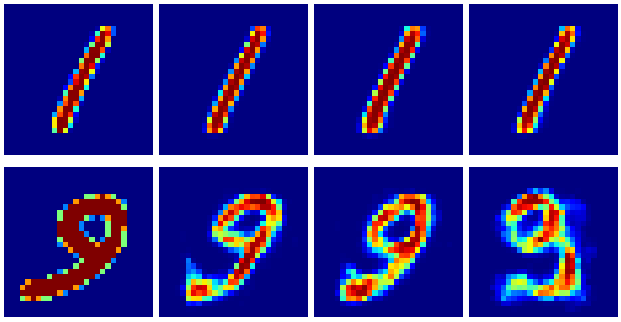

Fig. 4. Original images (first column) and fully connected ANN attack results over them, in 30 (second column), 20 (third column) and 10 (fourth column) secret key scenario. The MNIST image in the first row is the one from the test set with the lowest MSE (0.00112) in a 10 secret key scenario, and the MNIST image in the second row is the one from the test set with the highest MSE (0.15522) in a 10 secret key scenario.

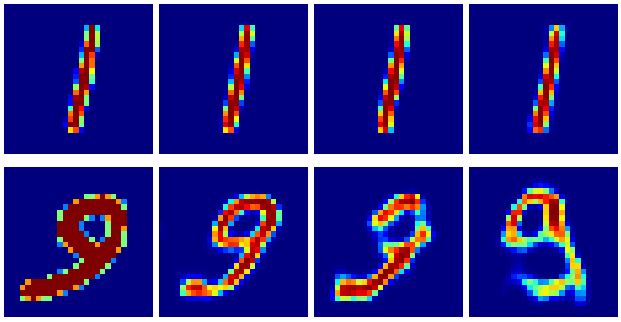

Fig. 5. Original images (first column) and CNN attack results over them, in 30 (second column), 20 (third column) and 10 (fourth column) secret key scenario. The MNIST image in the first row is the one from the test set with the lowest MSE (0.00105) in a 10 secret key scenario, and the MNIST image in the second row is the one from the test set with the highest MSE (0.19506) in a 10 secret key scenario.

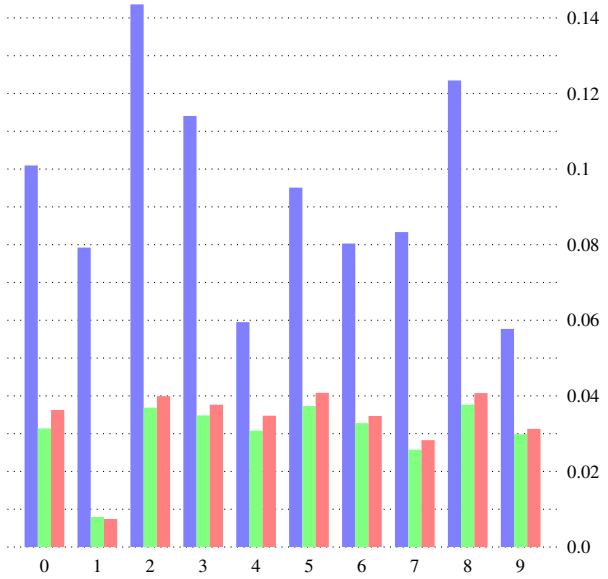

Fig. 6. Comparison between PCA (left column), fully connected ANN (middle column) and CNN (right column) attacks in the 10 inner-product context. Ordinate axis shows the average MSE for all digits in abscissa.

Visually, ANN attacks (Fig. 4 and 5) are more efficient than the PCA attack (Fig. 3). Since the experiment is performed over images, it is not surprising that the CNN attack (Fig. 5) is visually better than the fully connected ANN attack (Fig. 4). In a 30 secret key scenario, it is clear that the leakage is enough to recover an image looking exactly like the original one, even if the shape of the digit is complex.

Figure 6 shows for each digit and for each scenario (30, 20 and 10 key) the average MSE measure between the result of each of the three attacks and the original test set. We observe that the results of the fully connected ANN attack and the CNN attack are almost the same in terms of MSE, even if the fully connected ANN score is better by a small amount.

We can also notice from Figure 6, that there is an important inequality between digits regarding their MSE measure. Some of the digits reveal more about their shape than others. The digit "1" , for example, has the simplest shape and leaks the most, while curve shaped digits (like digit "2", digit "5" or digit "8") leak less and are the hardest to recover. We will now focus on the 10 secret key scenario with the CNN attack and we will show that it leaks already too much.

Figure 7 shows the MSE distribution in the CNN attack results over the test set in the 10 secret key scenario. It also shows image examples with MSE values of 0.02, 0.04, 0.06, 0.08, 0.10 and 0.12 for digit "9". Attack results with MSE under 0.04 give images really close to the originals, as we can see from the two image examples on the left. Such images (MSE under 0.04) represent 70% of the test set.

Figure 8 shows for each digit, in a descending order, the percentage of images with MSE under 0.04 with the CNN attack in a 10 secret key scenario. As mentioned before, the results depend on the digits. In this figure we can see that

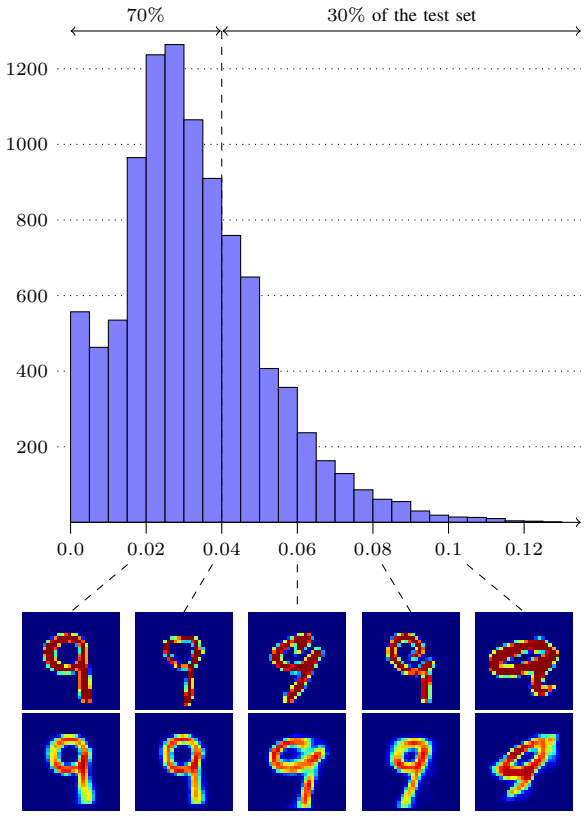

Fig. 7. Histogram of the MSE result of the CNN attack. Abscissa represent MSE mesure, and ordinate shows the number of image from the test set (total of 10,000 images). We omit the results for which the MSE is larger than 0.13, because it represents less than 0.1% of the test set.

1	7	9	4	6
98.50%	80.93%	75.61%	69.95%	68.99%

0	3	2	5	8
66.93%	64.55%	57.17%	56.05%	55.13%

Fig. 8. For each digit, percentage of images from the test with an MSE under 0.04 with the CNN attack in a 10 secret key scenario (descending order).

almost every digit "1" ($> 98\%$) are "correctly" recovered with the CNN attack, while digits "2", "5" and "8" are "correctly" recovered for more than half of the examples ($< 60\%$). In a 20 secret key scenario, the percentages are all higher than 93%, and in a 30 secret key scenario they are higher than 97%.

Ten secret keys are leaking enough to recover all digit "1" shapes, and more than 50% or the other digit shapes. The privacy preserving feature seems to be compromised in this case. Our study is not only valid in this context of classification, but as long as an inner-product functional encryption scheme is deployed and more than one key is generated, attacks like those we present can be used. The more limited the plaintexts are, the more efficient attacks are. An attacker can easily cut the plaintext space into classes, and then process as explained in the previous section to attack any ciphertext he receives. So it seems really essential to make

sure that we have a precise idea of the leakage when we use such schemes.

V. Conclusion And Future Work

In this work, we study the leakage of inner-product FE in a multi-secret key Functional Encryption context. We show that even if the underlying IPFE scheme is secure and collusion resistant, it is possible, in some cases, to use machine learning tools to extract more than what a user should get with the authorized inner-product values. Hence, when we use an IPFE scheme, we have to be careful about the number of published secret keys. There is a tradeoff between this number, the nature of the plaintexts, and the effective potential of leakage. As a result of it, the number of secret keys generated with an IPFE scheme must be carefully managed according to the number of linear equation of the secret input which it eventually provides. Another solution would be to use a FE scheme with nonlinear functionality (or, equivalently, an IPFE scheme on non linear terms), or the possibility to perform tests over ciphertexts (or even evaluating the max function in the FE scheme). But despite of a number of theoretical contributions [13] and a few recent tentatives [21], such powerful and practical FE schemes have yet to emerge. In both cases, due to the added non linearity, an ML attack will be much more difficult to perform.

References

[1] D. Ligier, S. Carpov, C. Fontaine, and R. Sirdey, "Privacy preserving data classification using inner-product functional encryption," in *Proceedings of the 3rd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, 2017, pp. 423–430.

[2] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST Database," http://yann.lecun.com/exdb/mnist/.

[3] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2005, pp. 457–473.

[4] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *Theory of Cryptography Conference*. Springer, 2011, pp. 253–273.

[5] M. Abdalla, F. Bourse, A. De Caro, and D. Pointcheval, "Simple functional encryption schemes for inner products," in *IACR International Workshop on Public Key Cryptography*. Springer, 2015, pp. 733–751.

[6] S. Agrawal, B. Libert, and D. Stehlé, "Fully secure functional encryption for inner products, from standard assumptions," 2015.

[7] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[8] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[9] B. Yegnanarayana, *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," vol. 86, no. 11. IEEE, 1998, pp. 2278–2324.

[11] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," *Artificial Neural Networks and Machine Learning–ICANN 2011*, pp. 52–59, 2011.

[12] L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L. D. Jackel, Y. LeCun, U. A. Muller, E. Sackinger, P. Simard *et al.*, "Comparison of classifier methods: a case study in handwritten digit recognition," in *International conference on pattern recognition*. IEEE Computer Society Press, 1994, pp. 77–77.

[13] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate indistinguishability obfuscation and functional encryption for all circuits," *SIAM Journal on Computing*, vol. 45, no. 3, pp. 882–929, 2016.

[14] S. Agrawal and J. R. Haritsa, "A framework for high-accuracy privacy-preserving mining," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*. IEEE, 2005, pp. 193–204.

[15] Z. Yang, S. Zhong, and R. N. Wright, "Privacy-preserving classification of customer data without loss of accuracy," in *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, 2005, pp. 92–102.

[16] C. C. Aggarwal and S. Y. Philip, "A condensation approach to privacy preserving data mining," in *International Conference on Extending Database Technology*. Springer, 2004, pp. 183–199.

[17] B. C. Fung, K. Wang, and S. Y. Philip, "Anonymizing classification data for privacy preservation," *IEEE transactions on knowledge and data engineering*, vol. 19, no. 5, 2007.

[18] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," Tech. Rep., February 2016. [Online]. Available: https://www.microsoft.com/en-us/research/publication/cryptonets-applying-neural-networks-to-encrypted-data-with-high-throughput-and-accuracy/

[19] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," Cryptology ePrint Archive, Report 2017/035, 2017, http://eprint.iacr.org/2017/035.

[20] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[21] C. E. Z. Baltico, D. Catalano, D. Fiore, and R. Gay, "Practical functional encryption for quadratic functions with applications to predicate encryption." *IACR Cryptology ePrint Archive*, vol. 2017, p. 151, 2017.